Bandwidth, Profile and Wavefront Reduction for Static Variable Ordering in Symbolic Model Checking

Erik Kemp
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
e.kemp@student.utwente.nl

ABSTRACT

In symbolic model checking, decision diagrams are used to store all reachable states of a computer program. The size of decision diagrams is highly dependent on the used variable ordering of the underlying structure, a matrix that represents transitions and variables of the system. The model checker LTSmin [16] currently uses a custom implementation to find a good variable ordering, but with this implementation LTSmin fails to produce good orderings for larger matrices. Therefore, this paper explores existing bandwidth, profile and wavefront reduction algorithms and properties, including algorithms used in the field of Structural Analysis in Civil Engineering. We will present the solution to apply existing algorithms to our rectangular matrices in symbolic model checking.

Keywords

Symbolic model checking, Static variable ordering, Bandwidth reduction, Profile reduction, Wavefront reduction, Nodal ordering algorithms, Structural analysis, Rectangular matrices, Graph theory, Bipartite graph, K-total graph

1. INTRODUCTION

A symbolic model checker generates an efficient representation of a modeled system or computer program [8]. It can verify the correctness of these systems, or discover errors. A dependency matrix, obtained by static analysis of the modeled system, is used to represent the modeled system. These matrices are used in the process of creating a Decision Diagram (DD). The problem is that the size of a DD is very dependent on the used variable ordering in the matrix. Initially, these matrices are not ordered in a particular way. In general, the size of the DD explodes. Optimal variable orderings efficiently exploit event locality using the matrix. Empirical results show that this happens when the bandwidth and profile of the matrix are reduced. The bandwidth minimization problem is proved NP-complete in 1976, which results in the existence of a lot of algorithms using heuristic methods. Early bandwidth reduction algorithms are the Reverse Cuthill-Mckee algorithm [10] and the GPS algorithm [14]. From the field of Structural Analysis in Civil Engineering, we used algorithms described in Kaveh's book [19] and the corresponding papers. Therein bandwidth, profile and wavefront reduction and methods for rectangular matrices are explicitly mentioned. Since the matrices LTSmin uses can be rectangular, these methods were found to be very helpful. By using the methods described, we were able to apply existing algorithms on our rectangular matrices in LTSmin.

23rd Twente Student Conference on IT June 22nd, 2015, Enschede, The Netherlands.

1.1 Symbolic model checking

The reasons behind and definitions of model checking are well described in McMillan [27]. "There are several practical reasons for applying formal verification methods to computer systems. The most obvious is the high cost of correcting errors in digital designs." In state of the art infrastructures, software errors can cause enormous problems, making it very worthy to exhaustively check every possible state, to prove no errors can occur. In model checking, the problem commonly known as state space explosion means that for large systems, the amount of reachable states grows exponentially. In the first form of model checking, explicit-state model checking, reachable states were traversed one by one by the model checker. But in [27], McMillan introduced Symbolic Model Checking, where state spaces of the model are not explicitly represented. Programs or systems are expressed as transition systems. The properties of this transition system are then expressed in fixed point logic, such as the μ-calculus. Then, multiple states are compressed in decision diagram nodes. A symbolic model checker can then verify the correctness of these systems, or discover errors, by traversing these nodes opposed to every reachable state one at a time.

1.2 Variable reordering

In LTSmin, a dependency matrix is obtained by static analysis of the modeled system. It represents an overapproximation of the system. This dependency matrix is used for efficient symbolic model checking. In this matrix, the rows represent operations (transitions of the transition system), and the columns represent the global variables of the system, which determine the state of the system. The initial matrix is usually a sparse matrix, sometimes rectangular, with the nonzero elements spread in no particular order. In this context, sparse means that there are relatively few nonzero elements in the matrix. This matrix is used in the process of creating a Decision Diagram (DD). The problem that arises here is that the size of the DD explodes, since most initial variable orderings are not optimal. This is due to not exploiting the event locality efficiently using the dependency matrix.

1.2.1 Creating a dependency matrix: Example

To explain how a dependency matrix is obtained, we provide an example based on the game Sokoban, a transport puzzle game. The problem of solving Sokoban puzzles is

proven to be NP-hard [11]. This makes it an excellent example for our model checker to solve it efficiently. In Sokoban, the playing field is a board of squares, where each square is a floor or a wall. The player can move onto empty squares. Some floor squares contain boxes and some are marked as storage locations. The goal of the player is to move all boxes to a marked storage location. An example of a Sokoban board is provided in Figure 1. We

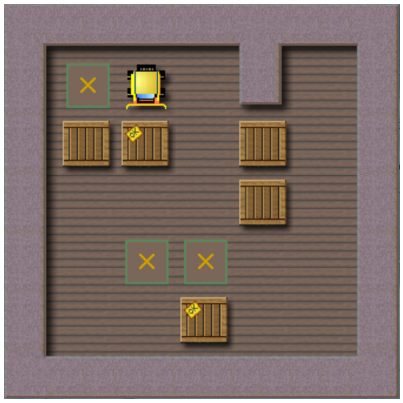


Figure 1: An example of a Sokoban board

can represent a state of the board by using the following ASCII notation, known as the XSB-format:

@	=	player
$	=	box
.	=	goal
_	=	empty square

For the sake of simplicity, consider the following board, with only 3 squares. When we label the rows and the columns and use this notation, we get the following:

	1	2	3
x	.	$	@

This state can be expressed in the following form: (.,$,@). 3 possible states are (.,$,@), ($,@,_),($,_,@).
The transitions between these states are:

$$\begin{array}{lcll} (.,\$,@) & \rightarrow & \text{push(3,2)} & \rightarrow & (\$,@,\_) \\ (\$,@,\_) & \rightarrow & \text{move(2,3)} & \rightarrow & (\$,\_,@) \\ (\$,\_,@) & \rightarrow & \text{move(3,2)} & \rightarrow & (\$,@,\_) \end{array}$$

Then we obtain the dependency matrix by checking which of the squares are 'read' or 'changed'. For example, the transition push(3,2) checks if x_1 is empty, if x_2 is a box and if x_3 is the player. Therefore, for $push(3,2)$, all columns should be filled with a 1.
The dependency matrix of the 3 transitions is:

$$\begin{array}{l} \\ push(3,2) \\ move(2,3) \\ move(3,2) \end{array} \begin{array}{c} \begin{array}{ccc} x_1 & x_2 & x_3 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \end{array}$$

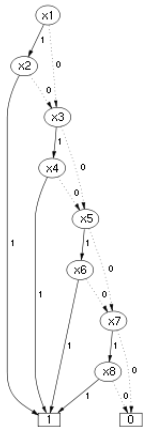
Now this is not a sparse matrix, but it is clear that if we have a larger Sokoban board which has more squares (columns) and more possible transitions (rows), the amount of 1s in a row will still be two or three. As a result, larger matrices become very sparse. This also holds for dependency matrices in general, as atomic operations in programs tend to interact with few global variables.

