Proceedings of the Workshop on Graph Theory and Applications, Porto Alegre, Brasil (2006)

Debugging Equation-Oriented Process Simulator Models using Graph Theory

R. P. Soares A. R. Secchi

March 30, 2007

Abstract

In the field of process simulation, a movement from modular oriented, which is currently the most widely used technique, to Equation-Oriented (EO) is clear. One of the key advantages of the EO approach is that the effort spent in model development is minimized by reusing the models in several different tasks, for instance: model linearization, simulation, optimization, and data reconciliation. EO tools support the implementation of models to a large extent, however there is almost no assistance in the model development process. In this work the currently available methods for detecting inconsistencies in systems of equations coming from both static and dynamic models are briefly reviewed. For the dynamic case a new algorithm, based on graph theory, is proposed. This algorithm is scalable for large problems and is a promising diagnosis tool to spread the usage of EO dynamic simulators. Finally, applications in test cases are used to illustrate the debugging techniques.

1 Introduction

Process engineering is found in a vast range of industries, such as chemical, petrochemical, food, pharmaceutical, and biotechnological. It focuses on the design, operation and maintenance of these processes. In doing so, the process engineer usually make use of specifically designed software tools - process simulators.

The current process simulators may roughly be classified into two groups: modular and equation-oriented [1]. A typical window of a process simulator can be seen in Figure 1.

In modular tools the models of process units are precoded in a programming language by a modelling expert and incorporated in a model library. The end user selects the models from the library and connects them to form the plant model. The incorporated chemical engineering knowledge as well as the model structure are largely fixed and not accessible [2].

In equation-oriented (EO) or equation-based implementations the equipment models are written in some descriptive or *modelling* language and usually are opened for inspection and extension. These models share with the plant model their equations and not only their numerical solution. As a consequence, the implementation of unit models is independent of any particular application or algorithm that may be used for their solution. Due these reasons, the EO technology has been demonstrated as effective in application to a wide range of

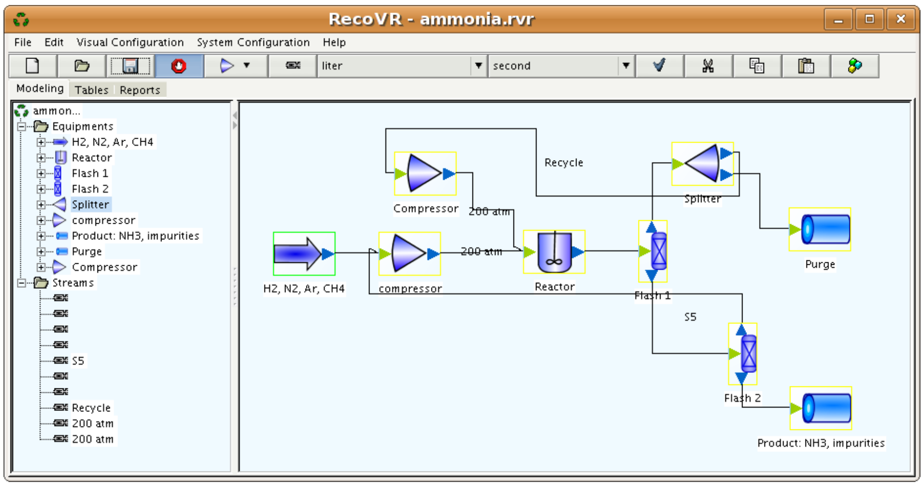

Figure 1: Typical process simulator Screenshot.

problems as model linearization, simulation, parameter estimation, and data reconciliation all using a single set of models [3]. Recognition of potential benefits of EO technology has led to the development of several tools [4]. Examples of implementations are SpeedUp [5], gPROMS [6], ASCEND [3], ABACUS II [7], and EMSO [8].

On the other hand, EO unit models do not carry any information about the valid set of specifications or initial conditions. The user needs to have at least a minimal knowledge of the model internals in order to estimate which variables can be fixed to close the degrees of freedom. Further, the user needs to know not only which variables can or cannot be fixed but if a given set of variables is valid because some variables cannot be fixed at the same time. For dynamic models the situation can be even worst because the same problems appear for the initial conditions. From the end user perspective, these aspects makes EO simulators harder to use. Actually, modelling has been the bottleneck to the widespread use of EO tools in the industrial practice and not numerical algorithms. Another problem is that the unit operation library developer has almost no assistance in fixing problematic models.

