

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

Jeroen Meijer and Jaco van de Pol

Formal Methods and Tools, University of Twente, The Netherlands
{j.j.g.meijer, j.c.vandepol}@utwente.nl

Abstract. We demonstrate the applicability of bandwidth and wavefront reduction algorithms to static variable ordering. In symbolic model checking event locality plays a major role in time and memory usage. For example, in Petri nets event locality can be captured by dependency matrices, where nonzero entries indicate whether a transition modifies a place. The quality of event locality has been expressed as a metric called (weighted) event span. The bandwidth of a matrix is a metric indicating the distance of nonzero elements to the diagonal. Wavefront is a metric indicating the degree of nonzeros on one end of the diagonal of the matrix. Bandwidth and wavefront are well studied metrics used in sparse matrix solvers.

In this work we prove that span is limited by twice the bandwidth of a matrix. This observation makes bandwidth reduction algorithms useful for obtaining good variable orders. One major issue we address is that the reduction algorithms can only be applied on symmetric matrices, while the dependency matrices are asymmetric. We show that the Sloan algorithm executed on the total graph of the adjacency graph gives the best variable orders. Practically, we demonstrate that our work allows to call standard sparse matrix operations in Boost and ViennaCL, computing very good static variable orders in milliseconds. Future work is promising, because a whole new spectrum of more off-the-shelf algorithms, including metaheuristic ones, become available for variable ordering.

Keywords: bandwidth, profile, wavefront, event span, symbolic model checking, sparse matrix, event locality, decision diagram, Petri net

1 Introduction

Model checking is an approach for finding errors in computer programs by computing reachable states of a program and evaluating formulas over these set of states. Some type of computer programs allow efficient storage of its set of reachable states by means of decision diagrams, this technique is known as symbolic model checking [7]. Storing sets of states symbolically entails storing sets of integer vectors as binary formulas in Binary Decision Diagrams (BDDs) [5] or more recent as Multi-value Decision Diagrams (MDDs) [16]. One major issue with this approach is the ordering of variables in decision diagrams (DDs) representing the formula. Improving variable ordering is known to be NP-complete [4],

thus heuristic [22] and metaheuristic [27] algorithms have been developed to improve static variable ordering. The static variable ordering approach orders variables before reachability analysis, while dynamic variable ordering happens during the computation of reachable states.

Static variable ordering typically exploits the notion of event locality. Events, such as program statements or transitions in Petri nets are often local, i.e. they modify or read only a few variables or places and ordering these local variables near each other tends to significantly reduce the memory footprint of the DDs. A good metric for event locality is called the Weighted Event Span (WES), by Siminiceanu et al. [27]. The WES metric is used to measure the total normalized distance between the minimum variable and maximum variable of all events. Furthermore the metric includes a moment which signifies the importance of involved events happening in the bottom of the DD. This is important, because with saturation operations in DDs are cheaper in the bottom, rather than the top. Every operation in the top of the DD recursively propagates down the DD.

The degree of locality of events can be visualized using matrices. Such an approach is taken in [24], where a dependency matrix has rows as transitions and columns as variables. A nonzero entry indicates that a transition depends on a variable, e.g. a transition can either read or write to a variable [24]. These dependency matrices tend to be sparse, hinting that traditional sparse matrix algorithms can be applied to these matrices.

A subcategory of sparse matrix algorithms are bandwidth and wavefront reduction algorithms. One key example of a bandwidth reduction algorithm is by Cuthill and McKee developed in 1969 [11]. The goal of these algorithms is very similar to WES reduction algorithms. The bandwidth measures the distance of nonzeros from the diagonal of the matrix, while wavefront measures the degree of nonzeros on one end of the diagonal of the matrix. Bandwidth is related to event span because reducing bandwidth must also reduce event span, because of the triangle inequality, which states that event span is always smaller than twice the bandwidth. Since wavefront reduction moves nonzeros to the right of the matrix, wavefront reduces the moment in the WES metric.