1.2.2 Event locality

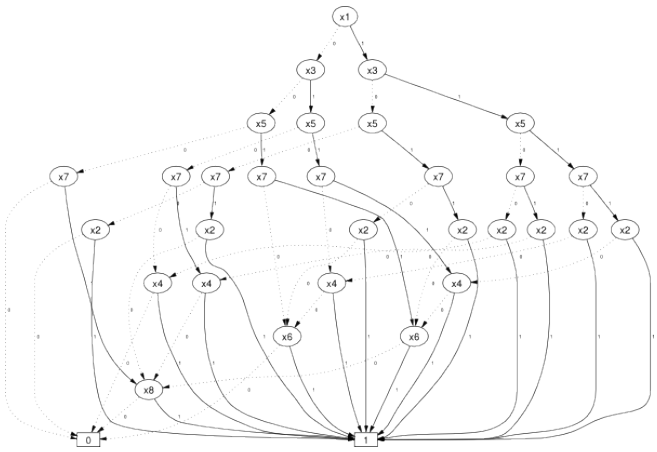
In Blom and Van de Pol [6], event locality is described as the key to applying symbolic techniques to on-the-fly models. "The notion of event locality refers to the fact that even though in a state several events could be enabled, each event separately affects just a small part of the state vector". In the context of our matrix, event locality is efficiently exploited if the nonzero elements in a row are close to each other. From empirical results [6], it is clear that efficient ordering of the variables improves, i.e. reduces, the size of the DD. Conversely, the size of the DD explodes if event locality cannot be exploited.

1.3 Variable reordering (continued)

Therefore, before transforming the matrix into a DD, it is very useful to find an optimal variable ordering for the matrix. In practice, this is an ordering where the nonzero elements of the matrix are near the diagonal. An example of DDs created with respectively good and bad variable orderings can be seen in Figure 2a and Figure 2b. The system enters a correct final state if one of the pairs x_1 and x_2, x_3 and x_4, x_5 and x_6, or x_7 and x_8 are both 1. The boolean formula that represents this simple system is: $(x_1 \wedge \& \; x_2) \vee (x_3 \wedge x_4) \vee (x_5 \wedge x_6) \vee (x_7 \wedge x_8)$



(a) DD with a good variable ordering



(b) DD with a bad variable ordering

Figure 2: An example of 2 DDs for a simple system. (a) has a good and (b) has a bad variable ordering. Images by Dirk Beyer[1], 2005

[1] http://www.sosy-lab.org/~dbeyer/ (March 2015)

1.4 Bandwidth, profile and wavefront

To explain the process of finding good variable orderings, we introduce two definitions: The *bandwidth* and the *profile* of a matrix. Since the matrices that LTSmin use are not limited to square matrices, we present the definitions that also hold for rectangular matrices, as described in Kaveh [19]:

Let **A** be a rectangular matrix with m rows and n columns, whose entries are denoted by b_{ij}. For each row i, the integer part of the real number $i(\frac{n}{m})$ is defined as i_d. For the first and the last row, $i_d = 1$ and $i_d = n$ respectively. Now, the entry of **A** at position (i, i_d) is considered as the i^{th} diagonal entry. For square matrices $m = n$ and $i = i_d$.

The bandwidth of A is then defined as

$$b(A) = m_r + m_l + 1,$$

with the upper bandwidth to the right of the diagonal

$$m_r = max\{k - i_d | a_{ik} \neq 0, k > i_d\}, \quad 1 \leq i \leq n$$

and the lower bandwidth to the left of the diagonal

$$m_l = max\{i_d - k | a_{ik} \neq 0, k < i_d\}. \quad 1 \leq i \leq n$$

For our rectangular non-symmetric matrices, we define the profile of **A** as follows:

$$p(A) = \sum_{i=1}^{n} b_i,$$

with

$$b_i = \max\{k | a_{ik} \neq 0\} - \min\{l | a_{il} \neq 0\}.$$

Here the row width b_i is defined as the number of inclusive entries from the first to the the last non-zero element in the row.

In literature, the term *bandwidth* is far more common than the term *profile*. After the preliminary literature research, our hypothesis was that for our problem, profile reduction is more appropriate than bandwidth reduction. We based this on the fact that we usually work with large matrices, where outliers in the row width will have a small influence on the final DD size, but will greatly increase the bandwidth. In Kaveh [19], we found that most algorithms aim to reduce both profile and bandwidth, so we don't have to choose and can investigate both.

1.4.1 Example

To ensure a clear understanding of bandwidth and profile, we provide an example matrix **B** and calculate its bandwidth and profile:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0^* & 1 & 0 & 1 & 0 \\ 1 & 0^* & 1 & 0 & 0 \\ 0 & 1 & 1^* & 0 & 1 \\ 1 & 0 & 1 & 0 & 0^* \end{bmatrix} \end{array}$$

The diagonal entries, defined for row i as $i(\frac{m}{n})$ are marked with a *. In the lower half, the element at (4,1) results in a lower bandwidth (m_l) of 4, since it has distance 4 to the diagonal entry in (4,5). In the upper half, the element at (1,4) similarly results in an upper bandwidth (m_r) of 3. Then the total bandwidth can be computed:

$$b(B) = 4 + 3 + 1 = 8.$$

The profile is the addition of the row bandwidths, being for rows 1 to 4 respectively $1, 1, 2$ and 1. Then the total profile can be computed:

$$p(B) = 1 + 1 + 2 + 1 = 5.$$

1.4.2 Wavefront

During the research, we found that another property of the matrix is also of interest: The *wavefront*. There are three common metrics defined concerning the wavefront: The maximum wavefront, the average wavefront and the root-mean-square wavefront. Let **A** again be a rectangular matrix with m rows and n columns. The column wavefront f_i for column i is defined as the number of rows before the first nonzero appears in that column. The following common wavefront metrics can be defined for matrix B. The maximum wavefront:

$$f_{max} = \max_{1 \leq i \leq n} (f_i),$$

the average wavefront:

$$f_{avg} = \frac{1}{n} \sum_{i=1}^{n} f_i,$$

and the root-mean-square wavefront:

$$f_{rms} = \left(\frac{\sum_{i=1}^{n} (f_i)^2}{n} \right)^{\frac{1}{2}}$$

1.5 Related work

We have two kinds of related literature, one part connected to profile and bandwidth minimization and the other part related to heuristics in static variable ordering. We propose to use profile and bandwidth reduction algorithms in static variable reordering, but no link between these two has been found in literature.