In this work, methods for diagnosing ill-posed models coming from EO tools are reviewed and extended. Making an analogy with software development, the methods which aid to detect and remove problems of the models are called *debugging*. Basically, the objective is to remove some of the deficiencies of the EO technology by answering the following questions:

- For an under-constrained model which variables can be fixed or specified?
- For an over-constrained model which equations should be removed?
- For dynamic simulations, which variables can be supplied as initial conditions?
- How to report the inconsistencies making it easy to fix?

In other words, debugging methods need to go beyond degrees of freedom and index analysis.

2 Nonlinear Systems

A general system of nonlinear nonlinear algebraic (NLA) equations can be represented by:

$$F(y) = 0 \tag{1}$$

where F is the function vector and y are the variables. System (1) has m equations in terms of n variables and appears in the solution of steady-state simulations of EO simulators. If all equations are linear, the system (1) can be reduced to:

$$Ay = b \tag{2}$$

The solution of (1) can be obtained using a Newton-like method. Such methods iteratively solve problems like (2), where A is some approximation of the Jacobian F_y. This means that F_y needs to be invertible along the solution path.

Most codes for matrix inversion are able to detect numerical singularities before failing. Some more robust codes can check if the matrix is structurally singular before actually try to solve the problem, *e.g.* [9]. However, these codes cannot be used to diagnose the source of the problem because they stop the analysis when the singularity is detected. In the following sections, currently available methods, based on graph theory, capable to detect and report the source of structural singularities for NLA systems are briefly reviewed.

2.1 Graph Theory Basics

A *graph* consists in a pair $G = (V, E)$ of sets satisfying $E \subseteq [V]^2$. Thus the elements of E are pairs of the elements of V. The elements of V are the *vertices* (or *nodes*, or *points*) of the graph G, and the elements of E are its *edges* (or *lines*).

In Figure 2 a typical graph is drawn. How the nodes and edges are drawn is considered irrelevant: all that matters is the information about which pairs of vertices form an edge and which do not. A good source for graph theory concepts is the book due to [10].

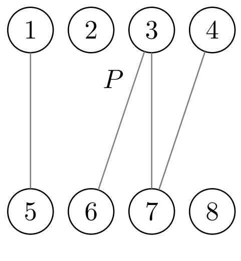

Figure 2: The graph on $V = \{1 \ldots 8\}$ with edge set $E = \{\{1,5\},\{3,6\},\{3,7\},\{4,7\}\}$.

In the graph in Figure 2, the subgraph $P = \{\{6,3\},\{3,7\},\{7,4\}\}$ is called a *path*. The number of edges of a path is its *length*. A path is often referred as the natural sequence of its vertices, say $\{6-3-7-4\}$. In this case, the path P *links* 6 and 4.

A graph $G = (V, E)$ is *bipartite* if V admits a partition in two classes such that every edge has its ends in a different class: vertices in the same partition must not be adjacent. In this work the partition sets of V in a bipartite graph are called V_e and V_v, then $V = V_e \cup V_v$.

A set M of independent edges in a graph is called a *matching.* The set of edges in M is such that no two of them have a vertex in common. The vertices included in M are then called *matched* or *covered* (by M); vertices not incident with any edge of M are *unmatched*, *not covered*, or *exposed.*

When working with matching, an interesting problem is the *maximum maching* problem. The maximum maching M^{max} of a graph G is a matching with as many edges as possible. If a matching leaves no vertex exposed, it is called *complete* or *perfect matching.* Obviouslly, a perfect matching can exist only if V_e and V_v consist of the same number of vertices.

The classic solution for a maximum matching problem is to start by considering an arbitrary matching M in G. An arbitrary matching for the graph in Figure 2 it is shown in Figure 3 (bold edges). The path $P = \{6 - 4\}$ contains, alternately, edges covered and exposed by M. A path like P is called an *alternating path* with respect to M. An alternating path that ends in an unmatched vertex is called an *augmenting path*, as the path P in Figure 3.

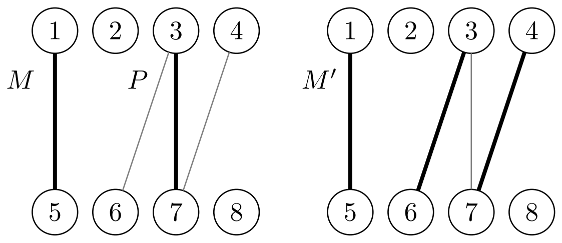

Figure 3: Augmenting the matching M by the augmenting path P.

