Minimal Equation Sets for Output Computation in Object-Oriented Models

Vincenzo Manzoni Francesco Casella
Dipartimento di Elettronica e Informazione, Politecnico di Milano
Piazza Leonardo da Vinci 32, 20133 Milano, Italy
{manzoni, casella}@elet.polimi.it

Abstract

Object-oriented models of complex physical systems can have a very large number of equations and variables. For some applications, only a few output variables of the model are of actual interest. This paper presents an application of the well-known Tarjan's algorithm, that allows to automatically select the minimal set of equations and variables required to compute the time histories of selected outputs of a given model. The application of the algorithm to a simple test case is illustrated in the paper.

Keywords: Structural analysis, BLT, reduced-order models

1 Introduction and motivation

Object-oriented models of complex physical systems, that can be handled by state-of-the-art tools, can easily count tens or even hundreds of thousands of equations and variables. It is often the case that many of these variables are computed in order to provide the end user with additional information about the behaviour of the system (e.g., 3D visualization of multibody systems) but are not always needed for some applications of the model. In fact, in some cases, only a very few variables are of actual interest for the user.

For example, consider the full dynamic model of a vehicle, built with the MultiBody library of the Modelica Standard Library. During the development of the model, it is of course interesting to visualize all the details in the motion of the suspension system, also for the sake of verifying the correctness of the model. Consider now two applications of this model: a real-time simulator of the car for pilot training, where the simulator cockpit is moved by actuators in order to somehow reproduce the accelerations that the pilot would feel on the real vehicle, and the design of an active suspension system for the same vehicle.

In the first case, the only data which are actually needed at each time step are the orientation and acceleration of the chassis, in order to compute the simulator cockpit motion, and the position and orientation of the windshield, in order to reconstruct a proper 3D view of the outer environment. In the second case, one may run a large number of simulations with different values of some controller parameters, and evaluate some comfort index based on the vertical acceleration of the pilot seat, which is then the only interesting output of the simulation code. In both cases it is important to avoid computing any variable which is not necessary to compute the required outputs, in order to make the time required for the simulation of each time step as short as possible. This might be essential to stay within the sampling time of the real-time simulator, or to avoid an excessively lengthy simulation session in the second case.

A partial solution to this problem is provided by the Modelica language, that allows to define conditional components. One can then include all auxiliary computations (e.g., for visualization) in such components and turn them off by boolean parameters when not needed. This approach has been used extensively in the MultiBody library. A major drawback of this approach is that it requires a significant additional design effort by the library developer; furthermore, it doesn't guarantee that the minimum number of equations which are necessary for the computation of the required outputs is actually selected for the simulation code generation.

The goal of this paper is then to describe an algorithm, based on the well-known strongly connected algorithm by Tarjan, that automatically selects the minimum number of equations and variables in a model which are required for the computation of the time histories of selected output variables. The paper is structured as follows: Tarjan's algorithm is reviewed in Section 2 and applied to the equation selection prob-

lem in Section 3. Section 4 discusses equation selection in the context of dynamic models, while in Section 5 the algorithm is illustrated with reference to a simple case study. Section 6 gives some concluding remarks.

2 Structural analysis by Tarjan's algorithm

Tarjan's algorithm [6] is a well-known graph-theoretical algorithm; its main purpose is to find the *strongly connected components* of a graph.

A graph is an ordered couple $G = (V, E)$, where V is the *vertex set* (or *node set*) and E is the *edge set* (or *arc set*), such that elements in E are couples of elements in V. If the order of the elements in the couple is important, the graph is called *directed graph* (or *digraph*), *undirected graph* otherwise. In particular, a *bipartite graph* (or *bigraph*) is a graph whose vertices can be divided into two disjoint sets V_1 and V_2, such that every edge connects a vertex in V_1 to one in V_2.