Another popular algorithm in numerical analysis is Sloan's [28] algorithm, which optimizes total bandwidth (also called profile) and wavefront. The graph algorithm has a very low time complexity $\mathcal{O}(\hat{D} \cdot \log \hat{D} \cdot |V|)$, where \hat{D} is the maximum degree, and V the set of vertices. This results in runtimes of mere seconds when applied to matrices with a million rows and columns – or transitions and variables. Conveniently, Sloan's algorithm is freely available in Boost's graph library¹. Every model checker written in C/C++ or Python can be linked to Boost without much effort.

While bandwidth and wavefront reduction algorithms have proven themselves during the past decades, they only work on symmetric matrices. A dependency matrix is asymmetric because clearly, transitions (rows) and variables (columns) are different objects and there exists no natural total order on the union of both. Reid et al. [25] discuss several methods of symmetrizing asymmetric matrices.

¹ http://www.boost.org/doc/libs/1_59_0/libs/graph/doc/sloan_ordering.htm

With visualizations and experimental data we show that indeed, simply assigning some total order, that preserves the partial order on transitions and variables works well for symbolic model checking.

We extensively benchmark the Cuthill McKee, Gibbs Poole Stockmeyer, King and Sloan nodal ordering algorithms implemented in Boost and ViennaCL² [26]. The benchmark consists of more than 300 Petri net models from the 2015 model checking contest [21]. Both libraries are linked to the LTSMIN model checker. The key feature of LTSMIN is the PINS architecture, extensively discussed in [17]. LTSMIN allows language independent model checking by exposing the *next-state* function in PINS, which language front-ends, such as DiViNE, PROMELA or UPPAAL implement.

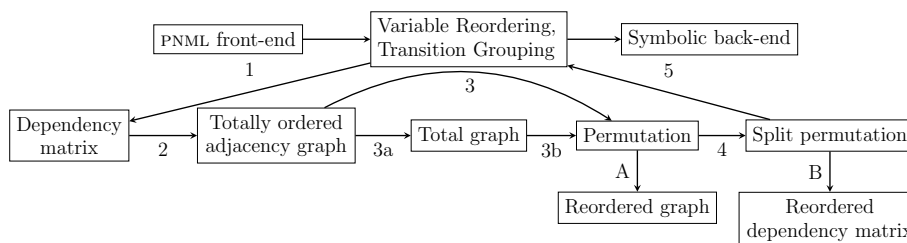


Fig. 1: PINS' extension

The proposed extension to PINS is shown in Figure 1. To perform variable reordering with nodal ordering algorithms the following steps are added to the *Variable Reordering, Transition Grouping* layer in PINS.

1. Existing feature in PINS. Dependency matrices are stored using a dense matrix format. Associated with the (asymmetric) dependency matrix is a partially ordered adjacency graph.
 2. Assign a total order to the adjacency graph of the dependency matrix using Boost or ViennaCL. Assigning a total order makes the dependency matrix symmetric.
 3. Obtain a permutation using a nodal ordering algorithm.
 - (a) Optionally create a total graph first.
 - (b) Obtain a permutation using a nodal ordering algorithm.
 4. Create a split permutation for the dependency matrix. I.e. one permutation for the rows and one for the columns.
 5. Make the symbolic back-end use the split permutation for transitions and variables.
- A. Optionally create the reordered graph to print metrics.
 B. Optionally create the reordered dependency matrix to print metrics.

2 Preliminaries

The bandwidth and wavefront reduction algorithms are designed for undirected ordered graphs. We therefore introduce the notion of unordered pairs. We also

² <http://viennacl.sourceforge.net>

define orders on sets, which allow us to define ordered graphs. The biadjacency matrix, which is a matrix representation of a bipartite graph, is important for understanding the dependency matrix. The two parts of a bipartite graph are the set of transitions and the set of variables. Additionally we introduce the notion of a total graph. Kaveh [19] suggests to create the total graph of the adjacency graph, because some algorithms produce even better permutations on the total graph at the expense of additional computation time.