Augmenting paths play an important role in the pratical search for a maximum matching, because they can be used to increase the number of edges in M. If one creates a new matching M' with all unmatched edges of an augumenting path P, then the set of matched vertices is increased by two, as can be seen in Figure 3.

In fact, if one starts with any matching and keep applying augmenting paths until no further such improvement is possible, a maximum matching is found. Algorithm 1 represents a classical implementation of a maximum macthing algorithm.

Basically, it starts from an empty matching and sequentially tries to find an augmenting path (line 4) until no further improvement is possible. The depth-first search Algorithm 2 is used iteratively by Algorithm 1 to augment the matching. Algorithm 1 is also known as the *classic* or *primitive algorithm* [12].

2.2 NLA Systems as Graphs

When analyzing a system of equations the equation-variable relationship is very important. For this combinatorial problem the concept of bipartite graphs can

Algorithm 1 Pseudocode for the classic maximum matching algorithm M^{max} for a given $G = (V_e \cup V_v, E)$.

```
MaxMatching
input: $(G = (V_e \cup V_v, E)$
output: $M$
1: $M \leftarrow \emptyset$
2: $flag \leftarrow$ true
3: for $v_e \in V_e$ do
4:     if not $AugmentMatching(G = (V_e \cup V_v, E), M, v_e)$ then
5:         $flag \leftarrow$ false
6:     end if
7:     uncolour $V_e$
8: end for
9: return $flag$
```

be used, where the partition classes V_e and V_v are the equation and variable nodes, respectively.

Using this notation the NLA system of equations (3) can be drawn as the bipartite graph shown in Figure 4 [11].

$$\begin{aligned}
&f_1(x_1) = 0\\
&f_2(x_1, x_2) = 0\\
&f_3(x_1, x_2) = 0\\
&f_4(x_2, x_3, x_4) = 0\\
&f_5(x_4, x_5) = 0\\
&f_6(x_3, x_4, x_5) = 0\\
&f_7(x_5, x_6, x_7) = 0
\end{aligned} \tag{3}$$

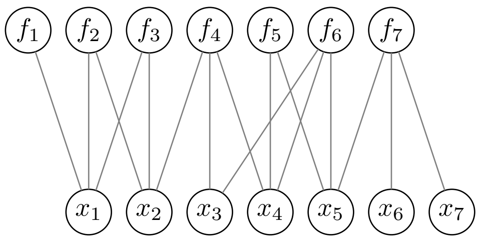

Figure 4: Graph for the NLA system (3).

As can be seen in Figure 4 the values or form of the equations in (3) are irrelevant, only the equation-variable relationship is considered. This is the essence of the structural analysis.

2.3 Debugging NLA Systems

When checking for problems in NLA systems the first check should be a degrees of freedom analysis. But systems with zero degrees of freedom still can be inconsistent, as in the case of system (3).

Algorithm 2 Pseudocode to augment a matching M for a graph $G = (V_e \cup V_v, E)$, starting at the vertex v_e.

```
AugmentMatching
input: $G = (V_e \cup V_v, E)$, $M$, $v_e$
output: $M$
1: colour $v_e$
2: if exists $\{v_e, v_v\} \in E$ and $\{v_e, v_v\} \ni M$ then
3:     $M \leftarrow M \cup \{v_e, v_v\}$
4:     return true
5: end if
6: for all $\{v_e, v_v\} \in E$ do
7:     if exists $\{v_{e2}, v_v\} \in M$ and $v_{e2}$ not colored then
8:         if $AugmentMatching(G = (V_e \cup V_v, E), M, v_{e2})$ then
9:             $M \leftarrow M \cup \{v_e, v_v\}$
10:            return true
11:        end if
12:    end if
13: end for
14: return false
```

The structural singularity of a NLA system can be easily checked using a maximum matching algorithm. One of the maximum matching associations for system (3) can be seen in Figure 5. In this figure, the edges which are part of the matching are shown in *bold* and nodes not covered by the association are *marked*.

