Survey of appropriate matching algorithms for large scale systems of differential algebraic equations

Jens Frenkel[1] Günter Kunze[1] Peter Fritzson[2]
[1]Dresden Technical University, Institute of Mobile Machinery and Processing Machines
[2]PELAB - Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden
{jens.frenkel, guenter.kunze}@tu-dresden.de,
peter.fritzson@liu.se

Abstract

This paper presents a survey on matching algorithms which are required to translate Modelica Models. Several implementations of matching algorithms are benchmarked on a set of physical models from mechanical systems in ODE and DAE representation. The major part of algorithms is based on the Augmenting Paths Method and one algorithm is based on the Push-Relabel Method. The algorithms are implemented in the programming language C and MetaModelica. In addition two cheap matching algorithms are used to jump-start the advanced matching process.

Keywords: matching; index reduction; modelimark

1 Introduction

A major benefit of Equation based Object Oriented modeling Languages (EOOL) like Modelica is the possibility of acausal modeling. It increases the reusability of models and simplifies the description of physical systems. In order to simulate an acausal model, all equations have to be transformed and sorted yielding a causal model description. The process of transforming equations into assignments is thus called *causalization*. The main task of causalization is to match each equation to a variable. It is one of the most important challenges of any EOOL compiler.

Most models from EOOL give rise to very large and sparse differential algebraic equation (DAE) systems [19],[20],[25]. The challenge of the matching process is therefore to transform the model into an ordinary differential equation (ODE), so that it can be solved through the application of standard numerical time integration algorithms.

Pantelides [21] provides an algorithm to get a so called *perfect matching*, transforming the system to block lower triangular form (BLT) providing all necessary information to apply index reduction and thereby transforming a DAE into an ODE. Driven by the need of numerical stability several index reduction algorithms have been developed in the past [16],[18],[19],[20],[25],[27].

There are other matching algorithms next to those presented by Duff [4]. They can be divided into different classes of worst case time complexities. The most common complexities are shown in Figure 1 [1][2].

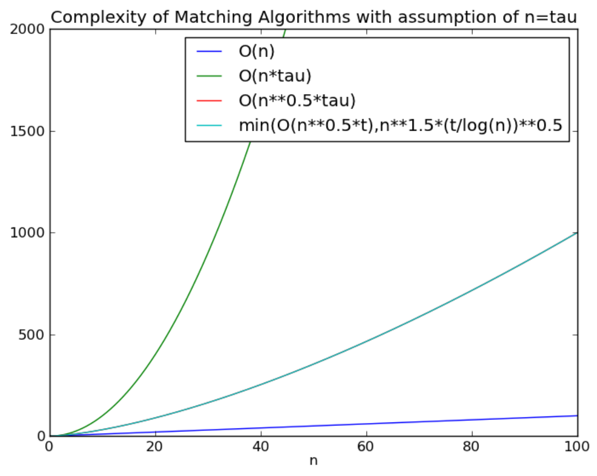

Figure 1: Typical worst case complexities of matching algorithms [9]

Since more powerful computers allow for larger models with more equations, a future challenge will be to optimize the scaling of EOOL compilers with respect to model size. As shown in [11] the effort of state of the art EOOL compilers is proportional to the

[1]n: Number of Equations
[2]τ: non zero entries in the Adjacency Matrix

second or even the third power of the number of equations, depending on the model structure. Thus it is worth studying how the combination of matching and Pantelides Algorithm can be further optimized.

The next section provides a brief introduction to matching theory and index reduction. It is followed by an overview on selected matching algorithms based on augmenting paths and the push relabel technique. Section 4 discusses the possibility to combine the matching algorithms with index reduction by looking at some examples. A comparison of runtimes of all these algorithms is presented in section 5 followed by a discussion and concluding remarks in section 6.

2 Theory of Matching and Index Reduction

2.1 Matching Theory

The aim of this section is to give an introduction to the general definitions of matching algorithms. For further information, the reader is referred to [5],[6],[8],[9],[2],[10]. As mentioned above and shown in detail by Elmqvist [10] matching algorithms are provide the information how a system of equations can be transformed symbolically into a system of assignments. The mathematical idea behind this, is to transform the system into block lower triangular (BLT) form and to solve it by a simple forward substitution process [5]. As Duff proposed in [5] the transformation to BLT form is split into two stages:

- Match each equation to a variable and transform the problem description into a directed graph
- Find a traversal of the directed graph which means to sort the equations and identify algebraic loops

For the second step Tarjan's Algorithm [26] is very efficient and offers time linear complexity with respect to the number of equations [6]. To understand the first step one has to look at the *Adjacency Matrix* of a system of equations. The rows of the Adjacency Matrix correspond to the equations whereas the columns correspond to the variables of the system. The Adjacency Matrix has an entry (=1) at row i and column j, iff equation i contains variable j. The number of entries in the Adjacency Matrix is denoted with τ. For a nonsingular system, the matching algorithm finds an unsymmetric permutation which produces a zero-free main diagonal. The set of all nonzero entries on the main diagonal is called a *transversal*. A set containing the maximum number of nonzero elements is called a *maximum transversal*. A simple example is shown in Figure 2.

$$a)\ \begin{array}{rl} b+c= & 0 \\ a= & 10 \\ a+c= & 2 \end{array}\ b)\begin{pmatrix} 0 & \mathbf{1} & 1 \\ \mathbf{1} & 0 & 0 \\ 1 & 0 & \mathbf{1} \end{pmatrix}\ c)\begin{pmatrix} \mathbf{1} & 0 & 0 \\ 1 & \mathbf{1} & 0 \\ 0 & 1 & \mathbf{1} \end{pmatrix}$$

Figure 2: Equation System (a) with Adjacency Matrix (b) and permuted matrix in BLT form (c) from matching highlighted in boldface.

