University of Veszprém
Department of Computer Science

DIPLOMA THESIS

Algorithms for investigating the effect of simplification assumptions on process models

Szabolcs Rozgonyi

Supervisor: Prof. Katalin Hangos
2003.

Alulírott Rozgonyi Szabolcs, diplomázó hallgató, kijelentem, hogy a diplomadolgozatot a Veszprémi Egyetem Számítástudomány Alkalmazása tanszékén készítettem mérnök-informatikus diploma (master of engineering in information technology) megszerzése érdekében.

Kijelentem, hogy a diplomadolgozatban foglaltak saját munkám eredményei, és csak a megadott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem azt, hogy a diplomadolgozatban foglalt eredményeket a Veszprémi Egyetem, valamint a feladatot kiíró szervezeti egység saját céljaira szabadon felhasználhatja.

2003. május. 8. Rozgonyi Szabolcs

I would like to express my thanks to my supervisor Prof. Katalin Hangos for her vast number of help both in professional and in personal sense.

In addition I would like to thank the immeasurable love, help and patience to my mother who ensured the tranquil conditions for me to study.

Abstract

The effect of model simplification assumptions on the differential index of DAE (differential-algebraic equation) process models is investigated in this thesis. A process model is considered as the set of model equations in the form of a DAE system together with a suitable specification forming a zero degree of freedom model. Such a model is described by its equation-variable graph. The effect of model simplification assumptions is analyzed by describing them as graph transformations acting on the structure graph of the process model.

An efficient incremental graph-theoretic algorithm is proposed to follow the changes in the variable-equation assignments and for analyzing the changes in the differential index during the model simplification phase of the modelling process. The algorithm provides guidelines on how to modify the model equations in order to obtain an index 1 model after model simplification.

Case studies are used to demonstrate the operation of the algorithm and to follow the effect of constant and steady-state assumptions on the differential index of a simple process model.

Keywords:
DAE-model, differential index, zero degree of freedom, equation-variable graph, model simplification transformation, giving advice

Tartalmi összefoglaló

Jelen dolgozatban a modell-egyszerűsítő feltételezéseknek a rendszer differenciális indexére gyakorolt hatását vizsgáljuk tökéletesen kevert mérlegelési térfogatokból álló folyamatrendszerek esetén. A folyamat-modellek ún. differenciál-algebrai egyenletrendszereknek (DAE) tekinthetők, melyek a megfelelő specifikációs egyenletekkel nulla szabadságfokú modellt alkotnak. Egy ilyen modell az egyenlet-változó gráfjával, a rajta végrehajtandó modell-egyszerűsítő transzformációk hatása pedig egyszerű gráf-transzformációkként írhatók le.

A modellépítés egyszerűsítő fázisában szükséges lehet, hogy a egyenlet-változó hozzárendelésben fellépő változások hatását követni tudjuk a rendszer differenciális indexére nézve. Ezen cél elérésére egy hatékony, polinomidejű inkrementális gráf-elméleti algoritmust ismertetünk. A bevezetésre kerülő algoritmus segítséget nyújt abban, hogy hogyan módosíthatjuk a modellegyenleteket úgy, hogy a modell a fenti hozzárendelést módosító transzformáció után is 1-es indexű maradjon.

Esettanulmányokon keresztül bemutatjuk az algoritmus működését egyrészt egyszerű konstans specifikáció, másrészt állandósult állapot feltételezése esetén, és megvizsgáljuk ezek hatását egy egyszerű folyamat-modellen.

Kulcsszavak:
DAE-modell, differenciális index, nulla szabadságfokú rendszer, egyenlet-változó gráf, model egyszerűsítő transzformáció, tanácsadás

Contents

1 Introduction **7**

2 Computational structure and simplification **9**
2.1 Graph-theoretical elements 9
2.2 Equation-variable graph 9
2.3 Assignments in the equation-variable graph 11
2.4 Model simplification transformations 12
2.5 The effect of model simplification transformations 13
2.6 Closest maximum assignment 13

3 Finding closest maximum assignments **15**
3.1 The algorithm to construct graph B 16
3.1.1 The algorithm 16
3.1.2 Remarks for the algorithm 17
3.1.3 The question of connectivity 17
3.2 The algorithm to find a closest maximum assignment 18
3.2.1 The algorithm 18
3.2.2 Discussion of the algorithm 19
3.2.3 The time complexity of the algorithm 20

4 Giving advice **21**
4.1 Brute-force method 21
4.2 Simple special cases 21
4.2.1 Case 1: a new specification (a) 21
4.2.2 Case 2: a new specification (b) 22

5 Case studies **24**
5.1 The original model structure 24
5.2 Test case 1: a single constant assumption 25
5.3 Test case 2: a single steady-state assumption 25

6 The software **32**
6.1 The manner the software works 32
6.1.1 The structure of in_file 33
6.1.2 The structure of out_file 35

7 Conclusion and discussion **43**
7.1 Evaluation of results and algorithms 43
7.2 Conclusion and future work 43

1 Introduction

Lumped process models are in the form of differential-algebraic equations [3] which are sometimes difficult to solve numerically, due to index and stiffness problems [10]. Therefore the analysis of solvability properties of process models in differential-algebraic equation (DAE) form is of primary importance for dynamic simulation.

The intelligent front-end of dynamic simulators, the so-called computer-aided modelling tools usually provide a visual, interactive means of analyzing the equation-variable patterns in order to choose appropriate variables to satisfy the zero degree of freedom requirement or to investigate equation ordering to enhance solution methods. Structural analysis methods can be found in various forms in *ModKit* [5], *Model.la* [8] and *ICAS* [4], to mention only a few.

The effect of some modelling decisions on the structural solvability has already been investigated [6] and it has been found that a change in the specification may transform an index 1 model to a higher-index one [3]. However, no systematic investigation has been carried out to follow the effect of the modelling decisions on the computational properties of process models over the entire modelling process.

At the same time it is intuitively clear that simplification assumptions, as important modelling decisions applied during the modelling process, may also affect seriously the differential index of lumped process models.

Most of the structural analysis methods use matrix representation of process model structures and linear algebraic tools based thereon. The combinatorial, graph-theoretical approach (see e.g. [7]) is far less popular: its possibilities are less known.

The present thesis is based on former results [9] which can be regarded as a first step towards the systematic analysis of the effect of modelling assumptions on the computational properties of process models using graph theoretical methods.

During the process of constructing model structures it is often necessary to apply simplifications in order to meet a *particular* purpose. In order to obtain models that are easy to solve it is important to keep their differential index equal to 1. The aim of this thesis is two fold.

1. The first aim of this thesis is to propose polynomial-time incremental algorithm for analyzing the changes in the differential index during the model simplification phase of the modelling process, based on graph-theoretical techniques.

2. The second aim is to give advice on how to modify model specifications to keep the index 1 property during model simplification.

In this context a process model is understood as the set of model equations in the form of a DAE system together with a suitable specification forming a model with zero degree of freedom. The effect of model simplification

assumptions is analyzed by describing them as graph transformations acting on the structure graph of the process model.

This thesis is organized as follows. The second chapter includes some definitions and concepts we are using. The third chapter is to describe the algorithm itself. The fourth chapter discusses the question of giving advice. The fifth chapter contains a few cases to demonstrate the algorithm. The sixth one gives a description of the program which has been developed. And in the last chapter we overview the results of this thesis.

2 Computational structure and simplification of lumped process models

This chapter introduces the basic tools and notions on representing process model structures by bipartite graphs and assignments by perfect matchings. This is used for analyzing the effect of model simplification assumptions represented by graph transformations on the computational structure of such models.

2.1 Graph-theoretical elements

Definition 1 *An ordered pair $G = (V, E)$ is called* graph *if V is a nonempty set and $E \subseteq V \times V$ where V is the set of* vertices *and E is the set of* edges.

Notation 2 *The vertex-set of graph G is notated by $V(G)$, the edge-set is by $E(G)$.*

In some cases we should use *directed graph*s instead of undirected ones. The underlying difference between undirected and directed graphs is that an edge of a graph is a set, for eg. $\{ a, b \}$ while an edge of a directed graph is an ordered pair, for eg. (a, b). More precisely:

Definition 3 *An ordered pair $G = (V, E)$ is called* directed graph *if V is a nonempty set and $E \subseteq V \times (V \times V)$ where V is the set of* vertices *and E is the set of* edges.

It could be easily seen that an ordered pair (a, b) is equivalent to $\{ a, \{ a, b \} \}$.

Definition 4 *A G graph is called* bipartite graph *if $V(G)$ can be partitioned into two disjoint sets (A and B) such that one vertex is in A and the other one is in B for each edge.*

Notation 5 *To get easier notations we shall describe the bipartite graph G in the following form:*
$G = (A, B, E)$, where $V(G) = A \cup B$, such that A and B are the two disjoint partition of the vertex set and $E(G) = E$.

Notation 6 *Assume that $X \subseteq V(G)$.*
$N(X) := \{ y \in V(G) \mid \{ x, y \} \in E(G) \}$ for undirected graphs and
$N(X) := \{ y \in V(G) \mid (x, y) \in E(G) \}$ for directed graphs.

2.2 Equation-variable graph

For the purpose of the analysis, the structure of a lumped process model is described by a bipartite graph called *equation-variable graph*.