Related work connected to heuristics and static variable ordering are presented first.

An interesting paper on static variable ordering is [33], that introduces a new family of metrics to be used as a guide for static variable ordering in symbolic methods. A survey of static variable ordering heuristics can be found in [31]. Recent comparative studies of heuristics for static variable ordering can be found in [5] and [9].

Regarding bandwidth and profile reduction, in 1976 Papadimitrou proved the bandwidth minimization problem NP-complete in [28]. In 1978, Rose and Tarjan provided a clear proof that finding a minimum ordering (profile reduction) is NP-complete in [32]. After the problems were proved NP-complete, a lot of heuristics are proposed to tackle these problems efficiently.

Most research has been conducted on bandwidth and profile reduction of symmetric matrices. In this paper multiple nodal ordering algorithms are discussed, which are on itself designed for symmetric matrices. A well-known and widely used algorithm is Cuthill-McKee (CM) [10], introduced in 1969. In 1971, reversing the order that is returned by CM was proposed by George (Reverse Cuthill-McKee, RCM), leading to better results. [26] [12]. Other well-known nodal ordering algorithms are King [25], Sloan [34] and GPS [14]. GPS is improved in [35] and [36] by Wang et al.

An early comparative study of bandwidth and profile reduction algorithms is [15] by Gibbs et al.

Another paper worth looking at is [2], where it is stated

that an algorithm which gives a lower profile does not give the smallest bandwidth. In addition, it states that in solving solving a linear system of equations, for numerical stability it is best to minimize bandwidth, and to minimize execution time it is best to minimize profile. In their results, their proposed algorithm seems to lead to a lower profile than both RCM and Sloan in most cases.

As we hope to be able to incorporate any nodal ordering algorithm in our static variable reordering algorithm, we also provide multiple recent nodal ordering algorithms.

Kaveh has multiple papers nodal ordering algorithms, such as [18], [21], [22], [23] and [24].

A very promising paper for this particular research is [30], on bandwidth reduction of unsymmetric matrices.

Kaveh's book Computational Structural Analysis [19], explicitly considers bandwidth reduction for rectangular matrices. Since this is exactly what we are looking for, it is a great source worth investigating. This book has been most helpful to understand the problem and come up with a solution.

1.6 Problem statement

Bandwidth and profile reduction algorithms are well-studied for square and symmetric matrices [34],[10], [29], [5], [9]. The goal of this research is to find a way to apply existing nodal ordering algorithms on rectangular matrices, and to implement this in the symbolic model checker LTSmin [16].

1.7 Research questions

The main research question is:

> To what extent are bandwidth and profile reduction algorithms suitable for static variable ordering?

We divide this question into these subquestions:

1. Which algorithm is most suitable for our static variable ordering problem?
2. Which parts of this algorithm have to be customized to implement it in LTSmin?
3. Does the new algorithm perform well, in ordering as well as computation time?
4. Does the new algorithm always provide a better bandwidth or profile?
5. What has the most influence on the actual computation time of the reachability analysis, bandwidth or profile reduction?

1.8 Goal

The goal of this research is to find a bandwidth or profile reduction algorithm that we can use for static variable ordering. When implemented in LTSmin, we expect this to lead to a decrease in computation time and memory footprint. If we succeed in achieving this, in the future LTSmin will be able to reorder the variables for large matrices within a reasonable time.

2. BACKGROUND

In this section, we shortly discuss the NP-completeness of bandwidth minimization [28], and profile minimization [32]. We introduce the current implementation in LTSMin. Then we provide brief explanations of well-known bandwidth and profile reduction algorithms, called nodal ordering algorithms, such as Cuthill-McKee [10], King [25], Sloan [34] and GPS [14].

2.1 NP-completeness

It is outside the scope of this paper to explain the full proofs of the NP-completeness for bandwidth minimization and profile minimization here. We do provide a short explanation of the proof for the profile minimization problem.

2.1.1 Bandwidth minimization

Papadimitrious proof of the NP-completeness of bandwidth minimization can be found in [28].

2.1.2 Profile minimization

Based on Rose and Tarjan, [32], this is a short explanation of how the problem of profile minimization is proved NP-complete: The problem can be formulated as follows, for any graph $G = (V, E)$ with size (profile) e': Does G have an ordering which produces a profile of e' edges or less? The proof then consists of two steps. The first step is to demonstrate that there is a non-deterministic polynomial-time algorithm for solving the problem. This algorithm is as follows: Guess an ordering and calculate its profile. If it produces a profile of e' or less, the algorithms answers "yes". The second step consists of reducing a known NP-complete problem to our problem. In the paper, the satisfiability problem is reduced to our problem. Since this was quite a bit harder according to the authors, we omit how this is done.

2.2 Current implementation

In the current version of LTSmin, a custom algorithm is used to sort the columns and rows, in order to get a better variable ordering. This implementation can be found online[2]. We won't explain how it works here, but the computational complexity of this algorithm is too high to make it viable for larger matrices. This is the reason we will provide a more efficient algorithm. The current implementation will be included in the test results, to show how the current algorithm performs.

2.3 Nodal ordering algorithms

Nodal algorithms from the Related Work section are Cuthill-McKee,Sloan and King. We present explanations of those algorithms.

2.3.1 Cuthill-McKee

The Cuthill-McKee algorithm [10] focuses on bandwidth reduction. It is implemented in the Boost library[3], as well as the ViennaCL library[4]. The Reverse Cuthill-McKee algorithm is well described by Ciprian Zăvoianu[5], in the following seven steps:

Step 0: Prepare an empty queue Q and an empty result array R.
Step 1: Select the node in G(n) with the lowest degree (ties are broken arbitrarily) that hasn't previously been inserted in the result array. Let us name it P (for Parent).
Step 2: Add P in the first free position of R.

[2]https://github.com/utwente-fmt/ltsmin/blob/master/src/dm/dm.c#L1079-L1207 (March 2015)
[3]http://www.boost.org (March 2015)
[4]http://viennacl.sourceforge.net (March 2015)
[5]http://ciprian-zavoianu.blogspot.nl/2009/01/project-bandwidth-reduction.html (March 2015)

Step 3: Add to the queue all the nodes adjacent with P in the increasing order of their degree.
Step 4.1: Extract the first node from the queue and examine it. Let us name it C (for Child).
Step 4.2: If C hasn't previously been inserted in R, add it in the first free position and add to Q all the neighbors of C that are not in R in the increasing order of their degree.
Step 5: If Q is not empty repeat from Step 4.1.
Step 6: If there are unexplored nodes (the graph is not connected) repeat from Step 1.
Step 7: Reverse the order of the elements in R. Element R[i] is swapped with element R[n+1-i].