The Adjacency Matrix can also be presented as a bipartite graph with one set of nodes representing equations (green) and another representing variables (yellow). The edges of the graph represent the nonzero entries in the Adjacency Matrix. For the simple example presented above in Figure 2 the bipartite graph is shown in Figure 3.

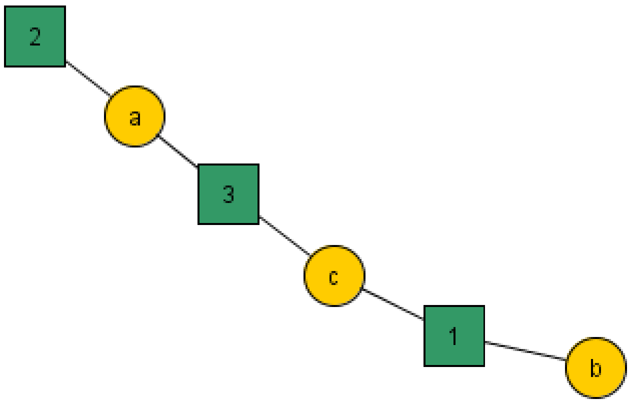

Figure 3: Bipartite graph for the example from Figure 2

A set of matched equations and variables is called *matching* or *assignment block*. If no additional matches can be found, the matching is called *maximum*. In case of a square matrix the matching is *complete* (*perfect*) if all equations are matched. In case of a non-square matrix the matching is complete if either all equations or all variables could be matched. A sequence of connected nodes is called *path*. If each of the nodes on a path belong to the matching, then it is called an *alternating path relative to an assignment*. If the alternating path has an unmatched equation at one end and an unmatched variable at the other end it is called a *augmenting path*. In such a case the matching could be increased by one if all assignments from the path are removed from the matching and all other assignments from the path are added. This procedure is called *reassignment* or *rematching* [8].

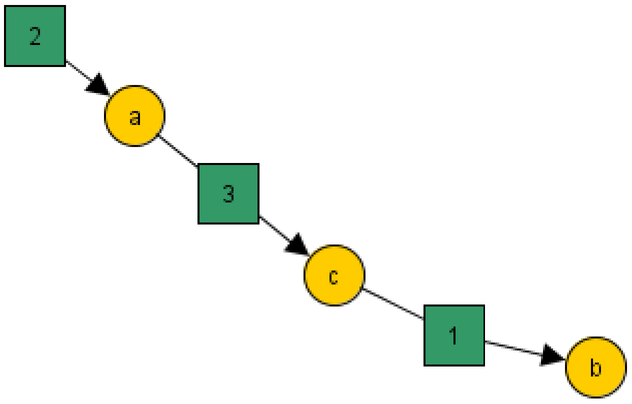

Figure 4: Matched Bipartite graph for the example from Figure 2 with alternating path M= {(2,a),(3,c),(1,b)}

2.2 Index Reduction

In case of a DAE system with differential index $v_d > 1$ [[27], Definition 2.1] no complete matching can be found. If the system is not structurally singular an appropriate symbolical index reduction algorithm must be employed to reduce the differential index v_d to at least one.

As mentioned in [27] and [25] several symbolical methods for index reduction are available. The graph-theoretical algorithm from Pantelides with improvements from Soares and Secchi [25] is most commonly used.

Pantelides' approach is to find a minimal structurally singular (MSS) subsets of equations. The equations of the subset are differentiated and replaced by their derivatives. The algebraic variables which get derived with respect to time in the process are marked as states and only their derivatives are considered for the next matching cycle. With the criterion, that the number of new equations generated through differentiation must not exceed the number of variables in the new subset, structural singular systems are detected and the algorithm terminates with an error. Due to the removed algebraic relations between the dynamic variables of the system and the algebraic variables marked as states the calculated results will be unusable. Appropriate algorithms to cover this issue are presented by several authors [16],[18], [19],[20].

3 Matching Algorithms

Since Pantelides' Algorithm does not rely on a particular matching algorithm, it is worth comparing different algorithms within that context. Guided by [9],[14],[24] a set of promising matching algorithms has been selected. While the majority of algorithms is based on a search for augmenting paths, one algorithm is employs a push-relabel strategy, designed for maximum flow problems [12],[14]. Since bipartite matching is a special case of the maximum flow problem, push-relabel might be well suited to solve the matching problem [14].