Definition 7 *A lumped process model is given as a DAE system with the special form*

$$\begin{aligned} (D) \quad & u = \frac{dx}{dt} = f(x, z_s, z) \\ (DS) \quad & x = \int u \, \mathrm{d}t + x_s \\ (A) \quad & 0 = g(x, z_s, z) \\ (AS) \quad & z_s = \boldsymbol{spec} \ (x_s = \boldsymbol{spec}) \end{aligned} \tag{2.1}$$

where x is the vector of differential variables, u contains their derivatives and x_s their initial values, z is the vector of algebraic variables, z_s is that of the specified algebraic variables, and **spec** *stands for a general given constant (may be different for each variable).*

It is important to emphasize that the process model is understood as a set of model equations (equations $(D), (DS), (A)$ together with a suitable specification (equations (AS)) as an integral part of the model.

The equations in the above general model structure are labelled before the equation with a label in parenthesis, like (D) or (A). Their label indicates a classification of the equations into the following categories:

- (D): differential equations (conservation balance equations)
- (DS): integrated differential equations
- (A): algebraic (constitutive) equations
- (AS): specification equations

Definition 8 *The equation-variable graph of the above model is a bipartite graph $B_0 = (X, Y, E)$, where one vertex class, Y represents the set of equations,*

$$Y = \{ (D), (DS), (A), (AS) \} \tag{2.2}$$

and the other one, X contains the variables

$$X = \{ x, u, z, z_s, x_s \} \ . \tag{2.3}$$

A variable-vertex is adjacent to an equation-vertex if and only if the variable in question appears in the corresponding equation:

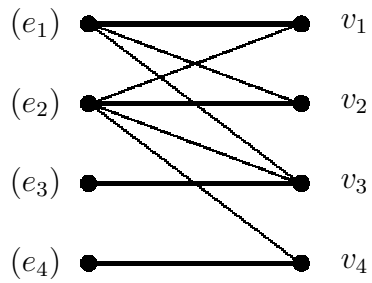
$$E = \{ \{ y, x \} \mid y \in Y, x \in X \text{ and variable } x \text{ takes place equation } y \} \tag{2.4}$$

It is important to distinguish two different types of edges: F and its complement set $E \setminus F$. F consists of the edges between an equation and its *specified variable*. They are the initial assignments in the equation-variable graph. We shall refer them to *thick* edges. The edges in $E \setminus F$ shall be referred to *thin* edges. This property is the colour of the edges. It is easy to see that $|F| = |X| = |Y|$.

Example 9 *Let us consider the following simple example:*

$$\begin{aligned}(e_1) \quad v_1 &= f(v_2, v_3)\\ (e_2) \quad v_2 &= g(v_1, v_3, v_4)\\ (e_3) \quad v_3 &= \textbf{\textit{const.}}\\ (e_4) \quad v_4 &= \textbf{\textit{const.}}\end{aligned}$$

Having built the equation-variable graph we should get the following graph (B_0):



2.3 Assignments in the equation-variable graph

Definition 10 *Assume that $M \subseteq E(G)$. The set M is called* matching *or* partial matching *if none of these edges has any common vertex. In this case they are said to be* independent edges. *The partial matching* covers *its edges' vertices. A matching is called* perfect matching *if it covers each vertex of G.*

Note that the perfect matching is not necessarily unique.

Theorem 11 (Frobenius) *In a $G = (A, B, E)$ bipartite graph there exists a perfect matching if and only if $|A| = |B|$ and $\forall X \subseteq A\colon\ |N(X)| \geq |X|$.*

Notation 12 *The number of components in G containing odd number of vertices is denoted by $c_p(G)$.*

Theorem 13 (Tutte) *In a $G = (A, B, E)$ bipartite graph there exists a perfect matching if and only if $\forall X \subseteq V(G)\colon c_p(G \setminus X) \geq |X|$.*

Theorem 14 ([3]) *Assume that the zero degree of freedom requirement, i.e. $DOF = 0$ is satisfied. Then the DAE model described above is of index 1 if and only if there exists a perfect matching in the equation-variable graph.*

The general model structure (**Def. 7**) implies a possible *natural assignment between the categories of the variables and equations* in the following way.

1. differential equations (D) — derivatives of differential variables u
2. integrated differential equations (DS) — differential variables x
3. algebraic equations (A) — algebraic variables z

4. specification equations (AS) — specification variables $\{ z_s, x_s \}$

Note that the variables in the specification $S = \{ z_s, x_s \}$ appear in the structure in a special way. *These variables can* only *be assigned to their corresponding specification equations from the equation subset* (AS) because a specification equation has no other variable than the one it specifies.

It is important to note that *an assignment between equations and variables of a process model with* $DOF = 0$ *is represented unique by a perfect matching in the equation-variable graph.*

2.4 Model simplification transformations

Modelling assumptions form the fundamental basis for the mathematical description of a process system. These assumptions can be translated into either additional mathematical relations or constraints between model variables, equations, balance volumes or parameters.

A formal representation of modelling assumptions as transformations acting on the set of model equations is proposed in [2] in order to analyze the effect of modelling assumptions on the properties of process models. This general approach is specialized here to have graph transformations acting on the equation-variable graph, to describe the effect of model simplification transformations on the differential index of lumped process models.

Based on their effect on process models, model transformations can be categorized as

- *algebraic transformations*, which produce an algebraically equivalent model with the same applicability domain,
- *model simplification transformations* being projections in the algebraic sense, which produce a model with narrower applicability domain,
- *model enrichment* or model building *transformations.*

It is easy to see that specification of one or more variables belongs to the second type of transformations. In this paper we will deal with the second group, i.e. with model simplification transformations. They can formally be described by a triplet of

$$\textit{ModelVariable} \quad \textbf{relation} \quad \textit{ConstantOrModelVariable} \tag{2.5}$$

where the **relation** sign is usually an equality sign ("="). When applying a model simplification transformation to a set of process models, one simply adds the transformation equation **Eq. (2.5)** to the already existing set of model equations.

In the case of process models with $DOF = 0$*, however, we need to change the set of already existing model equations (delete one equation in the general case) in order to preserve this property.* This induced change depends on the structure of the model equation set and on the applied model simplification transformation.

From the viewpoint of their effect on the assignment structure of a process model, we shall distinguish and use later on the following types of model simplification transformations.

1. *constant assumption*, when a transformation equation $ModelVariableZ = GivenConstant$ is applied where $ModelVariableZ$ refers to an algebraic variable z,

2. *steady-state assumption*, when we apply $ModelVariableU = 0$ with $ModelVariableU$ referring to the derivative of a differential variable.

2.5 The effect of model simplification transformations

Similarly to the general case, model simplification transformations are described as graph transformations on the equation-variable graph. Assume that a full equation-variable assignment (a perfect matching) is given together with an equation-variable graph. Then a simplification transformation may be:

1. *edge-changing* transformation when only some non-matching edges are removed or inserted,

2. *assignment-changing* transformation when some of the matching edges are deleted,

3. *vertex-changing* transformation when new equations and/or variables appear causing edge changes and change in the specification, too.

Note that both the constant and steady-state assumption types defined above belong to the group 2 or 3, i.e. are of the 2nd or 3rd type.

2.6 Closest maximum assignment

Let an equation-variable graph with a full equation-variable assignment be given. Moreover, let us consider a model simplification transformation of the 2nd or 3rd type affecting the equation-variable assignment.

Definition 15 *A* closest maximum assignment *is a (not necessarily full) assignment in the transformed equation-variable graph, which has the largest possible number of edges and under this requirement it has the largest number of matching edges in common with the original full assignment.*

Note that the closest maximum assignment is not necessarily unique.

Example 16 *As a small example, consider*

$$X = \{ x_1, x_2, x_3 \} \ , \ Y = \{ y_1, y_2, y_3 \} \ , \ E = \{ \{ x_i, y_j \} \mid 1 \leq i, j \leq 3 \} \ ,$$

and let the original full assignment be

$$F = \{ \{ x_1, y_1 \}, \{ x_2, y_2 \}, \{ x_3, y_3 \} \} \ .$$

If the transformation deletes $\{ x_1, y_1 \}$*, then both*

$$\{ \{ x_1, y_2 \}, \{ x_2, y_1 \}, \{ x_3, y_3 \} \} \quad \textit{and} \quad \{ \{ x_1, y_3 \}, \{ x_3, y_1 \}, \{ x_2, y_2 \} \}$$

are closest maximum assignments (but for e.g. $\{ \{ x_1, y_2 \}, \{ x_2, y_3 \}, \{ x_3, y_1 \} \}$ *is not).*

Our main purpose is to decide whether the model remains of index 1 after applying some transformation or not. That is why we should first find a closest maximum assignment and having done this we should examine whether the closest maximum assignment we found is a perfect matching or not (according to **Thm. 14**). It is easily decidable because—being bipartite graph—if the number of edges in the maximum closest assignment is equal to the cardinality of X or Y in B ($|X| = |Y|$), then the assignment we found is a perfect matching and the transformed model is of index 1.

3 Algorithm for finding closest maximum assignments

In this chapter we shall introduce an algorithm and prove that it can efficiently find closest maximum assignments satisfying the first part of our requirements.