This last step is the reason this algorithm is called 'Reverse' Cuthill-Mckee. It was not included in the original Cuthill-Mckee algorithm, but was proposed by George [12], and leads to a better profile while the bandwidth is never increased [26].

The time complexity for the algorithm is the following, according to the boost online boost library[6]:

$$O(m\ log(m)|E|) \text{ where } m = max\{degree(v)|v\ in\ V\}$$

2.3.2 Sloan

Sloan's algorithm [34] focuses on profile and wavefront reduction. It is implemented in the Boost library[6] It is well described in Reid [29], of which we present a short summary. Reid states that the algorithm consists of two phases. In the first phase, a start and end node are chosen. In the second phase, the start node from the first phase is numbered first. Then, at each stage of numbering a list is formed containing neighbours of nodes which have been numbered and their neighbours. The next node is chosen from the list by a priority function, where a node has a higher priority if it causes only a small increase, or none at all, to the current wavefront size and if it is at a large distance from the end node.

The time complexity for the algorithm is the following, according to the boost online boost library[6]:

$$O(log(m)|E|) \text{ where } m = max\{degree(v)|v\ in\ V\}$$

This is the lowest complexity of the three algorithms mentioned here.

2.3.3 King

We have not found a detailed description of how King's algorithm [25] works. The summary from boost is that the King algorithm focuses on reducing bandwidth, and that this is achieved by locally minimizing the row bandwidths.

The time complexity for the algorithm is the following, according to the boost online boost library[3]:

$$O(m^2log(m)|E|) \text{ where } m = max\{degree(v)|v\ in\ V\}$$

3. RESEARCH METHODOLOGY

3.1 Approach

The most valuable resource for this research proved to be Kaveh's book [19]. By carefully reading it, we obtained the idea that we could indeed use any nodal ordering algorithm if we could find a suitable representation for the rectangular matrix, and transform the returned permutation into row and column permutations for the rectangular matrix. How we have achieved this will be described in detail.

[6]http://www.boost.org (March 2015)

3.1.1 Representing the rectangular matrix

In Kaveh [19], p.177 - p.181, the idea of a K-total graph is suggested to order two types of matrices, a cutset basis incidence matrices and a cycle basis incidence matrix. Since our matrices were not applicable for the graph representations used, we used another representation, also found in Kaveh. This is the bipartite graph. Later, we also constructed the total graph of this bipartite graph to compare the results for reordering.

3.1.2 Constructing the bipartite graph

If we represent our rectangular matrix as a bipartite graph, we can apply any nodal ordering algorithm on the bipartite graph. The matrix is transformed into a bipartite graph through the following function. The input for the function is a rectangular matrix A, with m rows and n columns. U and V are the distinct Vertex sets of the bipartite graph, and E the set of edges. G is undirected.

$$createBipartiteGraph : A_{m*n} \rightarrow G = (U, V, E),$$

with

$$U = 1, ..., m,$$
$$V = m + 1, ..., m + n,$$
$$E = \{\{a, b + m\}|A_{ab} = 1\}.$$

3.1.3 Constructing the total graph

As hinted by Kaveh, we constructed the total graph as defined in [3],[4]. We compare the results of reordering the total graph to reordering the bipartite graph. From MathWorld[7]:
The total graph $T(G)$ of a graph G has a vertex for each edge and vertex of G and an edge in $T(G)$ for every edge-edge, vertex-edge, and vertex-vertex adjacency in G.

3.1.4 Processing the computed permutation

Now we can reorder the bipartite and total graphs of our matrix, by using any nodal ordering algorithm available. The algorithms will return a permutation, as a list of node numbers. Since we want to permute the rows and the columns of the matrix separately, we have to define a function to split the permutation. We define the output of the algorithms as the permutation vector $\mathbf{x}$, and again a rectangular matrix A, with m rows and n columns.

$$splitPermutation : \mathbf{x} < \mathbf{N} > \rightarrow (\mathbf{y} < \mathbf{N} >, \mathbf{z} < \mathbf{N} >),$$

where

$\mathbf{x}$ is the original permutation vector,

$\mathbf{y}$ is the row permutation vector,

$\mathbf{z}$ is the column permutation vector.

$$splitPermutation(\mathbf{x}) = (\mathbf{y}, \mathbf{z})$$

where the following function is applied sequentially to the elements of $\mathbf{x}$, starting with the first element in $\mathbf{x}$:

$$\forall x \in \mathbf{x} : \begin{cases} x \in \mathbf{y} \text{ if } x \leq m \\ x - m \in \mathbf{z} \text{ if } x > m \end{cases}$$

3.1.5 Obtaining test results

We first obtained models from an online source. For this research, we chose Petri Nets from the Model Checking Contest website[8]. We used these models as an input for

[7]Weisstein, Eric W. "Total Graph." From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/TotalGraph.html (March 2015)
[8]http://mcc.lip6.fr/models.php (March 2015)

the model checker LTSmin[16]. For the tests, we used nine different reordering parameters. Three of them are existing parameters, No variable ordering, Row sort and Column Swap - Row Sort (the currently used algorithm in LTSmin). The other six are the Reverse Cuthill-McKee, Sloan and King algorithms on both the bipartite and total graph created from the matrix. For the three reordering algorithms, we used the implementation in the boost library[6]. In total, 843 test runs had complete and useful output.

3.1.6 Measuring performance

One important consideration in this research is how to measure the performance of the algorithms. During the test, we will measure bandwidth, profile, average wavefront, maximum wavefront and root-mean-square wavefront, for the rows, the columns and combined. We will measure the following for each test run: regrouping time, reachability time, total computation time and peak nodes. These peak nodes are the maximum number of nodes representing a model during the reachability analysis. For measurement, we will use *memtime*, a tool that can be found online[9]. The maximum number of peak nodes is set to 20 million, for visibility. The maximum number of peak nodes reached during the tests is 18108480. These are worth measuring, because if the amount of nodes exceeds the available memory at some time in the program, the program is terminated. Due to time constraints, we set the maximum computation time for each test run to 3600 seconds (1 hour). If the regrouping, or the total computation time exceeded this this limit, the run was cancelled. These tests can be found in the total time plots, where the time is set to infinity. Due to functioning of the scripts to visualize test results, we omitted these tests in the regrouping time plots.

3.2 Results

First, we show some of the most promising results, achieved with applying Sloan's algorithm to the bipartite graph. The measured times are plotted in a scatter plot against the algorithm that is currently used in LTSmin (column swap, row sort). Note that the scatter plots have a logarithmic scale. The axes for time range from 0.1 seconds to an hour, the axes for peak nodes from 1 to 20 million nodes. We have to mention that in all plots, there exist some dots on the infinity value, for which no test has actually been run, due to a crash of the resource manager.