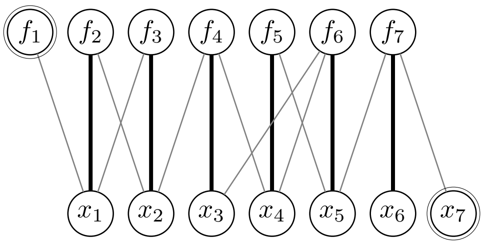

Figure 5: Maximum matching for system (3).

If a maximum matching association includes all variables and all equations (a perfect matching) then the system is structurally non-singular. Otherwise the system is structurally singular and any trial to numerically solve the system will fail.

On the other hand, the maximum matching checking cannot be used as a tool for fixing the problem because the source of the problem is still obscured. One step further can be achieved using the DM decomposition [13]. This method canonically decomposes any maximum matching of a bipartite graph into three distinct parts: over-constrained G^+, under-constrained G^-, and well-constrained G^w.

In a nutshell, the DM decomposition starts from any maximum matching M^{max} and constructs different partitions, as follows:

- G^+ the set of all alternating paths in G with respect to M^{max} starting from exposed equation nodes
- G^- the set of all alternating paths in G with respect to M^{max} starting from exposed variable nodes
- $G^w = G \setminus (G^+ \cup G^-)$

Sometimes the DM decomposition can be better understood by converting the undirected graph G into a directed graph using the information of a maximum matching M^{max}, as follows:

- exchange all edges not covered by M^{max} with directed edges oriented from the equation nodes to the variable nodes
- exchange all edges matched by M^{max} with directed edges in the opposite orientation (from variables to equations)

Using the directed graph, the over-constrained partition G^+ is determined by all vertices that can be reached, in the direction of the directed edges, starting from exposed equation nodes. The application of this procedure to the graph shown in Figure 5 can be seen in Figure 6. It should be noted that, for the determination of the under-constrained partition, the orientation of the edges needs to be inverted and the search starts from the exposed variable nodes (also shown in Figure 6).

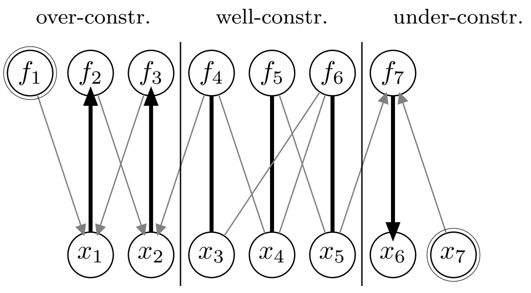

Figure 6: DM decomposition for system (3).

From Figure 6 a debugging tool can conclude that one of the equations $\{f_1, f_2, f_3\}$ needs to be removed and one additional equation involving x_6 or x_7 needs to be added. From the end user perspective of an EO tool, one conclusion could be: x_7 should be specified and f_1 should be removed or used to evaluate a wrongly specified variable.

3 Differential-Algebraic Systems

Differential-Algebraic Equation (DAE) systems arise naturally when dealing with dynamic simulation in EO tools. A general DAE system can be represented by:

$$F(t, y, y') = 0 \tag{4}$$

where t is the time and y' are the derivatives of y with respect to t.

The *index* of DAE systems is of great importance in the numerical classification of a DAE system [14]. Historically the analysis of this kind of problem was limited to degrees of freedom and index analysis, see [15, 16, 17, 18].

Today, the algorithm developed by [16] is the most widely used structural technique for analysis of DAE problems. The main objective of that work was to determine the number of initial conditions required for consistent initialization, in other words, to check the number of *dynamic* degrees of freedom. Unfortunately, the DM decomposition cannot be applied to the Pantelides' algorithm resulting graph.

In this work a new algorithm for the analysis of DAE systems is introduced. This algorithm is very similar to the algorithm proposed by [16] but its resulting graph is suitable for a DM decomposition.

3.1 New DAE Analysis Algorithm

DAE systems also can be represented as bipartite graphs. But in the dynamic case there are two new concepts: the derivatives of the variables are also considered and the equations can be differentiated inserting new elements into the graph. In order to illustrate these concepts, consider the following system of equations:

$$\begin{aligned} f_1(x_1', x_2) &= 0 \\ f_2(x_2) &= 0 \end{aligned} \tag{5}$$

where x_1' and x_2' are the time derivatives of the variables x_1 and x_2.

Figure 7 shows the graph representation for the system (5), but with the second equation *structurally differentiated* with respect to time. As can be seen in this figure, the variables are classified as algebraic (in *gray*) and differential (in *black*). It should be noted that, although x_1 does not appear in system (5), it is considered on the graph. Further, it is considered that the time derivative of f_2 involve only x_2' and not x_2 (structural differentiation).