Definition 17 *Given the equation-variable graph of the original model (B_0) and the set of desired transformations (i.e. at least one equation to add and the same number of equation(s) to delete). The graph with the maximum number of common edges and vertices with the original graph B_0 and satisfying the properties:*

- *the equation-vertices of the equations to delete are removed with all of their adjacent edges*
- *the new equation-vertices are inserted to the graph for the equations to add with their corresponding edges*

is called transformed equation-variable graph.

In this chapter an algorithm to construct the transformed graph B from the original equation-variable graph B_0 is described. At first, the following operations are introduced for using in the algorithm description:

AddNode $[n]$: It creates a new node n in B_0.
DelNode $[n]$: It removes the node n from B_0 with all adjacent edges.
AddEdge $[n_1, n_2, c]$: It puts an edge in B_0 between the two given nodes n_1 and n_2 with colour c.
DelEdge $[n_1, n_2]$: It removes the edge from B_0 between the two given nodes n_1 and n_2.
GetEqu $[n]$: It returns the label of equation defining the variable n.
GetVar $[n]$: It returns the label of variable defined by the equation n.

If a node n is associated to an equation in B_0, then we shall put it into parentheses to make it easier to follow the algorithms.

The algorithm can be divided into two main steps. The first one is to construct the transformed equation-variable graph is described in **Section 3.1**. The second one contains the finding method itself as it is described in **Section 3.2**.

The algorithm is based on sequential approach. That means that a sequence of simplification transformations can only be processed in steps. Each step involves a basic simplification transformation, i.e. it has only one equation to add and an other one to delete. Thus, if one would like to apply more than one transformation, then the transformations should be linked one after the other.

3.1 The algorithm to construct graph B

Let a desired model simplification transformation be given. It consists of two elements: one equation to add and another one to eliminate described in the following form:

$$\begin{array}{ll} \textbf{add:} & (l_1) \quad v_1 = f(v_2, v_3, \ldots, v_n) \\ \textbf{del:} & (l_2) \end{array} \tag{3.1}$$

It is obvious that $(l_1) \neq (l_2)$ because if they were equal there would not be any transformation at all.

3.1.1 The algorithm

The algorithm for executing a model simplification transformation in **Eq. (3.1)** is described as a sequence of steps as follows:

1. AddNode $[(l_1)]$

2. $(l_0) :=$ GetEqu $[v_1]$

3. DelEdge $[(l_0), v_1]$

4. Examine whether (l_0) has become an isolated node or not.

yes: If $(l_2) \neq (l_0)$, then the given transformation cannot be applied.

5. Examine the relation of (l_2) to (l_0).

- If $(l_2) \neq (l_0)$, then $(t) := (l_0)$
- If $(l_2) = (l_0)$, then $(t) := (l_1)$

6. $s :=$ GetVar $[(l_2)]$

7. DelNode $[(l_2)]$

8. AddEdge $[(l_1), v_i, \text{thin}]$, $2 \leq i \leq n$

9. Determine the colour of edge between (l_1) and v_1

- If $v_1 = s$, then AddEdge $[(l_1), v_1, \text{thin}]$
- If $v_1 \neq s$, then AddEdge $[(l_1), v_1, \text{thick}]$

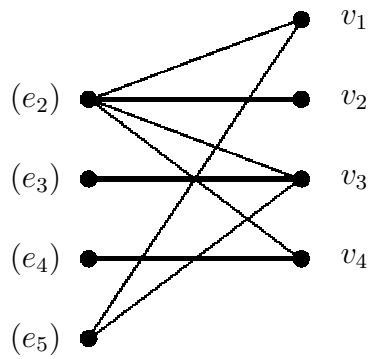
Having done these operations the original equation-variable graph denoted by B_0 becomes a transformed one B. It could be easily seen that the graph B satisfies the conditions described in **Def. 17**. It is also true that the transformed equation-variable graph B is unique under this construction.

Note that this algorithm does not construct the new perfect matching in the transformed equation-variable graph. It only constructs the transformed equation-variable graph itself.

Example 18 *Let us take* ***Example 9*** *and consider the following transformation:*

$$\begin{array}{ll} \textbf{\textit{add:}} & (e_5) \quad v_1 = f(v_3) \\ \textbf{\textit{del:}} & (e_1) \end{array}$$

Constructing the transformed equation-variable graph (B) one should get the following graph:



3.1.2 Remarks for the algorithm

It is important to make some observations to the following steps of the algorithm.

4. If there were *any* isolated node in the graph, it would mean that there is either an equation defining nothing or a variable defined by nothing.

5. If in the case of $(l_2) \neq (l_0)$ (t) were equal to (l_1), then (l_0) would remain without defined variable because there would not be any edge from (l_0) belonging to F. Moreover,that would mean that v_1 should be s.

7. If after erasing (l_2) there was any isolated node, then (l_2) should not be deleted. In this case the desired transformation cannot be applied.

3.1.3 The question of connectivity

The question of isolated nodes have already been discussed. Now we have to think over what happens if B_0 is not connected. By deleting some edges from the graph B_0 during **Algorithm 3.1** the graph may fall into set of non-connected sub-graphs.

Let $G_1, G_2, \ldots G_n$ are the connected subgraphs of B. Then

$$E(B) = \bigcup_{i=1}^{n} E(G_i), \quad \bigcap_{i=1}^{n} E(G_i) = \emptyset$$

and

$$V(B) = \bigcup_{i=1}^{n} V(G_i), \quad \bigcap_{i=1}^{n} V(G_i) = \emptyset.$$

Thus, $E(B)$ can be partitioned in the following way:

$$P_1 = \{\, E\,(G_i) \mid E\,(G_i) \subseteq F,\, i = 1, 2, \ldots n \,\}$$

and

$$P_2 = E(B) \setminus P_1.$$

That is, P_1 consists of the edge sets of subgraphs G_i containing only thick edges. It could be easily seen that these G_is belonging to P_1 contain only two vertices and one thick edge between them so they are represented by exactly one equation in the form of

$$(l) \quad \text{var} = \textbf{const}$$

Moreover, they are independent from the other parts of the system. If $P_1 = \emptyset$ and $n > 1$, then the model has been collapsed into n pieces of independent equation-system. If $n > 1$ and $P_2 \neq \emptyset$, we also have n pieces of independent equation-system but in this case we can simply omit P_1 from the graph.

3.2 The algorithm to find a closest maximum assignment

Let us briefly write again our definitions from the previous chapter. Let $B = (X, Y, E)$ be the transformed equation-variable graph of the process model in question, with two vertex classes X (variables), Y (equations) and edge set E (dependencies between the equations and the variables). Let $F \subseteq E$ be the assignment given (i.e. a matching selected in B, covering all but one vertex in each of X and Y). The elements of F are the edges denoted by $e = \{ x, y \}$ where $x \in X$ and $y \in Y$.

To get easier notations we should think of s and t as edges where s and t are the two vertices not incident with any matching edges as we generated in **Algorithm 3.1**. (Note that $s \in X$ and $t \in Y$.) Think of s as an edge in the following way: $\{ \_, s \}$ where $\_$ is a pseudo-vertex of graph B. Similarly think of t as $\{ t, \_ \}$. Under this agreement we can think $F \cup \{ s, t \}$ as edges. Being edges of an undirected graph the order of their elements is irrelevant.

3.2.1 The algorithm

Now we construct an auxiliary directed graph $D = (V_D, E_D)$ where $V(D) = V_D$ and $E(D) = E_D$ as follows.

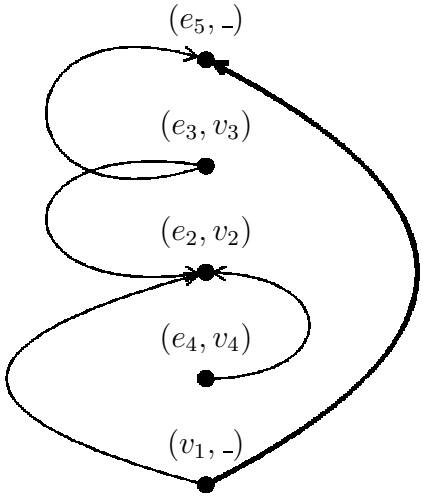
Let $V_D := F \cup \{ s, t \}$. An ordered pair (s, e) is an edge in D if and only if $\{ s, y \} \in E$, where $e \in F \cup \{ t \}$, $e = \{ x, y \}$, $x \in X$, $y \in Y$. Similarly, a pair (e, t) with $e = \{ x, y \} \in F$ is an edge in D if and only if $\{ x, t \} \in E$. Inside F, there is an edge from $f_i = \{ x_i, y_i \}$ to $f_j = \{ x_j, y_j \}$ if $\{ x_i, y_j \} \in E \setminus F$, where $\{ f_i, f_j \} \subseteq F$.

Now, we run the *Breadth-First Search* algorithm (shortly BFS algorithm) (see e.g. [1], Section 23.2) on D, starting from s. As it is well-known, BFS can decide whether t is reachable from s along a directed path or not, and if it is, then BFS also finds *a shortest $s \to t$ path*. Note that the directed path we found using BFS is not necessarily unique. Then we obtain a required solution based on the following two cases.

1. If there is no directed path from $s \to t$, then the given assignment F is of maximum size in B.

2. If t is reachable from s and BFS finds a shortest $s \to t$ path $(s, f_1, f_2, \ldots, f_k, t)$, then an assignment closest to F and of size $|F|+1$ is $(F \setminus \{ f_1, f_2, \ldots, f_k \}) \cup \{ (s, y_1), (x_1, y_2), \ldots, (x_{k-1}, y_k), (x_k, t) \}$, where $f_i = \{ x_i, y_i \}, i = 1, \ldots, k$.