3.1 Augmenting Paths Based Algorithms

3.1.1 DFS

The depth first search based matching algorithm (DFS) applies a depth first search on each unmatched column to find an augmenting path. To avoid double visits an array of size m - the number of rows - is used. The augmenting path can be retrieved from the stack of the DFS. The stack is used to backtrack after visiting all nodes and has the same size as the number of columns n. To improve the performance, an additional array of size n is used to keep the information of the last visited row for each column. In summary the algorithm needs $2n+m$ additional space to the memory for storing the assignments. Please note, that only the Adjacency Matrix but not its transpose is required, since the algorithm traverses only from columns to rows.

3.1.2 BFS

The breadth first search based matching algorithms (BFS) use a breadth first search for each unmatched column to find an augmenting path. The additional space consumption of a good implementation is $n+2m$. A queue of size n is needed to store the columns to visit next as well as an array of size m to mark the visited rows. The augmenting path is stored in an additional array of size m, saving the parent column to each row. Analogous to the DFS only the Adjacency Matrix is need for BFSB.

3.1.3 MC21A

The MC21A algorithm is based on a DFS with an additional look ahead mechanism. The look ahead mechanism first checks all rows of a column for an unmatched variable before going deeper. Implementing the look ahead mechanism requires an additional array of size n for the check. In total the implementation needs $3n+m$ additional space.[4][7]

3.1.4 PF

The algorithm by Pothen and Fan (PF) is very much alike MC21A. The difference lies in the usage of the visited flag. A PF phases starts with a queue of size n of all unmatched columns. On each column a DFS with look ahead is applied. The flag *visited* is not reset after the search. The column is dequeued if it is matched. The PF phases are applied until all columns are removed from the queue. The additional space is $4n+m$ and again only the Adjacency Matrix is need for PF.[23]

3.1.5 PF+

PF+ is a simple extension to PF by [9]. To decrease the sensitiveness of the algorithm for row and column permutations the traversal direction of the rows alternates. The additional space consumption is $4n+m$ as in PF.[9][14]

3.1.6 HK

The algorithm by Hopcroft and Karp (HK) is organized in phases comprising two parts. The fist part is a BFS from all unmatched columns to assign level numbers to the rows. The level numbers indicate the shortest path length from a row to an unmatched column. In the second part the level numbers are used to increase the assignments with a DFS. It is only allowed to traverse columns with decreasing level numbers. The additional space consumption is $2n+2m$ (stack(m),queue(n),nextcol(m),levels(m)). Note, since HK uses both BFS and DFS both the Adjacency Matrix and its transposed are required.[13][3]

3.1.7 HKDW

HK modified by Duff and Wiberg (HKDW) adds a third part to the HK phase. The third part is a DFS in the full graph for each remaining unmatched row to increase the matching. The flag *visited* is not reset between two DFS in part three. The additional space consumption with $2n+2m$ is similar to HK because the additional DFS needs no further memory. [8]

3.1.8 ABMP

The algorithm by Alt et al. (ABMP) is organized in two phases. The fist phase increases the matching by a sophisticated search procedure combining BFS and DFS. This phase is performed until the lower bound on the shortest augmenting path length exceeds a suitable value. Alt et al. suggest to use the bound $L=\sqrt{\tau \log n/n}$.[9] The additional space consumption is $2n+2m$.[1]

3.2 Push Relabel Based Algorithms

Push Relabel Algorithms are developed to solve the problem of maximum flow in networks. The idea behind is not to find augmenting paths but to search and augment together. Based on a set of rules speculative augmentations are performed by unmatching and matching.[14][24]

3.2.1 PR

A detailed description of the implemented push relabel algorithm can be found in [14]. The algorithm uses the same mechanism like PF+ to traverse the adjacency list in alternating order called fairness. The push order to select active columns for pushing is first-in-first-out (FIFO). The additional space consumption is $2n+m$ (row label(m),column label(m),queue(m)) and the Adjacency Matrix as well as its transposed are need.

3.3 Heuristic Based Algorithms

Next to the systematic algorithms discussed above, there are algorithms based on heuristics which are designed to increase the performance of a matching process. They are called cheap matching and their benefits strongly depend on the structure of the problem. Thus they are used as an initial guess or jump-start. In [9] a comprehensive overview on cheap matching algorithms is given. Based on the results from [9] two heuristics are selected for testing. The frequently used and the best one.

3.3.1 Cheap Matching

The cheap matching algorithm traverses all columns and matches the first unmatched row in the adjacency list of the column. The complexity of the algorithm is $O(n+\tau)$.

3.3.2 KS Rand Cheap Matching

The cheap matching algorithm by Karp and Sipser introduces a heuristic based on constructing a smaller graph through two rules and a random matching. More information can be found in [9].

3.4 Adaptability for Index Reduction

The matching algorithms discussed above can be classified based on their behaviour when encountering singular systems. While the *simple* matching algorithms terminate as soon as a single node cannot be assigned, the advanced algorithms terminate with a non empty set of unassigned nodes. Some of them allow the set to be collected in a post processing step.