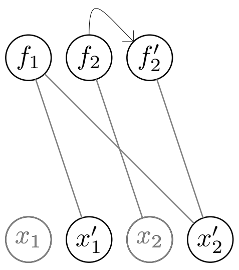

Figure 7: Graph for system (5) with the second equation differentiated.

In this work the Algorithm 3 is proposed for the analysis of DAE systems. This algorithm expects as input a graph G representing the system of equations and returns a new graph (with possibly more equations and variables) and its maximum matching M.

The algorithm Algorithm 3 is very similar to the maximum matching algorithm (Algorithm 1). Basically, it will loop for all equations (line 2) trying to augment the matching (line 3), but **ignoring** the algebraic variables. If such match is not possible it tries to find a match considering **all** variables (line 6). If

Algorithm 3 Pseudocode for the new DAE analysis algorithm for a given graph $G(V_e, V_v, E)$.

```
DAEAnalysis
input: $G = (V_e, V_v, E)$
output: $G = (V_e, V_v, E)$, $M$
 1: $M \leftarrow \emptyset$
 2: for $v_e \in V_e$ do
 3:   if not $AugmentMatching2(G = (V_e, V_v, E), M, v_e,$ false$)$ then
 4:     mark all colored $v_k \in V_e$
 5:     uncolour $V_e$
 6:     if not $AugmentMatching2(G = (V_e, V_v, E), M, v_e,$ true$)$ then
 7:       return false
 8:     end if
 9:     diff all marked $v_k \in V_e$
10:   else
11:     uncolour $V_e$
12:   end if
13: end for
14: return true
```

the match including all variables is not possible then the system is structurally singular.

The Algorithm 4 (upon which the Algorithm 3 is based) resembles Algorithm 2. It tries to augment the current match M by including a new matching for the given equation node v_e, but it has an additional argument specifying if the algebraic variables should be included or not in the valid node set.

If the last argument of Algorithm 4 is *false*, then it ignores the **algebraic** variables, otherwise **all** variables are considered (line 3 and line 6 of Algorithm 3, respectively). If the matching was augmented by the algorithm it returns *true*, otherwise it fails returning *false*.

When Algorithm 4 fails, the subset of the equations V_e reached by alternating paths starting at v_e is colored. When this happens in line 3 of Algorithm 3, all *colored* equations are *marked*. Equations *marked* at that step are structurally differentiated with respect to time in line 9. When equations are differentiated, new equations are added to the system and possibly new variables.

In order to clarify the application of the Algorithm 3, consider again the system of equations (5) and its graph in Figure 7. For the first equation f_1, the line 3 will find the match $\{f_1 - x_1'\}$. For f_2, the line 3 will fail (there is no possible match when the algebraic variables are ignored). As a consequence, f_2 will be marked at line 4. In line 6 (when **all** variables are considered), the match $\{f_2 - x_2\}$ is found. Figure 8 shows the graph and matching at this point.

At line 9 the new equation f_2' will be added. Finally the match $\{f_2' - x_2'\}$ if found at line 3 and the algorithm finishes. The final matching can be seen in Figure 9

The main advantage of the new algorithm is that its final association is suitable for a DM decomposition. For instance, the under-constrained partition will reveal all variables which can be supplied as initial conditions. Taking the system (5), the under-constrained partition will include only x_1. Using this

Algorithm 4 Modification of Algorithm 2 for the new DAE analysis algorithm.

```
AugmentMatching2
input: $G = (V_e \cup V_v, E)$, $M$, $v_e$, $alg$
output: $M$
 1: colour $v_e$
 2: if exists $\{v_e, v_v\} \in E$ and $\{v_e, v_v\} \ni M$ and $v_v$ is elegible then
 3:     $M \leftarrow M \cup \{v_e, v_v\}$
 4:     return true
 5: end if
 6: for all $\{v_e, v_v\} \in E$ do
 7:     if exists $\{v_{e2}, v_v\} \in M$ and $v_{e2}$ not colored and $v_v$ is elegible then
 8:         if $AugmentMatching2(G = (V_e \cup V_v, E), M, v_{e2}, alg)$ then
 9:             $M \leftarrow M \cup \{v_e, v_v\}$
10:             return true
11:         end if
12:     end if
13: end for
14: return false
```

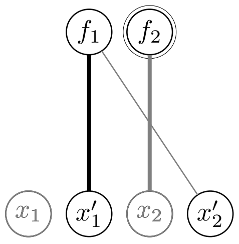

Figure 8: Graph for system (5) after the first three steps of the algorithm.