Example 19 *Let us take **Example 18**. Having run the algorithm above we shall get the graph with a directed path in bold below (graph D):*



Thus, the closest maximum assignment we found is the following set of edges: $\{ \{ e_2, v_2 \}, \{ e_3, v_3 \}, \{ e_4, v_4 \}, \{ e_5, v_1 \} \}$

3.2.2 Discussion of the algorithm

To see that these assertions are valid indeed, observe first that the second case already describes how a larger assignment can be derived from any $s \to t$ path. Conversely, given any assignment F' of size $|F'| > |F|$, the edges of F' must cover the entire $X \cup Y$, including s and t as well. Hence, there is an edge $\{ s, y_1 \} \in F'$. If $y_1 \neq t$, then $f_1 = \{ x_1, y_1 \} \in F$ holds for some $x_1 \in X$. This x_1 is again covered with some $\{ x_1, y_2 \} \in F'$. If $y_2 = t$, then (s, f_1, t) is an $s \to t$ path in D, while if $y_2 \neq t$, then we again find some $f_2 = \{ x_2, y_2 \} \in F$ and $\{ x_2, y_3 \} \in F'$, and so on, until we eventually reach t and hence obtain a path from s to t. Thus, the $s \to t$ paths $P(s,t)$ are in one-to-one correspondence with the full assignments F'. It remains just to observe that $|F \setminus F'|$ equals the number of vertices in $P(s,t)$ minus 2. Consequently, minimizing the length of $P(s,t)$ yields the largest possible number of edges preserved in $F \cap F'$, as required.

Assuming that the (unique) edge $f \in F$ incident with any given vertex $x \in X \setminus \{ s \}$ or $y \in Y \setminus \{ t \}$ can be identified in constant time, the construction of D requires no more than $O(|E|)$ steps by scanning $E \setminus F$. Since BFS can be implemented in linear time, the closest assignment is found in linear time, too.

3.2.3 The time complexity of the algorithm

The following results show the efficiency of our algorithm above. The first one is formally slightly more general than the case covered in the algorithm. A simple transformation can be applied, however. Namely we can contract the subset of X uncovered in F to a single vertex s, and the uncovered part of Y to another single vertex t. Then we obtain

Theorem 20 ([9]) *There is a linear-time algorithm to decide whether an assignment is of maximum size. In particular, if just one edge is deleted from a perfect matching, it can be decided in linear time whether the modified system still has differential index 1. Moreover, if the assignment is below the maximum size by 1, then a closest assignment can be found in linear time.*

Theorem 21 ([9]) *A closest assignment of maximum size can be found in polynomial time.*

4 Giving advice

It would be useful if there were a method to give some advice on how to choose an equation to delete in transformation **Eq. (3.1)** as it is described in **Chapter 3**. In some cases one has only an equation to add to the model but has no idea what to remove. In this case one would expect with reason some advice on the removal such that it keeps the model of index 1.

4.1 Brute-force method

The most obvious way would be to try to delete each equation and examine whether the model remains of index 1 or not. This method would contain generating a temporary transformed graph and applying the discussed algorithm (**Chapter 3**) for each possible equation. In spite of the fact that both steps can be done in polynomial time it would be beneficial to exclude some equations from the domain of this so called brute-force approach.

Example 22 *Consider the simple model described in* ***Example 9****. Let us add a new equation to the model:*

$$\textbf{\textit{add:}} \quad (e_5) \quad v_1 = h(v_3)$$

Now examine which equations can be removed. Having run the brute-force adviser we would find that there is only one equation which cannot be removed, namely (e_2)*. If we removed the equation* (e_2)*, then the transformed equation-variable graph would not be connected.*

4.2 Simple special cases

We use the notations as they are written in **Algorithm 3.1**.

4.2.1 Case 1: a new specification (a)

The most simple simplification case is to make a variable equal to a constant and remove the equation which determined this variable earlier. Let the original equation be the following:

$(e_p) \quad v_p = f(v_{p_1}, v_{p_2}, \ldots, v_{p_k})$

And let us consider the following transformation:

$$\begin{array}{ll} \textbf{add:} & (e_p{}') \quad v_p = \textbf{const} \\ \textbf{del:} & (e_p) \end{array}$$

Now apply the **Algorithm 3.1**.

$(l_1) = (e_p{}')$
$v_1 = v_p$
$(l_2) = (e_p)$

1. AddNode $[(e_p{}')]$

2. $(l_0) = \text{GetEqu}\,[v_p] = (e_p)$

3. $\text{DelEdge}\,[(e_p), v_p]$

4. It is not interesting whether (e_p) is isolated or not
because $(l_2) = (l_0)$ $(= (e_p))$.

5. $(l_2) = (l_0) \Rightarrow (t) = (e_p{}')$

6. $s = \text{GetVar}\,[(e_p)] = v_p$

7. $\text{DelNode}\,[(e_p)]$

8. There is no edge to add.

9. $v_p = s \Rightarrow \text{AddEdge}\,[(e_p{}'), v_p, \text{thin}]$

Now we have the transformed equation-variable graph B. It is easy to see that constructing the graph D we would have a 1 long path from $s \rightarrow t$. That means if the original model was of index 1, then the transformed model would keep this property. Thus, the advice for this case can be to remove the equation which determined the specified variable earlier.

Of course this procedure does not guarantee that the transformed model will be connected. That is why the connectivity-checking has to be done.

4.2.2 Case 2: a new specification (b)

Generalizing the results of the Case 1. now let us change an equation to an other equation. Let the original equation again be the following:

$(l_1) \quad v_p = f(v_{p_1}, v_{p_2}, \ldots, v_{p_k})$

Now consider the following transformation:

$$\begin{array}{ll} \textbf{add:} & (e_p{}') \quad v_p = f(v_1, v_2, \ldots, v_n) \\ \textbf{del:} & (e_p) \end{array}$$

Let us apply the **Algorithm 3.1** again. We can realize that the steps of the algorithm differ from the previous case in the Step 8 only.

$(l_1) = (e_p{}')$
$v_1 = v_p$
$(l_2) = (e_p)$

1. $\text{AddNode}\,[(e_p{}')]$

2. $(l_0) = \text{GetEqu}\,[v_p] = (e_p)$

3. $\text{DelEdge}\,[(e_p), v_p]$

4. It is not interesting whether (e_p) is isolated or not
because $(l_2) = (l_0)$ $(= (e_p))$.

5. $(l_2) = (l_0) \Rightarrow (t) = (e_p{}')$

6. $s = \mathrm{GetVar}\,[(e_p)] = v_p$

7. $\mathrm{DelNode}\,[(e_p)]$

8. $\mathrm{AddEdge}\,[(e_p), v_i, \mathrm{thin}]\,,\ i = 1, \ldots, n$

9. $v_p = s \Rightarrow \mathrm{AddEdge}\,[(e_p{}'), v_p, \mathrm{thin}]$

Now we have the transformed equation-variable graph B again. It is obvious from the previous case that the graph D contains a 1 long path from $s \rightarrow t$. That means if the original model was of index 1, then the transformed model keeps this property under this transformation again. Now, the advice can be to remove the equation which was modified.

Of course neither this procedure can guarantee that the transformed model will be connected. Thus, the connectivity of the transformed equation-variable graph should be checked again.

5 Case studies

The algorithm for finding a closest assignment is illustrated on the following examples by examining the effect of both constant and steady-state assumptions. The model used is a simplified version of the one described in [2].

5.1 The original model structure

In order to demonstrate the operation of our proposed algorithm and the findings obtained from the results, we consider a simple process system shown in **Fig. 1**. It is a simple tank with inflow (F) and outflow (L) containing liquid with overall mass M_L and internal energy U_L. The tank is equipped with a heater with heat flux Q. The liquid evaporates to the environment having temperature T^0 and pressure P^0. A single component liquid with constant physico-chemical properties is assumed.

The original model equations are the following (the labels of the equations are between parentheses before the equations):

$$
\begin{aligned}
(d1) \quad & m_L = \frac{dM_L}{dt} = F - E - L \\
(d2) \quad & u_L = \frac{dU_L}{dt} = Fh_F - Eh_{LV} - Lh_L + Q + Q_E \\
(ds1) \quad & M_L = \int m_L dt + M_{L0} \\
(ds2) \quad & U_L = \int u_L dt + U_{L0} \\
(a1) \quad & E = k_{LV}(P^* - P^0) \\
(a2) \quad & Q_E = u_{LV}(T_L - T^0) \\
(a3) \quad & T_L = \frac{U_L}{M_L c_{pL}} \\
(a4) \quad & P^* = H(T_L)
\end{aligned}
$$

where $H(\cdot)$ is a given invertible function.

Let us choose the specified variables as follows:

$$
\begin{aligned}
(as1) \quad & Q = \textbf{const} \\
(as2) \quad & F = \textbf{const} \\
(as3) \quad & L = \textbf{const} \\
(as4) \quad & M_{L0} = \textbf{const} \\
(as5) \quad & U_{L0} = \textbf{const}
\end{aligned}
$$

The constant physico-chemical parameters are: h_F, h_{LV}, h_L (mass-specific specific enthalpies), c_{pL} (specific heat), k_{LV}, u_{LV} (heat and mass transfer coefficients), the initial conditions for the mass and internal energy are M_{L0} and U_{L0}; and **const** denotes a given specified value for the variable in question (may be different for each variable).