3.2.1 Most promising results: Sloan

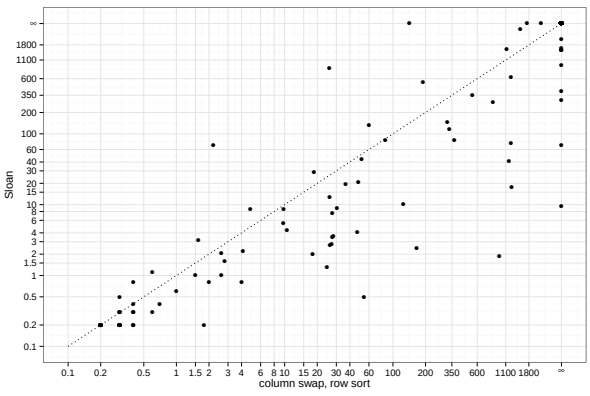


Figure 3: Total time: Column swap, row sort vs. Sloan

[9]http://fmt.cs.utwente.nl/tools/scm/memtime.git (March 2015)

The total time in this context is the total computation time measured by memtime from the start of the program to the end of the reachability analysis of LTSmin. In Figure 3, all dots beneath the diagonal indicate that the total computation time is lower for the Sloan algorithm than for the column swap, row sort algorithm currently used in LTSmin. We have checked all dots on infinity for Sloan manually, and we conclude that they are all because of the resource manager bug. By analyzing all produced scatter plots, we conclude that the Sloan algorithm on the bipartite graph is in general the best performing algorithm. From Figure 3 we derive that Sloan outperforms column swap, row sort for the major part of the tests. We will continue to investigate the other scatter plots for these two algorithms.

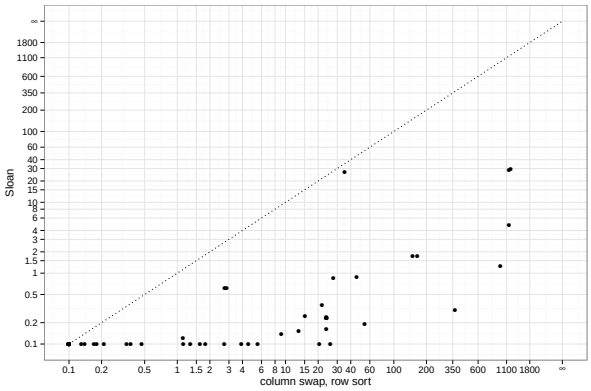


Figure 4: Regrouping time: Column swap, row sort vs. Sloan

In Figure 4, the power of the Sloan algorithm is clearly visible. It is faster than column swap, row sort for every single test. Since the growing regrouping times for large matrices was the main problem of column swap,row sort, this is exactly the result we hoped for. All regrouping times for column swap, row sort that exceed the 1 hour time-limit are even omitted in this plot.

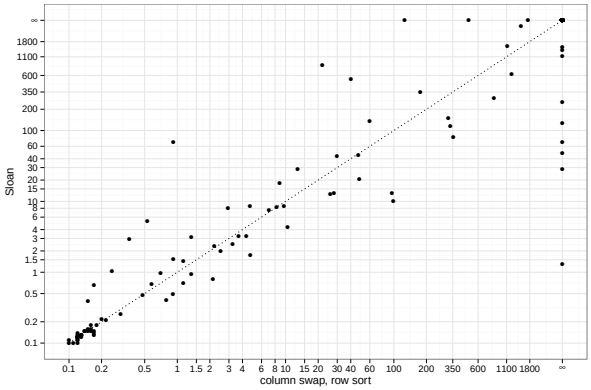


Figure 5: Reachability time: Column swap, row sort vs. Sloan

In Figure 5 we see that in terms of reachability time, there is no clear winner between the two algorithms. This matches our expectations from manual tests during the research, which showed that the ordering that column swap, row sort produces is actually pretty good.

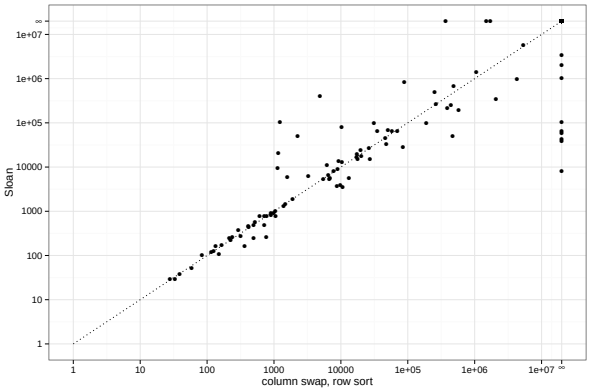


Figure 6: Peak nodes: Column swap, row sort vs. Sloan

If a test did not terminate within the time limit, the amount of peak nodes is set to infinity. The amount of peak nodes is similar for the two algoriths, as can be seen in Figure 6. Since this did not heavily influence the computation time, we omit further discussion of the peak nodes.

3.2.2 Results for the current algorithm

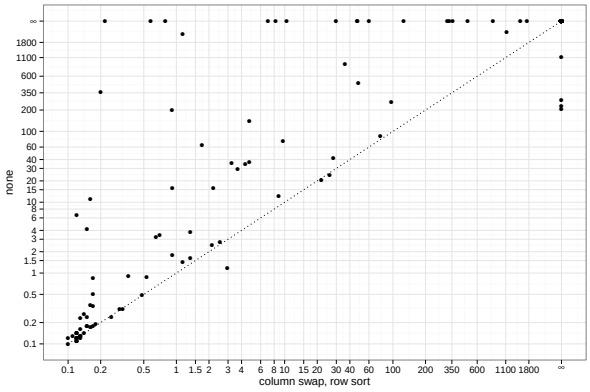


Figure 7: Reachability time: Column swap, row sort vs. no variable reordering

If we look at the reachability time scatter plot of column swap, row sort vs no variable reordering at all in Figure 7, we see that reordering the matrix indeed speeds up the reachability analysis. Some models initially already have a reasonably good variable ordering, but especially for larger matrices reordering pays off.