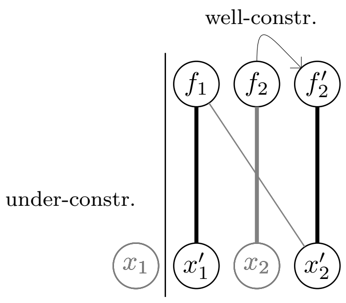

Figure 9: Graph for system (5) after the analysis.

information, an EO tool can tell to the end user that the only option for this model is to supply an initial value for x_1. All other variables $\{x'_1, x_2, x'_2\}$ are discarded from the initial conditions candidates.

It should be noted that the differentiations executed by the algorithm are only structural. This kind of differentiation can be implemented very efficiently and do not depend on the actual form of the equations.

The proposed algorithm can be applied without modifications to analyze high-index DAE systems. The equations differentiated by the algorithm can also be used to generate an index-reduced system, but index reduction is out of the scope of this paper.

4 Real Applications

In the previous sections very simple examples were used to illustrate the presented algorithms. In this section it is presented how the debugging techniques scale for larger and more complex problems.

4.1 Computation Time

In order to check how the new algorithm for DAE analysis performs for large scale problems a dynamic model for distillation processes was analyzed. This model has mass and energy balances for each tray besides thermodynamics and hydrodynamics equations.

For the case of the separation of isobutane from a mixture of 13 compounds in a 40-tray column the number of variables is almost 4,000. The computational time required to analyze the dynamic model of this system with different numbers of trays can be seen in Table 1.

Table 1: Time to analyze the dynamic model of a distillation column varying the number of trays.

Trays	Variables	Time (s)	Time $/N^2$ (s $\cdot 10^9$)
20	2157	0.04	9.46
40	3877	0.14	9.58
80	7317	0.52	9.79

The results shown in Table 1 were obtained in a Pentium M 1.70 GHz PC with 2 Mb of cache memory running Ubuntu Linux version 6.06. All analyzed cases are well-posed models - this test was not used to test the debugging power of the algorithm. As can be seen in the table, the performance is approximately quadratic bounded as are the majority of the solution methods.

Another good result is that the time required by the analysis is very acceptable for user interaction. With the current available hardware, this time is of the order of one second for systems with 10,000 variables. Moreover, the algorithm can be applied incrementally adding new equations and variables as the user interacts with the modelling environment. This fact can brake-up the analysis time, making the software more responsive to the end user.

4.2 Debugging Power

The algorithms presented in the previous sections can be used to discover problems in systems of equations. In order to show how unstable these algorithms can be regarding the system being analyzed, consider the ammonia synthesis process as shown in Figure 10 [19].

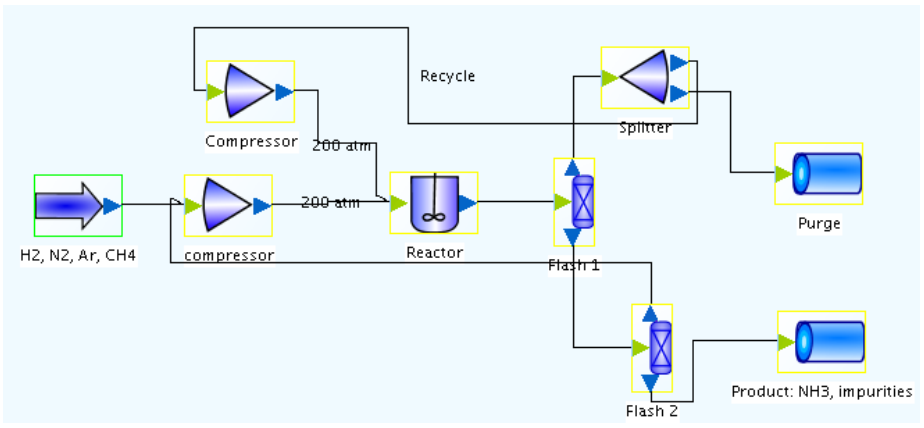

Figure 10: Ammonia synthesis process.