Fig. 2 shows the equation-variable graph of the above model together with the "natural" perfect matching. It is simple to see that this model has differential index 1.

5.2 Test case 1: a single constant assumption

We examine the effect of a constant assumption on the liquid temperature T_L. For this purpose we introduce the following new transformation equation:

$$Add(c^*) \quad T_L = \mathbf{const}$$

Since we have a new equation for determining T_L we need to delete one equation from the model to meet the zero degree of freedom requirement. Note that the new equation above will be assigned to T_L because it has no other variable present in it (this is a new specification equation). The obvious choice for the equation to be deleted is the "old" determining algebraic equation $(a3)$ which is described by the transformation equation:

$$Del(a3) \quad T_L = \frac{U_L}{M_L c_{pL}}$$

It is easy to see that the assignment will not change in this case except for the newly introduced edge $T_L - (c^*)$ which replaces the old edge $T_L - (a3)$. In order to get the transformed equation-variable graph we have to apply the **Algorithm 3.1**. Then we should get the graph as it is shown in **Fig. 3**.

The operation of **Algorithm 3.2** for this case is shown in **Fig. 4** where the vertices s and t are (c^*) and T_L, respectively, and the shortest path between them is their connecting edge.

It is important to note that we have a number of other choices for the equation to be deleted besides the above obvious choice. The most common choice would be to delete one of the specification equations $(as1) - (as4)$ in the model.

On the other hand, the choice to delete equation $(a3)$ leads to a model where the equation $U_L = c_{pL} M_L T_L$ no longer holds and thus, an important relationship is missing from the model.

5.3 Test case 2: a single steady-state assumption

Next we examine the effect of a steady-state assumption on the liquid mass M_L. For this, let us introduce the following new equation in the original model:

$$Add(a^*) \quad m_L = \frac{dM_L}{dt} = 0$$

The above equation will be necessarily assigned to m_L because no other variable is present in it.

Since we have a new equation, we need to delete an equation from the resulted set of equations. Note that, unlike the previous case, we do not have an obvious candidate equation to be deleted. Thus, we have to choose one

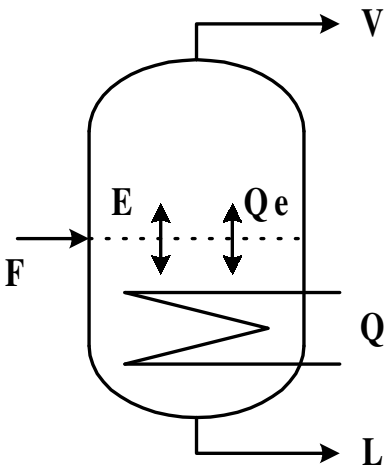


Figure 1: A simple process system

to reduce the number of specified variables, i.e. the number of specification equations. This is done by changing the specifications $(as1) - (as4)$ in such a way that we delete equation $(as4)$ from the new model.

$$Del(as4) \qquad M_{L0} = \mathbf{spec}$$

Having applied the **Algorithm 3.1** one gets the transformed equation-variable graph as it is seen in **Fig. 5**. Using **Algorithm 3.2** we find that the modified system still has differential index 1, and we can also determine the assignment which is closest to the original one. The original and the modified closest assignments are shown in **Fig. 6**.

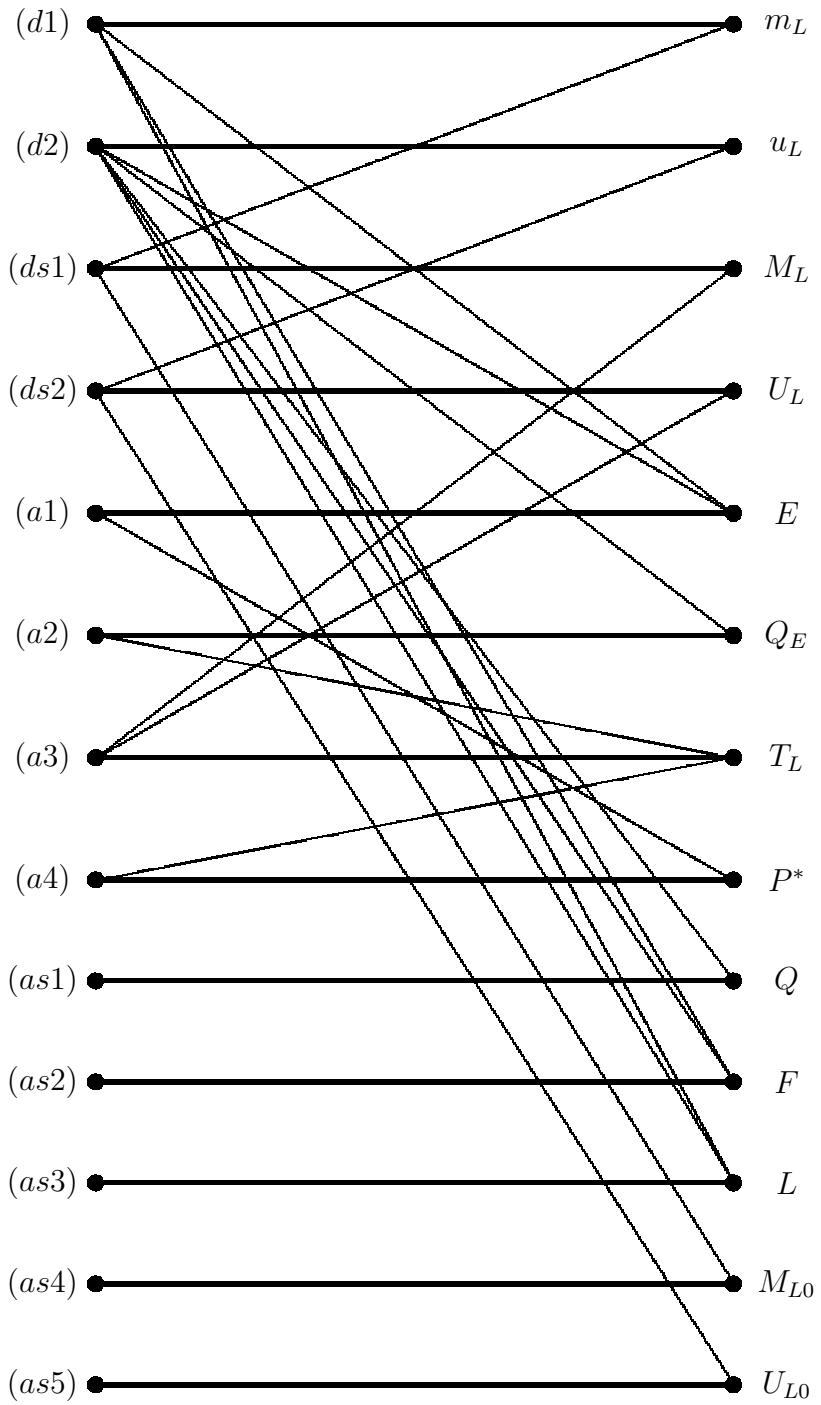


Figure 2: The equation-variable graph of the original model (graph B_0)