- Simple Matching Algorithms
  - DFSB
  - BFSB
  - MC21A
- Advanced Matching Algorithms
  - PF
  - PF+
  - HK
  - HKDW
  - ABMP
  - PR

In the original paper of Pantelides, the matching algorithm MC21A by Duff was used. MC21A belongs to the group of simple algorithms. Hence no changes have to be made to the Pantelides Algorithms for simple matching algorithms.

In case of a simple matching algorithm the MSS subset contains exactly one unmatched equation. The other equations of the subset are found by a search in the matched graph starting from the variables of the unmatched equation. During the search, each variable is visited only once. For all presented simple algorithms the search to get the MSS subset is not an extra step, it is found by storing the visited equations in each phase of the algorithm.

In case of an advanced matching algorithm, a search in the matched graph is necessary for each equation to get the MSS subsets. Each subset has to fulfil the criterion, that the number of new equations generated by differentiation must not exceed the number of variables in the new subset. Hence, obtaining the MSS subset is more costly compared to simple algorithms as the search is an extra step.

4 Measurements on Examples

Since there is no comparison of matching algorithms in the field of Modelica known to the author an extensive survey has been conducted. Therefore each matching algorithm has been implemented into the OpenModelica compiler (OMC) [3]. In order to be compatible with both simple and advanced matching algorithms the Pantelides index reduction had to be reimplemented modifying the interfaces and the compilation process. Since there exists only little experience about the runtime efficiency and comparability of MetaModelica [22], in which the OMC is written, an external C implementation of freely available matching algorithms [15] has been embedded as well.

The aim of this paper is to compare the computational effort of the matching algorithms with and without index reduction using selected examples. In addition the influences of the programming language and the usage of a cheap matching algorithm are investigated.

All measurements were accomplished using a Windows 7, 64 Bit System with Intel Core i7 860, 2.80 GHz and 8.0 GB RAM.

4.1 Examples

To do an extensive comparison of matching algorithms scalable Modelica models are needed. Since the author is mainly concerned with multi body systems, the following mechanical models will be used:

- chain structure Figure 5 (a)
- tree structure Figure 5 (b)
- grassland structure Figure 5 (c)
- kinematic loops 5 (d)

The models are based on the Modelica.Mechanics.Multibody library (MSL 3.1), the Planar Mechanics Library from DLR[4] and PyMbs [17]. PyMbs[5] is a Python based multi body tool to generate the equations of motion from a description similar to Modelica.Mechanics.Multibody. PyMbs generates efficient flat Modelica code which places very low demands on the EOOL compiler. Hence no index reduction step is necessary and one obtaints a benchmark for pure matching. The reason to use three

[3] www.openmodelica.org
[4] http://www.robotic.de/339
[5] http://sourceforge.net/projects/pymbs/

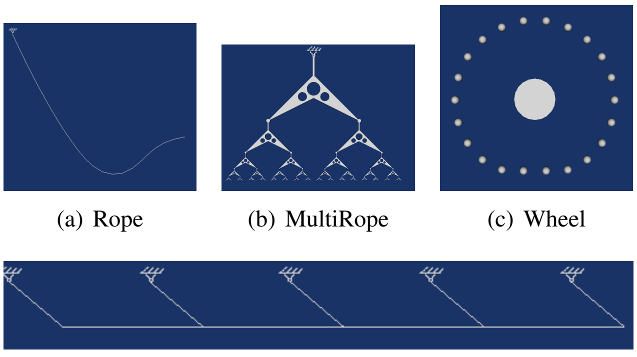

Figure 5: Example Models

different descriptions is to study the influences of the way a model is set up.

In addition to the four models, most examples from the Modelica.Mechanics package are used for the comparison with index reduction.

4.2 Results for Pure Matching

The results for pure matching on the rope model are presented in Figure 6 and Figure 7. Most of the algorithms show a linear relationship between effort and model size. The represented model size is the number of equations the matching algorithm operates on. Note, that this is the reduced size of the model. Because it was important for the benchmarks to be comparable with the usual modelling process all steps, for example the detection of simple equations like $a = b$ and $a = constant$ are performed before matching.