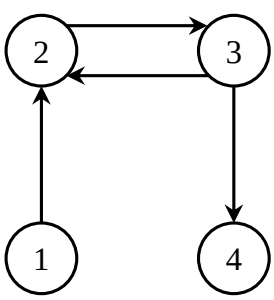
A directed graph is said to be strongly connected if there is a path from v to w for each couple $(v,w) \in V$. In particular, this means that for each $(v,w) \in V$, a path from v to w exists, as well as a path from w to v. The strongly connected components of a directed graph are its maximal strongly connected subgraphs. For further details, see [2].

Even though different algorithms have been proposed in the literature to compute the strongly connected components – such as Kosaraju's algorithm [5] or the Cheriyan-Mehlhorn algorithm [1] – Tarjan's algorithm is the most known and the one which performs better most of the time.

The algorithm is described in detail in [6]. The basic idea is to perform a depth-first search starting from a start node. The strongly connected components form the subtrees of the search tree; their roots are the roots of the strongly connected components. The complexity of the algorithm is $O(|V| + |E|)$.

For the sake of the presentation, the algorithm is now described by means of examples, shown in Figure 1 and Figure 2. The left hand side of the figures is devoted to show the graph, the right hand side instead shows the stack at each step of the algorithm.

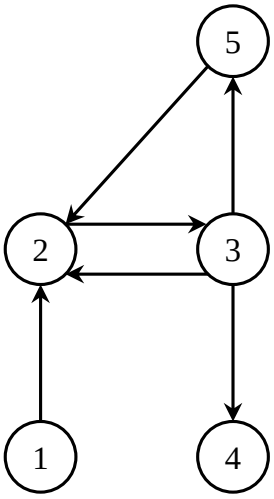
In the first example, the algorithm starts from node 1. In the first three steps, the stack simply records the growing path $1 \to 2 \to 3$. At step 3 we find an edge connecting the node at the top of the stack (node 3) to one lower down (node 2). Since we know that there is a path between 2 and 3, this tell us that $(2,3)$ lies on a closed path. This is recorded by putting a frame



Step	Stack
1	1
2	1 2
3	1 [2 3]
4	1 [2 3] [4]
5	1 [2 3]
6	[1]

Figure 1: An example of Tarjan's algorithm.

around the nodes which belong to the closed path. We now look for unsearched edges at node 3 and we find that 4 is connected to it. There are no more edges from node 4, which also does not have any link to lower nodes. Therefore, node 4 is labeled as a trivial strongly connected component and removed from the stack. Similarly, there are no more edges from the node 3 and from the node 2; the strong component is removed from the stack. The trivial strong component [1] follows. Summing up, the strong components found in this digraph are [4], [2 3] and [1].



Step	Stack
1	1
2	1 2
3	1 [2 3]
4	1 [2 3] [4]
5	1 [2 3 5]
6	[1]

Figure 2: Another example of Tarjan's algorithm.

The second examples shows a more general case. The algorithm starts again from node 1. The first four steps are the same as the previous example. At step 5, node 5 is added to the stack because of the edge $(3,5)$. The edge 5 has a link to a lower node (node 2) which belongs to the strong component. Therefore, node 5 is added to the strong component. Finally, the strong components [1] follows. Summing up, the identified strong components are [4], [2 3 5], and [1].

Step	Stack
1	5
2	5 2
3	[5 2 3]
4	[5 2 3] [4]
5	[5 2 3]

Table 1: The stack of the Tarjan's algorithm applied to the second example, starting from node 5.

Suppose now to apply Tarjan's algorithm on the same graph as in Figure 2, but starting from node 5 instead. Table 1 shows the stack at each step.

Even if the strong components of the graph are obviously the same, Tarjan's algorithm finds only two of them (i.e. [5 2 3] and [4]). In fact, there are no edges that go from a node to node 1, which has only an outgoing edge.

The algorithm can be easily extended to handle this condition, e.g., by restarting from any node not yet considered. However, this situation can be also exploited for other purposes.