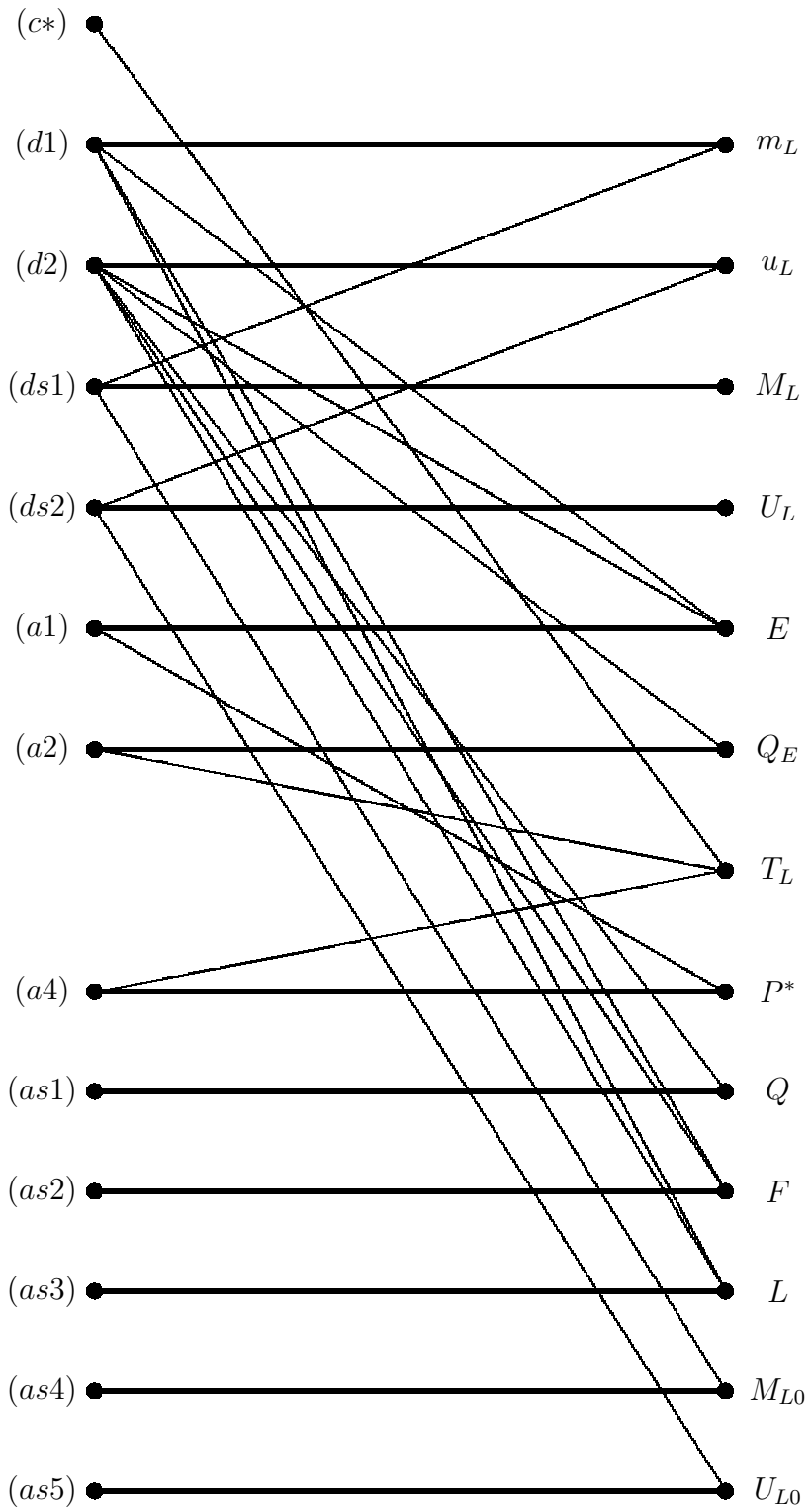


Figure 3: The operation of **Algorithm 3.1** for Case 1. (graph B)

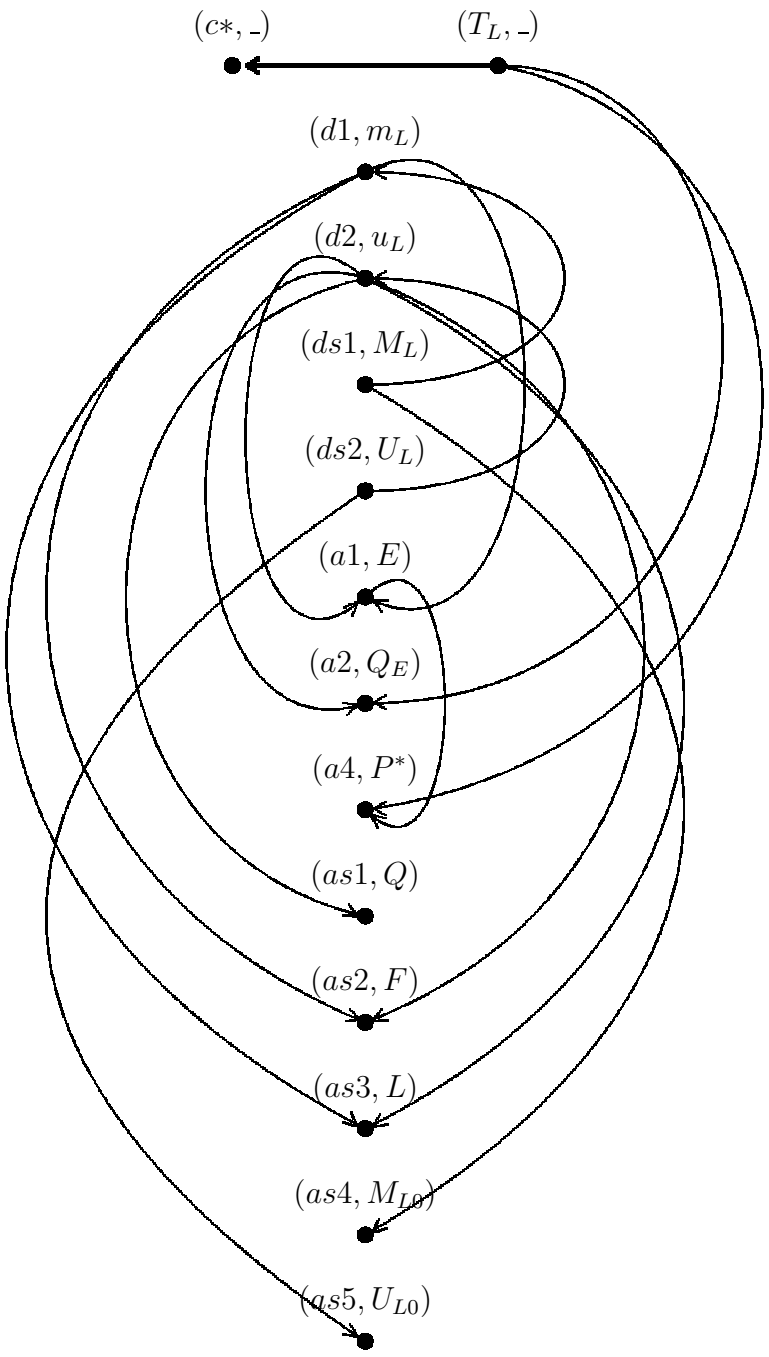


Figure 4: The operation of **Algorithm 3.2** for Case 1. (graph D)

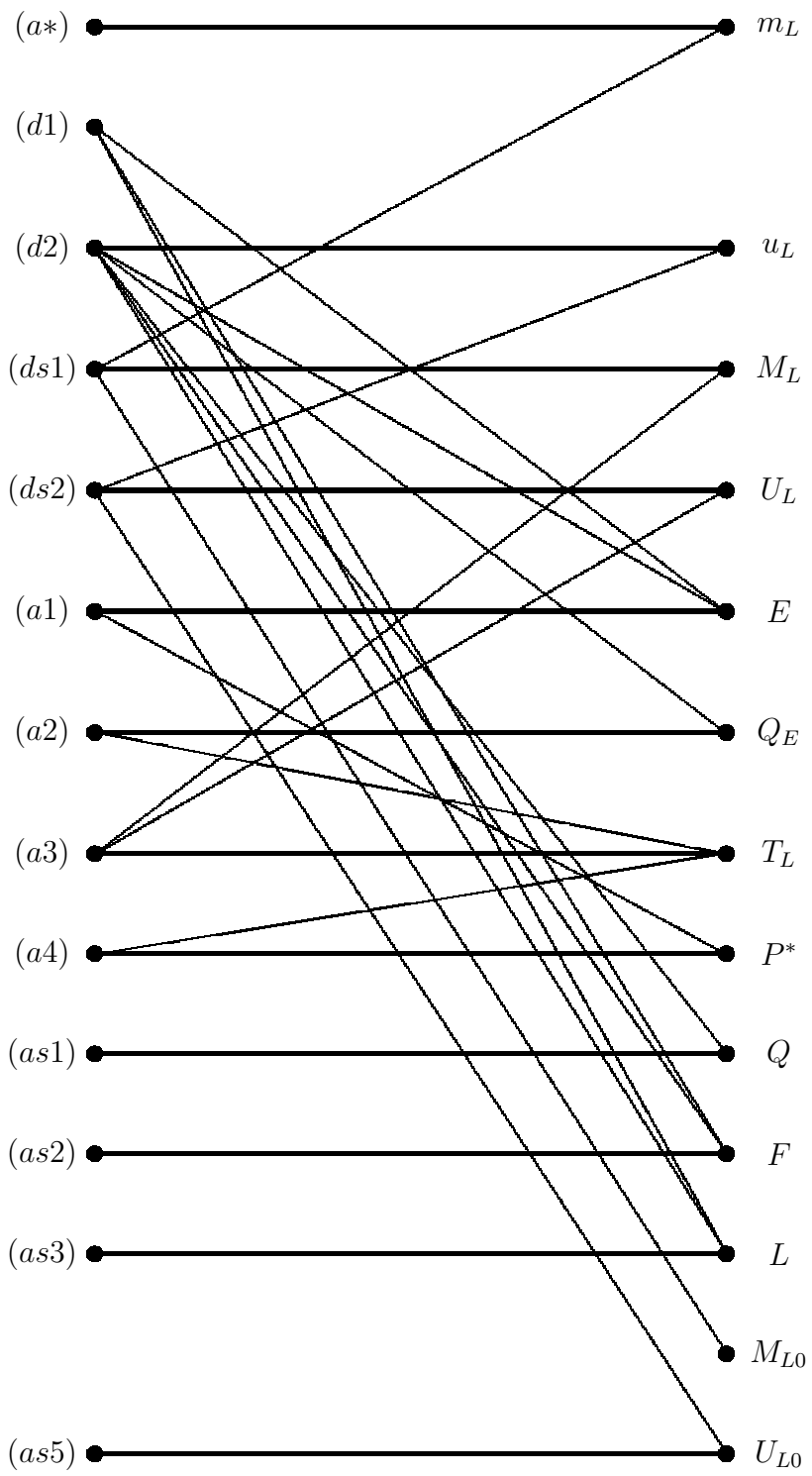


Figure 5: The operation of **Algorithm 3.1** for Case 2. (graph B)