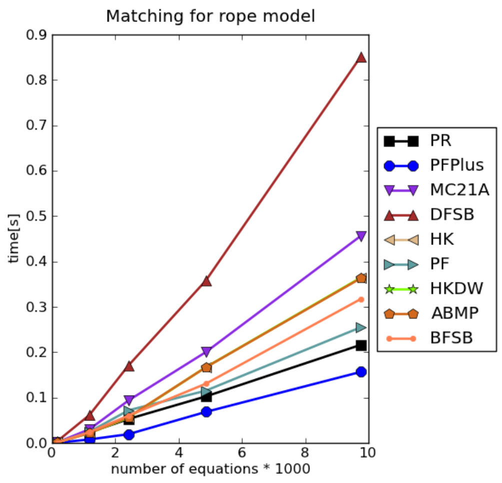

Figure 6: Results from Rope examples, MetaModelica implementation

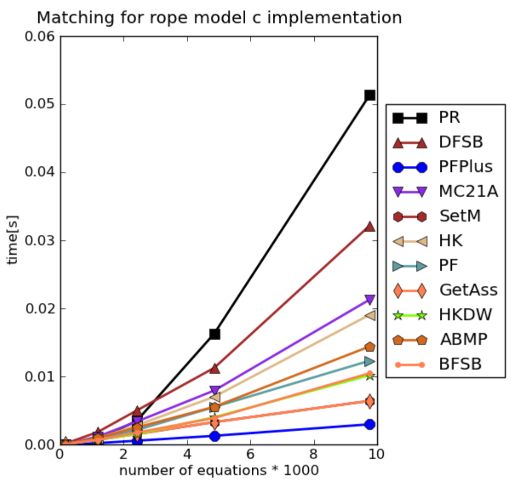

Figure 7: Results from Rope examples, C implementation

The PF+ algorithm is the fastest, while the simple DFS algorithm needs the most time. The PR algorithm is the second fastest, only beaten by PF+. While the MetaModelica implementation suggests that the push relabel algorithm seems to be very efficient, results from the C implementation show a different picture 7. Here the PR scales non-linear and needs the most time. Again, the DFS is slowest and the PF+ is the fastest augmentation path based algorithm. Generally speaking, the C implementation is around ten times faster than the MetaModelica implementation, including the time to pass the incidence Matrix (SetM) and to return the assignments (GetAss) as shown in Figure 7. Copying the Incidence Matrix and returning the Assignments takes twice the time needed to match the system using the PF+ algorithm, rendering the overall time similar to the fastest MetaModelica implementation. Figure 8 and Figure 9 show the results for the MultiRope model. Again, PF+ is the fastest, DFS needs most time and the C implementation is around 10 times faster.

Figure 10 show the results for the wheel example. Here some algorithms scale non-linear in time and a few scale linear. Still, PF+ is one of the fastest algorithms and DFS needs the most time.

The results for the kinematic loop model are shown in Figure 11. Here, the fastest algorithm is HK closely followed by HKDW. Nonetheless, PF+ still belongs to one of the fastest algorithms.

In summary the fastest overall algorithm in case of pure matching is the PF+ algorithm. It scales linear in time for all test cases and therefore seems well suited for large scale systems.