For instance, let's suppose that the graph in Figure 2 represents dependencies of objects among each other, where nodes model the objects and each arc (v,w) the relation "v needs w to be evaluated first". It is then worth noting that only the strong components dependent on node 5 are computed.

3 Minimal equation set selection

One application of Tarjan's strongly connected components algorithm is the computation of the *Block Lower Triangular* (BLT) form of an incidence matrix. A Block Lower Triangular matrix is a square matrix such that non-zero square blocks are present on the main diagonal, while all blocks above the diagonal are all zeros. An example of BLT matrix is shown in (1), where all matrices $A_{j,j}$ are square.

$$\begin{pmatrix} A_{1,1} & 0 & \cdots & 0 \\ A_{2,1} & A_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A_{n,1} & A_{n,2} & \cdots & A_{n,n} \end{pmatrix} \tag{1}$$

This forms allows the corresponding set of equations to be solved as a sequence of subproblems; this is particularly convenient for sparse matrices, which are very common in Object-Oriented models.

The incidence matrix of a system of equations can also be represented by a graph. The application of an algorithm to compute the strongly connected components of the graph – like Tarjan's algorithm – allows to determine the BLT transformation. In this section, we show how to apply Tarjan's algorithm for computing both the BLT transformation and the minimal set of equations and variables according to the selected output variables.

In general, an *incidence matrix* is a matrix which shows the relation between classes X and Y of objects. The size of the incidence matrix is $n \times m$, where the number of rows n is the cardinality of the class X and the number of columns m is the cardinality of the class Y. The matrix element (i, j) is 1 if the object i belonging to the class X and the object j belonging to the class Y are in relationship (or incident), 0 otherwise.

When the incidence matrix is applied to systems of equations, the X class represent the equations and the Y class the variables from which they depend. Only square systems are considered in this paper, i.e., $n = m$.

Consider a generic system of equations S, represented in residual implicit form:

$$S: \begin{cases} f_1(z_1, z_2, \ldots, z_n) = 0 & e_1 \\ f_2(z_1, z_2, \ldots, z_n) = 0 & e_2 \\ \vdots & \\ f_n(z_1, z_2, \ldots, z_n) = 0 & e_n \end{cases}, \tag{2}$$

where e_i $(i = 1, 2, \ldots, n)$ represent the equations and z_j $(j = 1, 2, \ldots, n)$ represent the variables of the system. For each equation e_i, the function $f_i(\cdot)$ determines the dependency between the equation and its variables.

The incidence matrix (3) is the structural representation of system (2)

$$\begin{array}{c} \\ e_1 \\ e_2 \\ \vdots \\ e_n \end{array} \begin{array}{c} \begin{array}{cccc} z_1 & z_2 & \cdots & z_m \end{array} \\ \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{pmatrix} \end{array}. \tag{3}$$

The element $a_{i,j}$ is 1 if the residual of equation e_i depends on the value of z_j, 0 otherwise.

An incidence matrix associated to a system of equations can be represented by a bipartite graph $G = (V_1 \cup V_2, E)$, where V_1 is the vertex set which contains the equations (i.e. the rows), V_2 is the vertex set which contains the variables (i.e. the columns) and there exists an arc $(v_i, v_j) \in E$ if the entry (i, j) of the incidence matrix is 1. Such graph is called *equations-variables bipartite graph*, or E-V graph in short.

First, a row permutation of the incidence matrix is computed, such that the value of each entry on the

main diagonal is 1. This has been proven to be equivalent to finding a transversal of the equations-variables graph. A graph transversal is a subset of the edges such that each node belongs only to one arc. This can be computed by using one of the many algorithm in the literature which solve the matching problem, e.g., the Push-relabel algorithm [4] or the Edmonds-Karp algorithm [3].