3.2.3 Results for other algorithms

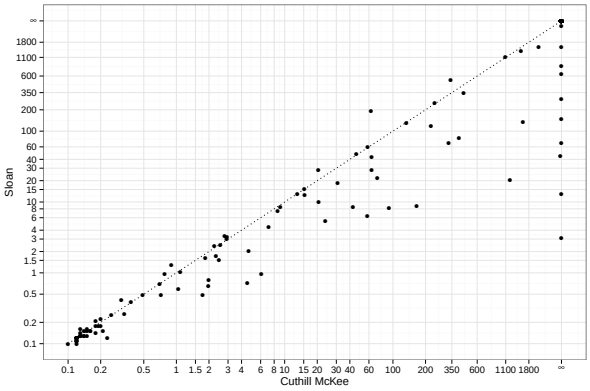


Figure 8: Reachability time: Cuthill-McKee vs. Sloan

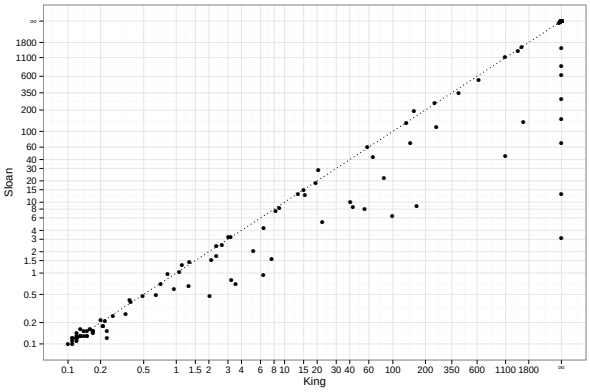


Figure 9: Reachability time: King vs. Sloan

The regrouping time for all new algorithms are almost the same, both on bipartite and on total graphs. The scatter plot shown for Sloan vs. column swap, row sort in Figure 4 is almost identical to all other scatter plots that plot a new algorithm against the column swap, row sort algorithm. The reachability time for both the Reverse Cuthill-McKee and the King algorithms led to worse results than Sloan's algorithm. Two relevant scatter plots are included in Figure 8 and Figure 9.

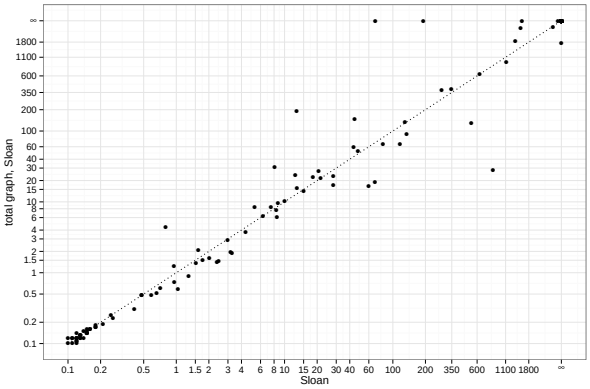


Figure 10: Reachability time: Sloan on total graph vs. Sloan on bipartite graph

Regarding the total graphs, Sloan also outperforms both Cuthill-McKee and King when the latter two are applied on a total graph. The scatter plot of Sloan on the bipartite graph vs. Sloan on the total graph is provided in Figure 10.

Kaveh [19] states that reordering the total graph usually returns a better ordering than reordering the bipartite graph, at the cost of execution time. We don't see a significantly better reachability time when we use the total graph, though the reachability time is a bit lower for Sloan on the total graph for a majority of the tests. We note that maybe Kavehs argument was only when regarding cut sets or cocycle basis, which we did not.

3.2.4 Detailed results

We give detailed results for some points from the column swap, row sort vs. Sloan scatter plot. From Figure 3, we show the details of the point in the right lower corner, where the total time for column swap, row sort is 962.92 seconds, and the total time for Sloan is 1.9 seconds. This is the model for 100 dining philosophers.

Table 1: Test results for the 100 dining Philosophers model

	Column swap, row sort	Sloan
Bandwidth	279	17
Profile	5183	3777
Max. wavefront	500	500
Avg. wavefront	254	255
RMS wavefront	293	294
Peak nodes	9806	3968
DD nodes	9806	3968
Regrouping time	961.99	1.25
Reachability time	0.79	0.41

The matrix is a square matrix with 500 rows and 500 columns. We see that the column swap, row sort algorithm has a very high regrouping time. It produces a slightly higher profile than Sloan, but a very high bandwidth. Since the reachability time for both algorithms seems proportional to the profile rather than to the bandwidth, this supports our hypothesis that for our problem, bandwidth alone doesn't have a big influence on the reachability time. Of course, if bandwidth is reduced a lot, the upper bound of the profile also decreases.

From our results, we found that for certain models, profile and wavefront indeed seem proportional to the reachability time. We present graphs for the profile, root-mean-square (RMS) wavefront and the bandwidth plotted versus the reachability time, for the combination of all dining philosophers models. In the appropriate cases, we added a trend line.

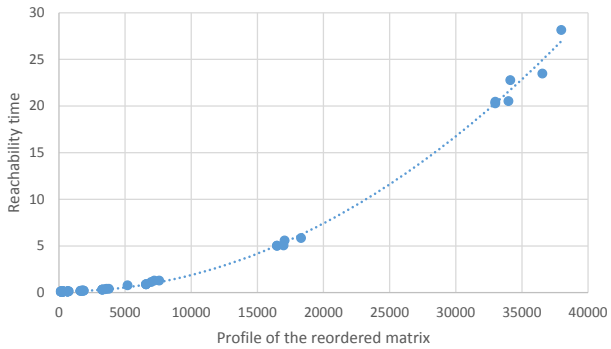


Figure 11: All dining philosopher models combined, profile vs. reachability time

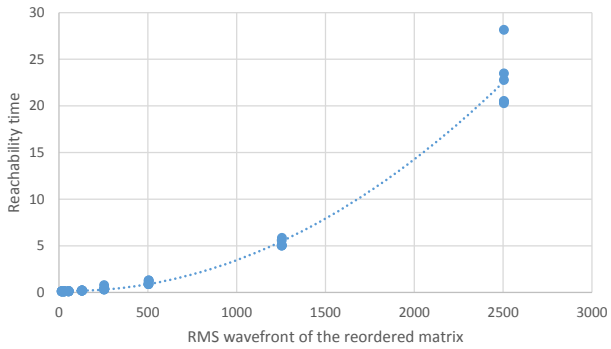


Figure 12: All dining philosopher models combined, root-mean-square wavefront vs. reachability time

It makes sense that the points are in different parts of the graph, as the Philosopher problems are of different size, with great gaps between the different models. The right upper dots are all from the 1000-philosophers problem, the middle dots are from the 500-philosophers problem, and then to the lower left corner are the 200-, 100-, 50-, 20-, 10-, 5-philosophers problem.