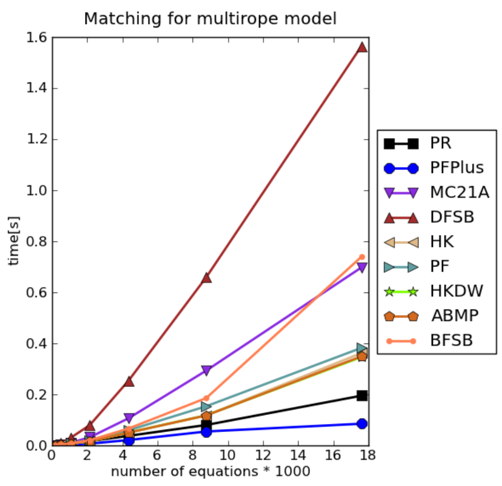

Figure 8: Results from MultiRope examples, MetaModelica implementation

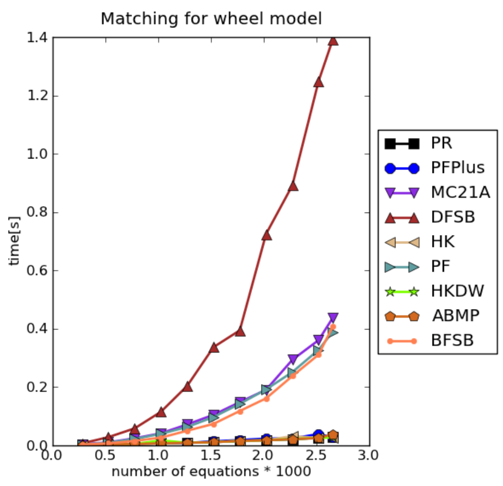

Figure 10: Results from Wheel examples, MetaModelica implementation

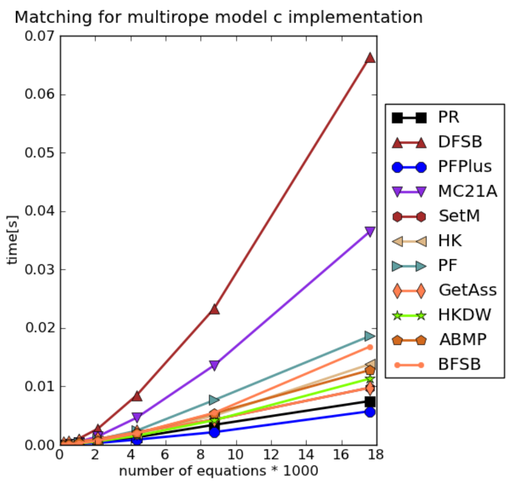

Figure 9: Results from MultiRope examples, C implementation

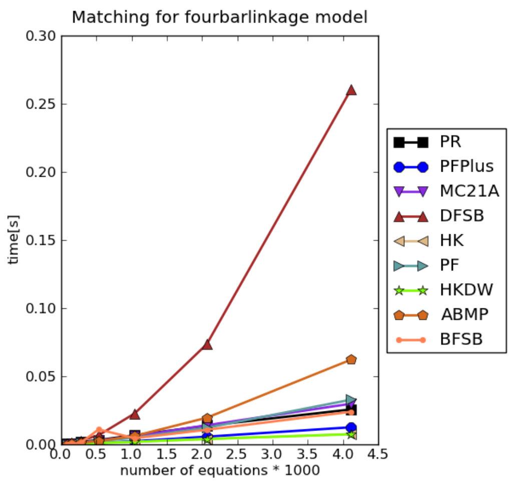

Figure 11: Results from FourBarLinkage examples, MetaModelica implementation

4.3 Results for Matching and Index Reduction

The result for the rope model is shown in Figure 12 and 13. Again PF+ is one of the fastest algorithm and scales linear in time. Since all other models do not show a mentionable difference their results are not shown explicitly. Please note, that due to the lower demands on the EOOL compiler, the OMC manages to process models of up to 200 bodies when described with PyMbs. The upper boundary for the MSL lies at around 50 bodies.

In addition to the models presented above Figure 14 shows the results for the examples included in the package Modelica.Mechanics. The results are presented with a logarithmic time axis. The grey curves represent linear relationships between time and number of equations. The suffix *Ext* marks the C implementation. Because some models have roughly equal numbers of equations, the graph looks quite scattered. Again, PF+ is one of the fastest algorithm and scales linear in time.

4.4 Results for Cheap Matching

The results from the usage of heuristic algorithms are shown in Figure 15 and 16 for the cheap matching and in Figure 17 and 15 for the KS cheap matching algorithm. It can be seen that especially the BFS and DFS MetaModelica implementations benefit from the usage of a cheap matching algorithm. The time saved

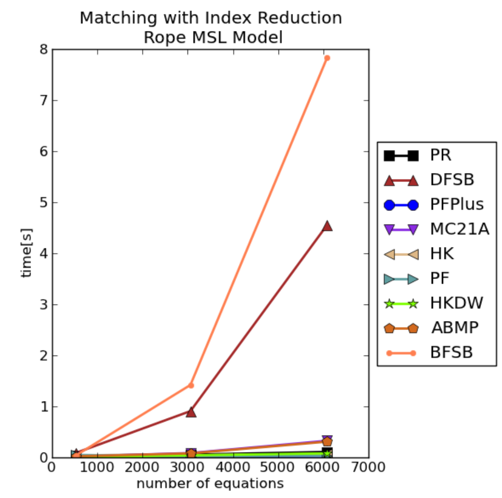

Figure 12: Results from Rope MSL examples, MetaModelica implementation

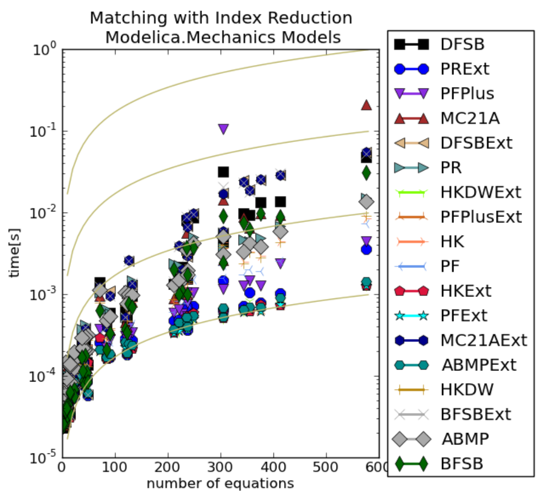

Figure 14: Results from Matching with Index Reduction for Modelica.Mechanics Example Models

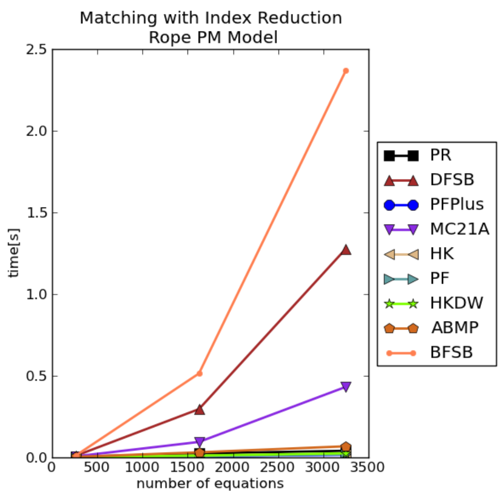

Figure 13: Results from Rope PM examples, MetaModelica implementation

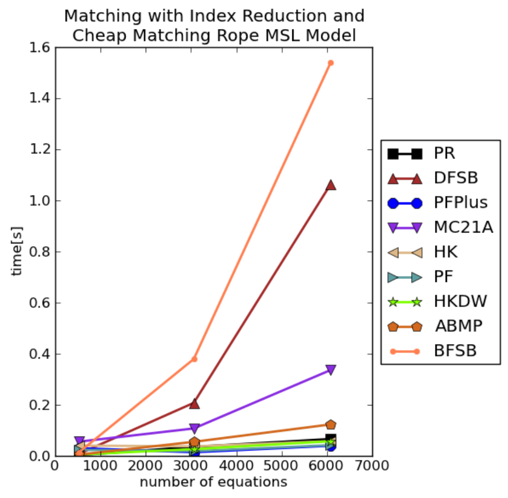

Figure 15: Results from Rope MSL examples, MetaModelica implementation

for both algorithm is around 80%.