At the end of the procedure, the incidence matrix will look like:

$$\begin{pmatrix} 1 & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & 1 & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & a_{n-1,n} \\ a_{n,1} & a_{n,2} & \cdots & 1 \end{pmatrix} \tag{4}$$

Finally, Tarjan's algorithm is applied on the graph associated to the diagonal incidence matrix. As we have seen on Section 2, a single run of the algorithm returns different strong components according to the starting node. In particular, if the graph is a dependency graph, only the strong components which depend on the starting node are computed. Therefore, if an output variable is chosen as a starting node, the BLT transformation will only contain the equations and the variables which depend on it.

In order to specify to that one is interested in more than one output variable, a new node s (*source node*) is added to the graph. The node s is then connected by means of outgoing edges to the nodes which represent the output variables. Tarjan's algorithm will then use node s as the starting node. At the equation level, this corresponds to adding to the problem a dummy output variable s and a dummy equation relating s to the required outputs: $s = f_s(y_1, ..., y_h)$.

4 Application to dynamic models

In the context of object-oriented modelling, continuous-time systems are represented by means of differential-algebraic equations. After flattening and index reduction, the system is described by $n+m$ differential equations:

$$\begin{cases} f_1(x_1,\ldots,x_n,\dot{x}_1,\ldots,\dot{x}_n,y_1,\ldots,y_m) = 0 \\ f_2(x_1,\ldots,x_n,\dot{x}_1,\ldots,\dot{x}_n,y_1,\ldots,y_m) = 0 \\ \vdots \\ f_{n+m}(x_1,\ldots,x_n,\dot{x}_1,\ldots,\dot{x}_n,y_1,\ldots,y_m) = 0 \end{cases}, \tag{5}$$

where x_i $(i = 1,\ldots,n)$ are the state variables and y_j $(j = 1,\ldots,m)$ are the algebraic variables. If the states x_i are known at a certain time instant, these equations can be solved to compute the derivatives and the algebraic variables at the same time. However, for the sake of the equation selection, what really matters is which equations are strictly necessary to compute the *trajectories during time* of the selected output variables, not only at a given initial time t_0, but for an entire interval $t_0 \le t \le t_f$. Therefore, it is necessary to consider the implicit relationship between each variable x_i and its derivative $\dot{x}_i$, since the latter is uniquely determined once the time history of the former is known.

The incidence matrix for the equation selection algorithm can therefore be set up as follows: the set of variables z_i is given by the state variables x_i, by their derivatives $\dot{x}_i$, and by the algebraic variables y_j; the set of equations e_i is given by the set (5), augmented by n dummy equations, each relating a state variable x_i with its derivative $\dot{x}_i$. The algorithm described in Section 3 is then applied to the resulting E-V graph.

If the object-oriented model is hybrid, i.e., it also contains discrete variables and discrete equations that are active only at events (inside when-clauses in Modelica), the above-described procedure can be suitably extended. In this case, the set of variables z_i should also include the discrete state variables q_h $(h = 1,\ldots,u)$, their previous values $pre(q_h)$, and all other discrete variables r_k $(k = 1,\ldots v)$, while the set of equations should also contain all the $u+v$ discrete equations contained inside the when clauses, as well as u dummy equations relating each discrete state variable q_h with its corresponding previous value $pre(q_h)$.

5 Case study

A simple problem is now used to explain how the algorithm works. Consider the continuous-time dynamical model (6). It has 3 state variables (x_1, x_2, and x_3) and two algebraic variables (y_1 and y_2).

$$\begin{cases} \dot{x}_1(t) = -x_1(t) \\ \dot{x}_2(t) = x_1(t) - x_2(t) \\ \dot{x}_3(t) = x_1(t) \\ y_1(t) = 3x_2(t) + x_1(t) \\ y_2(t) = 2x_3(t) \end{cases}. \tag{6}$$

For sake of conciseness, hereafter the time dependency is omitted.