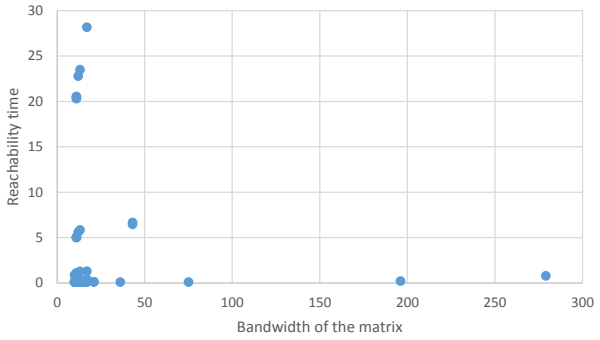


Figure 13: All dining philosopher models combined, bandwidth vs. reachability time

From this figure, it is clear that a lower bandwidth does not necessary results in a lower reachability time.

There are other scalable problems that have similar-looking scatter plots for profile and wavefront, but not all of them. Here we present some scatter plots for profile against reachability time, where we see no proportionality.

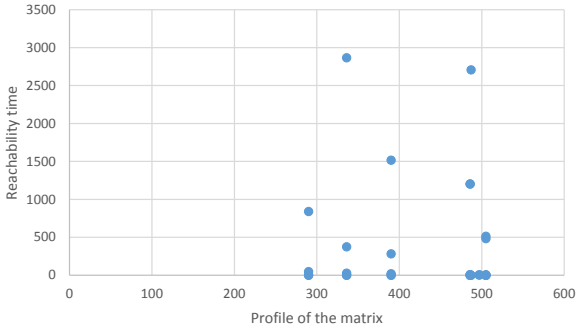


Figure 14: All angiogenesis models combined, profile vs. reachability time

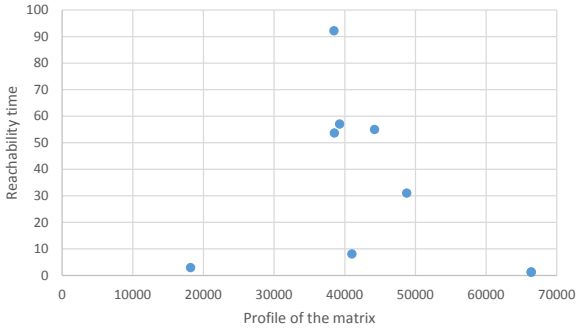


Figure 15: ARM Cache Coherence, profile vs. reachability time

In Figure 14 and Figure 15, we see that there is no apparent relation between the profile of the matrix and the reachability time, because a lower profile does not always lead to a lower reachability time. We manually checked the bandwidths and wavefronts for these problems too,

but they do not reveal a relation either. In addition, no direct proportionality is found between profile / wavefront and reachability time in general.

3.3 Conclusion

The most important result from this research is the notion that we can transform rectangular matrices into bipartite graphs (and optionally into total graphs), then apply the permutation returned by a nodal ordering algorithm on the matrix by splitting it into a row and column permutation. Furthermore, our results show that for LTSmin, it is undeniably a good choice to use bandwidth or profile reduction algorithms in the future. We found that Sloan had the best results of the three new algorithms that we tested. Both the application on the on the bipartite and total graph of the matrix led to good results.

What we do not know, is what matrix metric has the most influence on the reachability time. We found that only for a few scalable models, profile and the RMS wavefront seem to be proportional to the reachability time, but this is not definitive and does not hold for all models.

All in all, the results from this research are directly applicable and very promising, and further study in the subject will probably lead to even better results. A lot of recent literature on nodal ordering algorithms is worth investigating in future work.

4. FUTURE WORK

There are three main challenges for future work based on this research. The first one is to find out what metric has the most influence on the reachability time of LTSmin. Our hypothesis is now that any nodal ordering algorithm can be applied to the bipartite and total graph. Then the second challenge is to find an optimal nodal ordering algorithm, in computation time as well as in the optimized values. Therefore, we include some recent nodal ordering algorithms. The last challenge is to implement the found optimal nodal ordering algorithm efficiently. We have found one paper that proposes the first parallel implementations of the older nodal ordering algorithms.

4.1 Finding the optimal metrics

It has to be proven which metric has most influence on reachability time: bandwidth, profile, wavefront, or something else. A metric that might be considered is the distance to the diagonal, from both the horizontal as the vertical axis. So this is something like the summation of all column wavefronts with all row wavefronts, the distance from the axis until the first element in the column or row. Additionally, the performance differences between applying a nodal ordering algorithm on a bipartite graph or a total graph have to be investigated.

Papers mentioned in the Related work section that could be useful for investigating metrics are Siminiceanu and Ciardo [33] from 2006, that introduces a new family of metrics to be used as a guide for static variable ordering in symbolic methods. A survey of static variable ordering heuristics can be found in Rice and Kulhari [31], from 2008. Recent comparative studies of heuristics for static variable ordering can be found in Bernardes and de Oliveira [5] and Chagas and de Oliveira [9], both from 2015.

4.2 Nodal ordering algorithms

Now that we have incorporated implementations of RCM, Sloan and GPS in LTSmin, newer algorithms from the Related Work section can hopefully also be implemented. Here we present a list of relatively recent algorithms.

An algorithm for which multiple (open source) implementations can be found online is the Minimum Degree Ordering Algorithm. A historical overview of this algorithm can be found in [13]. An implementation is AMD ([1]), which is available in MATLAB and in the SuiteSparse online library[10].

Reid and Scott [30], from 2006, should be investigated because they focus on other ways to apply nodal ordering algorithms to unsymmetric matrices.

Zhou and Ren [37], from 2009, proposing a new clustering algorithm to reduce the profile of a sparse asymmetric 0-1 matrix.

Wang et al. [36], from 2009, with an improved nodal ordering algorithm based on GPS.

Boutora et al. [7], proposing a new fast method for profile and wavefront reduction, from 2011.

Three recent algorithms by Kaveh are:
Kaveh and Sharafi [23] from 2009, nodal ordering for profile reduction is performed using ant colony optimization. The results show that Sloan's method can be improved using the new parameters. The second algorithm is Kaveh and Sharafi [24] from 2011, introduces a new nodal ordering algorithm, which uses a meta-heuristic optimization method known as charged system search. Results show that it is applicable to bandwidth and profile optimization. It performs better than Sloan and several other algorithms, in the optimized value as well as the computation time. The third algorithm can be found in Kaveh and Bijari [20], from 2015. It is the most recent one, and it uses two recently developed meta-heuristic optimization methods for optimum nodal ordering to minimize bandwidth of sparse matrices. These methods are Colliding Bodies Optimization and Enhanced Colliding Bodies Optimization. The author states in his concluding remarks that the obtained values by these algorithms are the best results so far.

4.3 Parallelization

For possible improved performance, the nodal ordering algorithm could be parallelized. Parallelization falls outside the scope of this research, but a paper on parallelization is Karanthasis et al. [17]. In this paper the first parallel implementations of the Reverse Cuthill-McKee and Sloan algorithms are proposed, which perform significantly better than their sequential implementations in the HSL library[11]. In the future, the optimal nodal ordering algorithm could possibly also be parallelized.