5 Conclusion

An extensive survey has been conducted by the author to find the best suited matching algorithm for EOOL compilers. Several real life models have been used for testing. It was found that that the PF+ algorithm performed best on almost all models.

Moreover, it has been found that the PF+ algorithm, although it has a non-linear worst case time complexity, scales linear for the models tested within this survey. This makes it ideally suited for the application in large scale models. Unfortunately, further increase in model size, to support that claim, was hindered due to the memory consumption of the OpenModelica compiler. Future work will aim at increasing the manageable model size and rerun the benchmarks.

It could also be shown that MetaModelica seems not to be well suited for such algorithms since the C implementation is at least 10 times faster. Maybe some further language and compiler features could decrease the time difference to a natural C implementation. The main difference of implementation is caused by the storage of the Adjacency Matrix. The C implementation uses an array to store the values and an additional array to store the column indices. In MetaModelica the matrix is stored as an array of lists. To traverse the lists in MetaModelica recursive function calls are needed whereas the c implementation simply stores

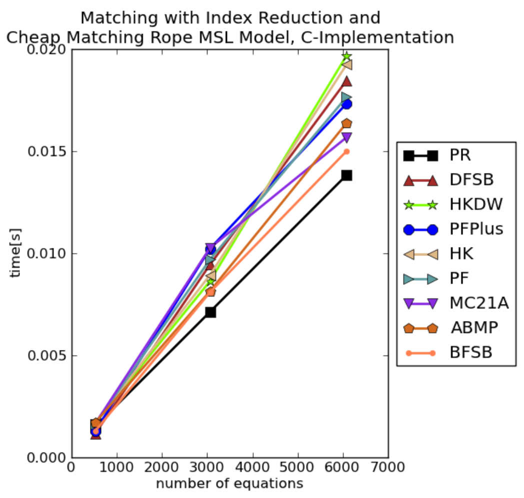

Figure 16: Results from Rope MSL examples, C implementation

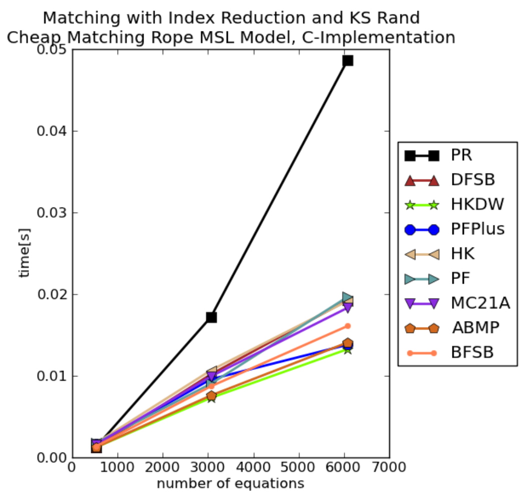

Figure 18: Results from Rope MSL examples, C implementation

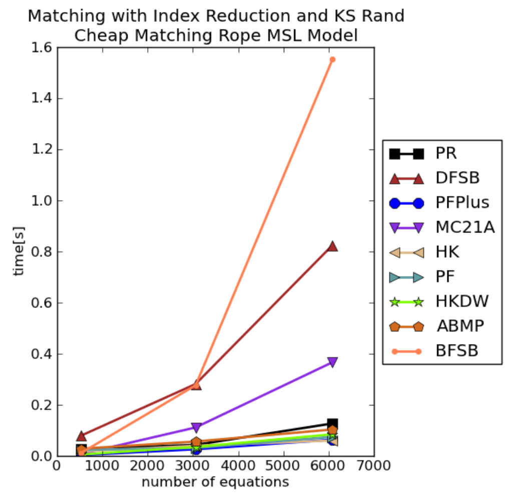

Figure 17: Results from Rope MSL examples, MetaModelica implementation

the needed indices for the traversal in arrays.

Since the implementation is freely available in the OpenModelica Compiler, the survey may be extend with models from other physical domains.

References

[1] Alt, H.; Blum, N.; Mehlhorn, K.; Paul, M.: Computing a maximum cardinality matching in a bipartite graph in time $O(n^{1.5}\sqrt{m/log(n)})$, Information Processing Letters, Volume 37, Issue 4, 28 February 1991, Pages 237-240

[2] Berge, C. The Theroy of Graphs. Methuen, London, 1962

[3] Blum. N.: A simplified realization of the Hopcroft-Karp approach to maximum matching in general graphs. Technical report, Universität Bonn, 1999.