The system is written in explicit form. It is apparent that the value of the algebraic variable y_1 does not depend on the value of the state variable x_3, neither directly nor indirectly. Similarly, the value of the algebraic variable y_2 does not depend on the value of the state variable x_2.

The system is now rewritten in implicit form (7); e_5, e_6 and e_7 are the three dummy equations added to represent the implicit relationship between each variable and its derivative.

$$\begin{cases} \dot{x_1} + x_1 & = 0 & e_0 \\ \dot{x_2} - x_1 + x_2 & = 0 & e_1 \\ \dot{x_3} - x_1 & = 0 & e_2 \\ y_1 - 3x_2 - x_1 & = 0 & e_3 \\ y_2 - 2x_3 & = 0 & e_4 \\ x_1 - f(\dot{x_1}) & = 0 & e_5 \\ x_2 - g(\dot{x_2}) & = 0 & e_6 \\ x_3 - h(\dot{x_3}) & = 0 & e_7 \end{cases} \tag{7}$$

The incidence matrix associated to the system (7) is shown in (8).

$$\begin{array}{c} \\ e_0 \\ e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \\ e_7 \end{array} \begin{array}{c} \begin{array}{cccccccc} x_1 & x_2 & x_3 & \dot{x_1} & \dot{x_2} & \dot{x_3} & y_1 & y_2 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array} \tag{8}$$

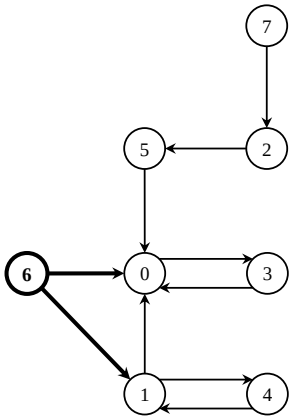
The first step of the algorithm is to find a permutation of the rows of the matrix (8) such that the resulting matrix is diagonal. As outlined in Section 3, this can be done finding a transversal of the E-V graph. The diagonal matrix is shown in (9).

$$\begin{array}{c} \\ e_0 \\ e_1 \\ e_7 \\ e_5 \\ e_6 \\ e_2 \\ e_3 \\ e_4 \end{array} \begin{array}{c} \begin{array}{cccccccc} x_1 & x_2 & x_3 & \dot{x_1} & \dot{x_2} & \dot{x_3} & y_1 & y_2 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \tag{9}$$

The rows and the columns of the diagonal matrix (9) are now relabeled to ease the definition of the digraph. The symbol ${}^j e_i$ indicates that the equation e_i lies in the j-th row of the diagonal matrix. Similarly, ${}^j x_i$ indicates that the variable x_i lies in the j-th column. Matrix (10) shows the result of the relabeling.

$$\begin{array}{c} \\ {}^0e_0 \\ {}^1e_1 \\ {}^2e_7 \\ {}^3e_5 \\ {}^4e_6 \\ {}^5e_2 \\ {}^6e_3 \\ {}^7e_4 \end{array} \begin{array}{c} \begin{array}{cccccccc} {}^0x_1 & {}^1x_2 & {}^2x_3 & {}^3\dot{x_1} & {}^4\dot{x_2} & {}^5\dot{x_3} & {}^6y_1 & {}^7y_2 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \tag{10}$$

Assume now that only the value of y_1 is of interest as a system output. Figure 3 shows the graph associated to the matrix (10). The output y_1 is represented by the node with a bold border. Bold arrows connect the output variable to the state variables which have a direct impact on it.



Step	Stack
1	6
2	6 0
3	6 0 3
4	6 1
5	6 1 4
6	6

Figure 3: Graph associated to matrix (10).

Tarjan's algorithm is now applied to this graph, starting from node number 6. The algorithm's stack is shown on the right hand side of the same figure. The strong components identified by the algorithm are $\boxed{0\ 3}$, $\boxed{1\ 4}$ and $\boxed{6}$, which correspond to equations $e_5 e_0$, $e_6 e_1$, and e_3, respectively.