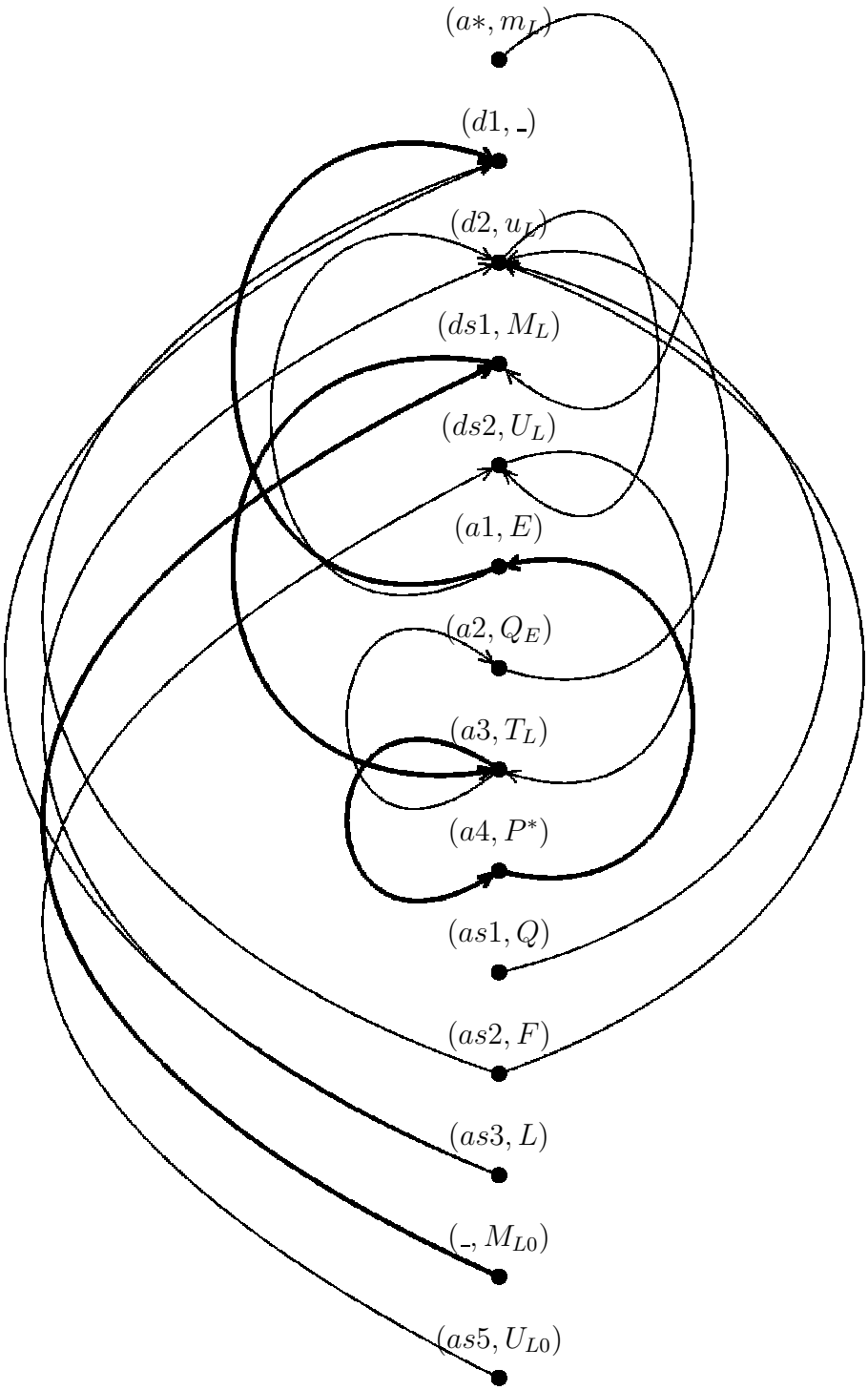


Figure 6: The operation of **Algorithm 3.2** for Case 2. (graph D)

6 The software

A software has been developed to demonstrate the discussed algorithm using C++ language. This language has been chosen because it is one of the most flexible programming languages and it can be considered platform-independent. To make the implementation easier an appropriate template-library has been searched. The whole documentation and the source code of this well polished and documented library are found at WWW.BOOST.ORG on the Internet.

6.1 The manner the software works

The use of this program is quite simple. One should describe the set of model-equations in a special form and should give the desired transformation. Describing the transformation can be made in two different ways.

The first way is used for examining the differential index of the model having applied the given transformation. After running the program this way it gives answers for the following problems:

- Is the transformed model of index 1?
- Is the transformed model connected?
- Finds a maximum closest assignment in the transformed model (naturally if it exists) and writes the equations describing the assignment it found.

The second way is for determining the set of erasable equations. One should describe the set of model equations also using this way and give the equation to add. Then the program determines the set of erasable equations.

The parametrization of the program goes as follows:

```
graph.exe <in_file> <out_file> [-v]
```

where

in_file contains the description of the model-equations and the desired transformation.

out_file contains the maximum closest assignment it found.

-v is an optional parameter which orders the program to give verbose output during its running. The output appears on the standard output making it easy to redirect it to anywhere you want to (for eg. printer or file).

6.1.1 The structure of in_file

The `in_file` consists of two main parts. The first part (`equations{...}`) is to describe the equations of the model. The second part (`transforms{...}`) is to describe the desired transformations. The format of the `in_file` is as follows:

```
equations
{
  <equation>;
  <equation>;
  ...
}

transforms
{
  <add>;
  <del>; | advice;
}
```

The term `<equation>` in the section `equations{...}` describes one equation of the model. Its form is:

`equ(equation_label,defined_variable,list_of_used_variables)`

where `list_of_used_variables` can be either

- `{v_1,v_2,...,v_n)` where `v_i`, `i` $= 1, 2, \ldots, n$ are the variables appearing in the right side of the equation or
- `_` ("hard space") should be used when the given equation defines a constant value (i.e. it has no free variables at all).

The term `<add>` in the section `transforms{...}` specifies the new equation to add to the model. Its form is is the same as `<equation>`'s is. The term `<del>` specifies the equation to delete from the model. Its syntax is:

`del(equation_label)`

If you have no idea what equation you could delete, then you may use the adviser function of this software. You can invoke this feature by giving the command `advice` instead of the term `<del>` in the section `transforms{...}`.

Using the adviser no `out_file` will be generated. In spite of the fact that in the case of using the adviser the second argument of the program (`out_file`) will not have any effect, it has to be specified. It is suggested that you should write `advice` in the place of `out_file`.

Example 23 *The description file of Case 2 in Chapter 5 can be found in this example.*

```
equations {
  equ(d1,mL,{E,F,L});
  equ(d2,uL,{F,E,L,Q,QE});
  equ(ds1,ML,{mL,ML0});
  equ(ds2,UL,{uL,UL0});
  equ(a1,E,{P*});
  equ(a2,QE,{TL});
  equ(a3,TL,{UL,ML});
  equ(a4,P*,{TL});

  equ(as1,Q,_);
  equ(as2,F,_);
  equ(as3,L,_);
  equ(as4,ML0,_);
  equ(as5,UL0,_);
}

transforms {
  add(a*,mL,_);
  del(as4);
}
```

Example 24 *The description file of* ***Example 22*** *can be seen in this example. It is to demonstrate how the adviser feature works.*

```
equations {
  equ(e1,v1,{v2,v3});
  equ(e2,v2,{v1,v3,v4});
  equ(e3,v3,_);
  equ(e4,v4,_);
}

transforms {
  add(e5,v2,{v4});
  advice;
}
```

The result is put to the standard output, no `out_file` *will be generated.*

```
You may delete equation: e1
You may delete equation: e3
You may delete equation: e4
```

We can see that the equation (e_2) *is not among them as it was discussed in* ***Example 22****.*

6.1.2 The structure of out_file

Having run the program it finds a maximum closest assignment and writes the assignment it found to the `out_file` in the same form of the `equations{...}` section in `in_file`. Using the same description format makes it possible to chain the transformations in succession.