5. REFERENCES

[1] P. R. Amestoy, T. A. Davis, and I. S. Duff. Algorithm 837: Amd, an approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 30(3):381–388, 2004.

[2] A. Baldi and A. de Paulis. On the profile reduction of sparse symmetric matrices. *Ratio Mathematica*, 4:1–11, 1992.

[3] M. Behzad. *Graphs and their chromatic numbers*. PhD thesis, 1965.

[4] M. Behzad. A characterization of total graphs. *Proceedings of the American Mathematical Society*, 26(3):383–389, 1970.

[10] http://faculty.cse.tamu.edu/davis/suitesparse.html (March 2015)

[11] http://www.hsl.rl.ac.uk/

[5] J. A. B. Bernardes and S. L. G. de Oliveira. A systematic review of heuristics for profile reduction of symmetric matrices. *Procedia Computer Science*, 51:221–230, 2015.

[6] S. Blom and J. van de Pol. Symbolic reachability for process algebras with recursive data types. *International Colloquium on Theoretical Aspects of Computing*, 5:81–95, 2008.

[7] Y. Boutora, R. Ibtiouen, S. Mezani, N. Takorabet, and A. Rezzoug. A new fast method of profile and wavefront reduction for cylindrical structures in finite elements method analysis. *Progress In Electromagnetics Research B*, 27:349–363, 2011.

[8] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking: 1020 states and beyond. In *Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on e*, pages 428–439, Jun 1990.

[9] G. O. Chagas and S. L. G. de Oliveira. Metaheuristic-based heuristics for symmetric-matrix bandwidth reduction: A systematic review. *Procedia Computer Science*, 51:211–220, 2015.

[10] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th National Conference*, ACM '69, pages 157–172, New York, NY, USA, 1969. ACM.

[11] D. Dor and U. Zwick. Sokoban and other motion planning problems. *Computational Geometry*, 13(4):215–228, 1999.

[12] A. George and J. W. Liu. *Computer solution of Large Sparse Positive Definite*. Prentice Hall Professional Technical Reference, 1981.

[13] A. George and J. W. H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31(1):pp. 1–19, 1989.

[14] N. E. Gibbs, J. Poole, William G., and P. K. Stockmeyer. An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM Journal on Numerical Analysis*, 13(2):pp. 236–250, 1976.

[15] N. E. Gibbs, W. G. Poole, Jr., and P. K. Stockmeyer. A comparison of several bandwidth and profile reduction algorithms. *ACM Trans. Math. Softw.*, 2(4):322–330, Dec. 1976.

[16] G. Kant, A. Laarman, J. Meijer, J. van de Pol, S. Blom, and T. van Dijk. Ltsmin: High-performance language-independent model checking. *Tools and Algorithms for the Construction and Analysis of Systems*, 90:1, 2015.

[17] K. I. Karantasis, A. Lenharth, D. Nguyen, M. J. Garzarán, and K. Pingali. Parallelization of reordering algorithms for bandwidth and wavefront reduction. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 921–932, Piscataway, NJ, USA, 2014. IEEE Press.

[18] A. Kaveh. Advances in computational mechanics via graph theory. *Asian Journal of Civil Engineering*, 7(4):393–410, 2006.

[19] A. Kaveh. *Computational Structural Analysis and Finite Element Methods*. SpringerLink : Bücher. Springer, 2013.

[20] A. Kaveh and S. Bijari. Bandwidth optimization using cbo and ecbo. *Asian journal of Civil Engineering*, 16(4):535–545, 2015.

[21] A. Kaveh and A. Mokhtar-Zadeh. Bandwidth optimization for rectangular matrices. *Computers & Structures*, 73(1-5):497 – 509, 1999.

[22] A. Kaveh and P. Sharafi. Optimal priority functions for profile reduction using ant colony optimization. *Finite Elem. Anal. Des.*, 44(3):131–138, Jan. 2008.

[23] A. Kaveh and P. Sharafi. Nodal ordering for bandwidth reduction using ant system algorithm. *Engineering Computations*, 26(3):313–323, 2009.

[24] A. Kaveh and P. Sharafi. Ordering for bandwidth and profile minimization problems via charged system search algorithm. 2011.

[25] I. P. King. An automatic reordering scheme for simultaneous equations derived from network systems. *International Journal for Numerical Methods in Engineering*, 2(4):523–533, 1970.

[26] W.-H. Liu and A. H. Sherman. Comparative analysis of the cuthill-mckee and the reverse cuthill-mckee ordering algorithms for sparse matrices. *SIAM Journal on Numerical Analysis*, 13(2):pp. 198–213, 1976.

[27] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1992. UMI Order No. GAX92-24209.

[28] C. Papadimitriou. The np-completeness of the bandwidth minimization problem. *Computing*, 16(3):263–270, 1976.

[29] J. K. Reid and J. A. Scott. Ordering symmetric sparse matrices for small profile and wavefront. *International Journal for Numerical Methods in Engineering*, 45(12):1737–1755, 1999.

[30] J. K. Reid and J. A. Scott. Reducing the total bandwidth of a sparse unsymmetric matrix. *SIAM J. Matrix Anal. Appl.*, 28(3):805–821, Aug. 2006.

[31] M. Rice and S. Kulhari. A survey of static variable ordering heuristics for efficient bdd/mdd construction. *University of California, Tech. Rep*, 2008.

[32] D. J. Rose and R. E. Tarjan. Algorithmic aspects of vertex elimination on directed graphs. *SIAM Journal on Applied Mathematics*, 34(1):pp. 176–197, 1978.

[33] R. I. Siminiceanu and G. Ciardo. New metrics for static variable ordering in decision diagrams. In *Proceedings of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'06, pages 90–104, Berlin, Heidelberg, 2006. Springer-Verlag.

[34] S. W. Sloan. An algorithm for profile and wavefront reduction of sparse matrices. *International Journal for Numerical Methods in Engineering*, 23(2):239–251, 1986.

[35] Q. Wang, Y.-C. Guo, and X.-W. Shi. A generalized gps algorithm for reducing the bandwidth and profile of a sparse matrix. *Progress In Electromagnetics Research*, 90:121–136, 2009.

[36] Q. Wang and X.-W. Shi. An improved algorithm for matrix bandwidth and profile reduction in finite element analysis. *Progress In Electromagnetics Research Letters*, 9:29–38, 2009.

[37] J. Zhou and Y. Ren. An algorithm for reducing the profile of a sparse asymmetric 0-1 matrix. In *2009 WRI World Congress on Software Engineering*, volume 2, pages 234–238, 2009.