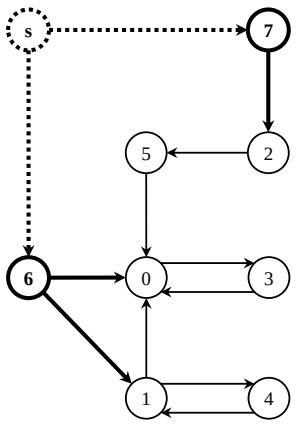
The incidence matrix output of the algorithm is given in (11). The order of the system has been reduced, since the state variable x_3 and its derivative, as well as the algebraic variable y_2, do not contribute either directly or indirectly to the value of the output variable y_1. Moreover, the procedure also returns the

incidence matrix in BLT form.

$$\begin{array}{c} \\ e_5 \\ e_0 \\ e_6 \\ e_1 \\ e_3 \end{array} \begin{array}{c} \begin{array}{ccccc} \dot{x_1} & x_1 & \dot{x_2} & x_2 & y_1 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix} \end{array} \tag{11}$$

As noted in Section 3, more than one output variable can be specified. For instance, assume that we are now interested in the time histories of both y_1 and y_2. The graph is modified by adding the source node s. This node is connected to the interested output variables, in our case to the node 6 (the output variable y_1) and node 7 (the output variable y_2). The source node can also be seen as dummy output variable $y_s(t)$ appended to the system, whose value is a function of the output variables of interest $y_1(t)$ and $y_2(t)$, so that $y_s(t) = f_s(y_1(t), y_2(t))$.

The graph augmented with the source node and the appropriate edges is represented in Figure 4.



Step	Stack
1	s
1	s 6
2	s 6 0
3	s 6 [0 3]
4	s 6 1
5	s 6 [1 4]
6	s [6]
7	s 7
8	s 7 2
9	s 7 2 [5]
10	s 7 [2]
11	s [7]
12	[s]

Figure 4: Graph with the source node s.

It is apparent from (7) that in order to compute the value of both the output variables, all the state variables are required. In fact, Tarjan's algorithm returns the matrix (12) which has the same dimension of the original one.

$$\begin{array}{c} \\ e_5 \\ e_0 \\ e_6 \\ e_1 \\ e_3 \\ e_2 \\ e_7 \\ e_4 \end{array} \begin{array}{c} \begin{array}{cccccccc} \dot{x_1} & x_1 & \dot{x_2} & x_2 & y_1 & \dot{x_3} & x_3 & y_2 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array} \tag{12}$$

6 Conclusions

This paper presents application of Tarjan's algorithm to determine the minimal set of variables and equations in an object-oriented model, which are strictly necessary to compute the time histories of selected output variables. The algorithm has been thoroughly illustrated in a simple test case.

This feature can be easily implemented in all Modelica tools and can be very valuable for end-users, when their application does not require to inspect all the model variables and puts a premium on fast simulation performance. In particular, it is planned to implement this feature in the OpenModelica compiler.

Significant applications include real-time code generation, sensitivity or parameter-sweep analysis, and in general all control-oriented applications where the input-output behaviour of the system is of interest.

A particularly nice application could be the case of planar multibody systems, built with the standard Modelica MultiBody library. If only outputs corresponding to the in-plane movement of some points of the system are selected, the procedure illustrated in this paper could allow to remove all the out-of-plane equations of motion, which would then be irrelevant, thus allowing a substantial reduction in the number of equations and state variables of the system.

References

[1] J. Cheriyan and K. Mehlhorn. Algorithms for dense graphs and networks on the random access computer. *Algorithmica*, 15(6):521–549, 1996.

[2] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*, chapter 22, pages 527–529. MIT Press and McGraw-Hill, second edition, 2001.

[3] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

[4] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.

[5] R.S. Kosaraju. Unpublished, 1978.

[6] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146, 1972.