A static model with 134 variables for the process in Figure 10 was constructed. If all specifications are supplied correctly the maximum matching algorithm finishes with a perfect matching. But if one specification is missing, for instance the process feed flow rate, then the under-constrained partition will involve 96 variables. This means that the well-constrained partition covers only about 30% of the variables.

Unfortunately the majority of the models have a similar behavior: in the case of a singularity the number of fixing options is quite large. No doubt this is a weak point of the presented debugging techniques. In order to reduce this deficiency the fixing options could be ranked by heuristic rules that present the more meaningfull options first. These rules are currently being developed.

5 Conclusions

In this work, techniques which aid in the location and removal of inconsistencies of models coming from Equation-Oriented simulators were called debugging methods. For static models (NLA systems) mature methods were found in the literature and briefly reviewed.

For the dynamic case (DAE systems) a very less mature situation is found. Historically, the analysis of such systems was limited to degrees of freedom and index analysis. A new algorithm, based on graph theory, for structural analysis of DAE systems was proposed. The key advantage of this algorithm is that it can be used for debugging purposes. This brings the analysis of DAE systems to the same level as NLA systems.

All methods presented in this work were implemented in C++ and are freely available from the author. The major deficiency of the presented methods is the large number of fixing options when a singular model is found. In order to reduce this deficiency, heuristic rules for ranking the fixing options are being studied. Furthermore, these codes are being incorporated in the EMSO [8] process simulator.

References

[1] J. F. Boston, H. I. Britt, and M. T. Tayyabkhan. Tackling tougher tasks. *Chemical engineering progress*, 89(11):38–49, 1993.

[2] W. Marquardt. Trends in computer-aided process modeling. *Computers & Chemical Engineering*, 20(6):591–609, 1996.

[3] B. A. Allan. *A More Reusable Modeling System.* PhD thesis, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA, USA, 1997.

[4] Vicente Rico-Ramirez. *Representation,analysis And Solution Of Conditional Models In An Equation-Based Environment.* PhD thesis, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA, USA, 1998.

[5] C. C. Pantelides. Speedup-recent advances in process simulation. *Computers & Chemical Engineering*, 12(7):745–755, 1988.

[6] M. Oh and C. C. Pantelides. A modelling and simulation language for combined lumped and distributed parameter systems. *Computers & Chemical Engineering*, 20:611–633, 1996.

[7] J. E. Tolsma, J. Clabaugh, and P. I. Barton. ABACUSS II: Advanced modeling environment and embedded simulator, 1999.

[8] R. P. Soares and A. R. Secchi. EMSO: A new environment for modelling, simulation, and optimisation. In *ESCAPE 13th*, volume 1, pages 947–952. Elsevier Science Publishers, 2003.

[9] Patrick Amestoy, Enseeiht-Irit, Timothy A. Davis, and Iain S. Duff. Algorithm 837: Amd, an approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.*, 30(3):381–388, 2004.

[10] Reinhard Diestel. *Graph theory.* Springer-Verlag, New York, 2 edition, 2000.

[11] Peter Bunus. *Debugging and Structural Analysis of Declarative Equation-Based Languages.* PhD thesis, Department of Computer and Information, Science Linkpings Universitet, Linkping, Sweden, 2002.

[12] H. Saip and C. Lucchesi. Matching algorithms for bipartite graph. Technical Report DCC-03/93, Departamento de Cincia da Computao, Universidade Estudal de Campinas, Campinas, Brazil, 1993.

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

[14] K. E. Brenan, S. L. Campbell, and L. R. Petzold. *Numerical Solution of Initial-Value Problem in Differential-Algebraic Equations.* North-Holland, New York, 1989.

[15] I. S. Duff and C. W. Gear. Computing the structural index. *SIAM Journal on Algebraic and Discrete Methods*, 7(4):594–603, 1986.

[16] C. C. Pantelides. The consistent initialization of differential-algebraic systems. *SIAM J. Sci. Stat. Comp.*, 9(2):213–231, March 1988.

[17] R. Bachmann, L. Brüll, T. Mrziglod, and U. Pallaske. On methods for reducing the index of differential algebraic equations. *Computers & Chemical Engineering*, 14(11):1271–1273, 1990.

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

[19] L. T. Biegler, I. E. Grossmann, and A. W. Westerberg. *Systematic Methods of Chemical Process Design.* Prentice Hall International Series in Phisical and Chemical Engineering Sciences, 1997.