Example 25 *The generated* `out_file` *of Case 2 in Chapter 5 is found in this example.*

```
equations
{
  equ(d2,uL,{F,E,L,Q,QE});
  equ(ds2,UL,{uL,UL0});
  equ(a2,QE,{TL});
  equ(as1,Q,_);
  equ(as2,F,_);
  equ(as3,L,_);
  equ(as5,UL0,_);
  equ(a*,mL,_);
  equ(ds1,ML0,{mL});
  equ(a3,ML,{UL});
  equ(a4,TL,_);
  equ(a1,P*,_);
  equ(d1,E,{F,L});
}
```

The information appearing on the standard output with verbose output is shown below.

```
***start parsing***
new equ: d1
new var: mL
new edge: (d1,mL,thick)
new var: E
new edge: (d1,E,thin)
new var: F
new edge: (d1,F,thin)
new var: L
new edge: (d1,L,thin)
new equ: d2
new var: uL
new edge: (d2,uL,thick)
already exists(var): F
new edge: (d2,F,thin)
already exists(var): E
new edge: (d2,E,thin)
already exists(var): L
```

```
new edge: (d2,L,thin)
new var: Q
new edge: (d2,Q,thin)
new var: QE
new edge: (d2,QE,thin)
new equ: ds1
new var: ML
new edge: (ds1,ML,thick)
already exists(var): mL
new edge: (ds1,mL,thin)
new var: ML0
new edge: (ds1,ML0,thin)
new equ: ds2
new var: UL
new edge: (ds2,UL,thick)
already exists(var): uL
new edge: (ds2,uL,thin)
new var: UL0
new edge: (ds2,UL0,thin)
new equ: a1
already exists(var): E
new edge: (a1,E,thick)
new var: P*
new edge: (a1,P*,thin)
new equ: a2
already exists(var): QE
new edge: (a2,QE,thick)
new var: TL
new edge: (a2,TL,thin)
new equ: a3
already exists(var): TL
new edge: (a3,TL,thick)
already exists(var): UL
new edge: (a3,UL,thin)
already exists(var): ML
new edge: (a3,ML,thin)
new equ: a4
already exists(var): P*
new edge: (a4,P*,thick)
already exists(var): TL
new edge: (a4,TL,thin)
new equ: as1
already exists(var): Q
new edge: (as1,Q,thick)
new equ: as2
already exists(var): F
```

```
new edge: (as2,F,thick)
new equ: as3
already exists(var): L
new edge: (as3,L,thick)
new equ: as4
already exists(var): ML0
new edge: (as4,ML0,thick)
new equ: as5
already exists(var): UL0
new edge: (as5,UL0,thick)
vertex to delete: as3
***end parsing***

***vertices in B0***
d1       equation
mL       variable
E       variable
F       variable
L       variable
d2       equation
uL       variable
Q       variable
QE       variable
ds1       equation
ML       variable
ML0       variable
ds2       equation
UL       variable
UL0       variable
a1       equation
P*       variable
a2       equation
TL       variable
a3       equation
a4       equation
as1       equation
as2       equation
as3       equation
as4       equation
as5       equation
***END of vertices in B0***

***edges in  B0***
(d1,mL,thick)
(d1,E,thin)
(d1,F,thin)
```

```
(d1,L,thin)
(d2,uL,thick)
(d2,F,thin)
(d2,E,thin)
(d2,L,thin)
(d2,Q,thin)
(d2,QE,thin)
(ds1,ML,thick)
(ds1,mL,thin)
(ds1,ML0,thin)
(ds2,UL,thick)
(ds2,uL,thin)
(ds2,UL0,thin)
(a1,E,thick)
(a1,P*,thin)
(a2,QE,thick)
(a2,TL,thin)
(a3,TL,thick)
(a3,UL,thin)
(a3,ML,thin)
(a4,P*,thick)
(a4,TL,thin)
(as1,Q,thick)
(as2,F,thick)
(as3,L,thick)
(as4,ML0,thick)
(as5,UL0,thick)
***END of edges in B0***

new equ (l1): a*
v1: L
equation to delete (l2): as3
l0: as3
s: L
l0 is isolated
t: a*

The number of components in B: 1

***vertices in B***
d1       equation
mL       variable
E       variable
F       variable
L       variable
d2       equation
```

```
uL       variable
Q       variable
QE       variable
ds1       equation
ML       variable
ML0       variable
ds2       equation
UL       variable
UL0       variable
a1       equation
P*       variable
a2       equation
TL       variable
a3       equation
a4       equation
as1       equation
as2       equation
as4       equation
as5       equation
a*       equation
***END of vertices in B***

***edges in  B***
(d1,mL,thick)
(d1,E,thin)
(d1,F,thin)
(d1,L,thin)
(d2,uL,thick)
(d2,F,thin)
(d2,E,thin)
(d2,L,thin)
(d2,Q,thin)
(d2,QE,thin)
(ds1,ML,thick)
(ds1,mL,thin)
(ds1,ML0,thin)
(ds2,UL,thick)
(ds2,uL,thin)
(ds2,UL0,thin)
(a1,E,thick)
(a1,P*,thin)
(a2,QE,thick)
(a2,TL,thin)
(a3,TL,thick)
(a3,UL,thin)
(a3,ML,thin)
```

```
(a4,P*,thick)
(a4,TL,thin)
(as1,Q,thick)
(as2,F,thick)
(as4,ML0,thick)
(as5,UL0,thick)
(a*,Q,thin)
(a*,L,thin)
***END of edges in B***

***F***
d1mL)
d2uL)
ds1ML)
ds2UL)
a1E)
a2QE)
a3TL)
a4P*)
as1Q)
as2F)
as4ML0)
as5UL0)
***END of F***

***F U {t}***
(d1,mL)
(d2,uL)
(ds1,ML)
(ds2,UL)
(a1,E)
(a2,QE)
(a3,TL)
(a4,P*)
(as1,Q)
(as2,F)
(as4,ML0)
(as5,UL0)
(a*,)
***END of F U {t}***

***vertices in D***
(d1,mL)
(d2,uL)
(ds1,ML)
(ds2,UL)
```

```
(a1,E)
(a2,QE)
(a3,TL)
(a4,P*)
(as1,Q)
(as2,F)
(as4,ML0)
(as5,UL0)
(L,)
(a*,)

s: (L,)
t: (a*,)
***END of vertices in D***

***edges in D***
(d1,mL)->(ds1,ML)
(d2,uL)->(ds2,UL)
(ds1,ML)->(a3,TL)
(ds2,UL)->(a3,TL)
(a1,E)->(d1,mL)
(a1,E)->(d2,uL)
(a2,QE)->(d2,uL)
(a3,TL)->(a2,QE)
(a3,TL)->(a4,P*)
(a4,P*)->(a1,E)
(as1,Q)->(a*,)
(as1,Q)->(d2,uL)
(as2,F)->(d1,mL)
(as2,F)->(d2,uL)
(as4,ML0)->(ds1,ML)
(as5,UL0)->(ds2,UL)
(L,)->(d1,mL)
(L,)->(d2,uL)
(L,)->(a*,)
***END of edges in D****

There is path from s to t.
The list of vertices in the path:
(L,)
(a*,)

***The closest maximum assignment have been
found (in equation-variable order)***
(d1,mL)
(d2,uL)
```

```
(ds1,ML)
(ds2,UL)
(a1,E)
(a2,QE)
(a3,TL)
(a4,P*)
(as1,Q)
(as2,F)
(as4,ML0)
(as5,UL0)
(a*,L)
***END***

The transformed model is of index 1.
```

7 Conclusion and discussion

7.1 Evaluation of results and algorithms

This diploma work was initiated along the lines of a draft algorithm for investigating the effect of simplifications on process models [9]. Furthermore, the resulted exact algorithm has been implemented in C++.

The implemented software is easy to use and well structured, so it could be easily expanded if it would be necessary.

It has been shown and it is important to emphasize that this algorithm can be done in polynomial time. Thus, the program will be useful during simplification of real process models.

A third main goal to reach was to develop adviser functions in the program. Unfortunately this aim has not been completed to the full. Besides of the two very simple cases discussed in **Chapter 4**, only a brute force adviser works at the present stage. The time available for this thesis has not allowed to test several simplification cases, which could serve as a basis for more advanced adviser functions.

7.2 Conclusion and future work

Some improvements can be imagined in relation to this algorithm.

First of all, the capability of the adviser part should be developed further by extending the domain of the equations to be left out. Moreover, different kinds of models should be examined by applying a wide range of simplification transformations. Using these studies, it might be able to derive consequences from engineering viewpoint.

In the present stage the algorithm can only process one simplification transformation in one step, i.e. it is possible to add and to remove only one equation-pair at the same time. It would be beneficial if one could apply more transformations in one step.

In addition, one would like to visualize the transformed graphs and the path in the graph D graphically as it is seen in **Chapter 5**. So the program could be extended with a picture-generator which could denote the different types of edges in different ways.

References

[1] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge (MA), 1990.

[2] K. M. Hangos and I. T. Cameron. A formal representation of assumptions in process modelling. *Computers and Chemical Engineering*, 25:237–255, 2001.

[3] K. M. Hangos and I. T. Cameron. *Process Modelling and Model Analysis*. Academic Press, London, 2001.

[4] A. Krogh Jensen. *Generation of Problem Specific Simulation Models Within an Integrated Computer Aided System*. PhD thesis, Danish Technical University, 1998.

[5] ModKit. Computer aided process modeling. `http://www.lfpt.rwth-aachen.de/Research/Modeling/modkit.html`, 2000.

[6] H. I. Moe. *Dynamic Process Simulation, Studies on Modeling and Index Reduction*. PhD thesis, University of Trondheim, 1995.

[7] K. Murota. *Systems Analysis by Graphs and Matroids*. Springer-Verlag, Berlin, 1987.

[8] G. Stephanopoulos, G. Henning, and H. Leone. MODEL.LA a modeling language for process engineering - 1 the formal framework. *Computers and Chemical Engineering*, 8:813–846, 1990.

[9] Zs. Tuza, G. Szederkényi, and K.M. Hangos. The effect of modelling assumption on the differential index of lumped process models. In *European Symposium on Computer Aided Process Engineering*, volume 12, pages 979–984. Ed. J. Grievink and J. van Schijndel. Elsevier Science, 2002.

[10] J. Unger, A. Kroner, and W. Marquardt. Structural analysis of differential-algebraic equation systems - theory and application. *Computers chem. Engng.*, 19:867–883, 1995.