[4] Duff, I. S. On algorithms for obtaining a maximum transversal. ACM Trun.s. Math. Softw. 7(1981), 315-330.

[5] Duff, I.S.; Erisman, A.M.; Reid, J.K.: Direct methods for sparse matrices,1986,Clarendon Press Oxford

[6] Duff, I. S.; Reid J. K.; Harwell, A.: An implementation of Tarjan's algorithm for the block triangularization of a matrix,in ACM Trans. Math. Software Volume 4, pp. 137-147, 1978

[7] Duff, I. S.: Algorithm 575: Permutations for a Zero-Free Diagonal [F1]. ACM Trans. Math. Softw. 7, 3 September 1981, 387-390

[8] Duff, I. S.; Wiberg, T.: Remarks on implementation of $O(n^{1/2}\tau)$ assignment algorithms ,in ACM Trans. Math. Software Volume1 4, pp. 267-287, 1988

[9] Duff, I.S.; Kaya, K.; Uçar, B.: Design, implementation, and analysis of maximum transversal algorithms,ACM Transactions on Mathematical Software (TOMS),38,2,13,2011,ACM

[10] Elmqvist, H.: A Structured Model Language for Large Continuous Systems, Ph.D. Dissertation, Report CODEN: LUTFD2/(TFRT-1015), Dept.

of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1978

[11] Frenkel, J.; Schubert, C.; Kunze, G.; Fritzson, P.; Sjölund, M.; Pop, A.: Towards a Benchmark Suite for Modelica Compilers: Large Models. In: Proceedings of the 8th Modelica Conference 2011, Dresden, Germany, Modelica Association, 20-22 March 2011. https://www.modelica.org/events/modelica2011/Proceedings/pages/papers/07_1_ID_183_a_fv.pdf

[12] Goldberg, A. V.; Tarjan, R. E.: A new approach to the maximum flow problem. Annual ACM Symposium on Theory of Computing, Proceedings of the eighteenth annual ACM symposium on Theory of computing, 136-146

[13] Hopcroft, J. E.; Karp, R. M.: A $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM Journal of Computing, 2(4): 225-231, 1973

[14] Kaya, K.; Langguth, J.; Manne, F.; Uçar, B.: Experiments on Push-Relabel-based Maximum Cardinality Matching Algorithms for Bipartite Graphs, CERFACS Tech. Report TR/PA/11/33, May, 2011

[15] Kaya, K.: http://bmi.osu.edu/ kamer/research.html, last visit 2012-02-05, Matchmaker v0.3

[16] Kunkel, P.; Mehrmann, V.: Index reduction for differential-algebraic equations by minimal extension. Z. angew. Math. Mech., 84: pp. 579-597, 2004

[17] Kunze, G.; Frenkel, J.; Knoll, C.; Schubert C.; Voigt, S.: PyMbs: Ein generisches Software Werkzeug für die Simulation von Mehrkörpersystemen, VDI Mechatronik Tagung, 2011.

[18] Mattsson, S.; Söderlind, G.: Index reduction in differential-Algebraic equations using dummy derivatives, SIAM J. Sci. Comput. 14, 677-692, 1993.

[19] Mattsson, S.E.; Olsson, H; Elmqviste, H. Dynamic Selection of States in Dymola. In: Proceedings of the Modelica Workshop 2000, Lund, Sweden, Modelica Association, 23-24 Oct. 2000.

[20] Mattsson, S.E.; Söderlind, G.: A new technique for solving high-index differential-algebraic equations using dummy derivatives, Computer-Aided Control System Design, 1992. (CACSD), 1992 IEEE Symposium on , pp.218-224, 17-19 Mar 1992

[21] Pantelides C. The Consistent Initialization of Differential-Algebraic Systems.SIAM J. Sci. and Stat. Comput. Volume 9, Issue 2, pp. 213-231, March 1988.

[22] Pop, A.; Fritzson, P.: MetaModelica: A Unified Equation-Based Semantical and Mathematical Modelling Language. In Proceedings of Joint Modular Languages Conference 2006 (JMLC2006) LNCS Springer Verlag. Jesus College, Oxford, England, Sept 13-15, 2006.

[23] Pothen, A; Fan; C.-J.: Computing the block triangular form of a sparse matrix. ACM Trans. Math. Softw. 16, 4 ,December 1990, 303-324

[24] Setubal, J.C.: Sequential and parallel experimental results with bipartite matching algorithms , in Technical Report EC-96-09, Institute of Computing, University of Campinas, Brasil, 1996

[25] Soares, R. de P.; Secchi, A. R.: Direct Initialisation and Solution of High-Index DAESystems. in Proceedings of the European Symbosium on Computer Aided Process Engineering - 15, Barcelona, Spain, 2005

[26] R. Tarjan, Depth-first search and linear graph algorithms, in Conf.Record 1971 IEEE 12th Annu. Symp. Switch. Automata Theory, 1971,pp. 114-121

[27] Unger, J.; Kröner, A.; Marquardt,W.: Structural analysis of differential-algebraic equation systems-theory and applications, Computers & Chemical Engineering, Volume 19, Issue 8, August 1995, Pages 867-882