2.1 Sets

The set \mathbb{N} denotes the set of natural numbers excluding 0, and \mathbb{N}^0 including 0. The set $\mathbb{N}_{\leq m}$ denotes the natural numbers less than or equal to m . Given a set S , an unordered pair of elements from S is denoted $\{a, b\} \in \binom{S}{2} = \{\{c, d\} \mid c \in S \wedge d \in S \wedge c \neq d\}$. We write $S^2 = S \times S$, for the set of ordered pairs.

Definition 1 (order). *Given a set V , an order on V is a (reflexive, antisymmetric and transitive) relation $O \subseteq V^2$. We write $a \leq b = (a, b) \in O$. If $\forall a, b \in V: (a, b) \in O \vee (b, a) \in O$ then O is total, otherwise O is partial. We define function $p_O: V \rightarrow \mathbb{N}_{\leq |V|}$, such that $v \mapsto |\{u \mid (u, v) \in O\}|$ gives the position of an element in an order. If O is clear from the context, we write $p(v)$ as shorthand for $p_O(v)$. Given a set $U \subseteq V$ we write $O_U = O \cap U^2$. Given a permutation $\pi: V \rightarrow V$, a permuted order O^π on V under π is $a \leq_\pi b = (a, b) \in O^\pi \iff \pi(a) \leq \pi(b) = (\pi(a), \pi(b)) \in O$. To avoid ambiguity, we note $O_U^\pi = (O^\pi)_U$.*

2.2 Graphs

Definition 2 (undirected ordered graph). *An undirected ordered graph is a triple $G = (V, E, O)$, where V is a set of vertices, $\{a, b\} \in E \subseteq \binom{V}{2}$ is a set of undirected edges and $O \subseteq V^2$ defines an order on V . A bipartite graph is denoted $G = (P \cup \bar{P}, E, O)$, with no edges between nodes in P nor \bar{P} . The function $n_G: V \rightarrow 2^V$, such that $u \mapsto \{v \mid \{u, v\} \in E\} \setminus \{u\}$ gives all the neighbors of a vertex. The degree of a vertex v is $|n(v)|$ (in this paper all graphs are simple graphs).*

Definition 3 (adjacency matrix). *Given a graph $G = (V, E, O)$ the adjacency matrix of G is a function $\mathbb{N}_{\leq |V|} \times \mathbb{N}_{\leq |V|} \rightarrow \{0, 1\}$, also denoted*

$$\hat{A} = \begin{bmatrix} \hat{a}_{11} & \hat{a}_{12} & \cdots & \hat{a}_{1|V|} \\ \hat{a}_{21} & \hat{a}_{22} & \cdots & \hat{a}_{2|V|} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{a}_{|V|1} & \hat{a}_{|V|2} & \cdots & \hat{a}_{|V||V|} \end{bmatrix} \in \{0, 1\}^{|V| \times |V|}, \text{ such that}$$

$$(i, j) \mapsto \begin{cases} 0 & \text{if } \{u, v\} \notin E \wedge p(u) = i \wedge p(v) = j, \\ 1 & \text{if } \{u, v\} \in E \wedge p(u) = i \wedge p(v) = j. \end{cases}$$

If G is bipartite with parts $R \cup C = V$ and $\forall r \in R . \forall c \in C: r < c$ the adjacency matrix is symmetric, of the form $\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{0}^{|R| \times |R|} & \mathbf{A} \\ \mathbf{A}^T & \mathbf{0}^{|C| \times |C|} \end{bmatrix}$ and \mathbf{A} is called the biadjacency matrix. The dependency matrix is this biadjacency matrix. The element at (i, j) of an $M \times N$ matrix is on the diagonal iff $i = j$.

Definition 4 (total graph). Given an undirected ordered graph $G = (V, E, O)$, a total graph of G is $G^T = (V^T, E^T, O^T)$, where $V^T = V \cup E$ is a set of vertices and $E^T = E \cup \{\{a, \{a, b\}\} \mid \{a, b\} \in E\} \cup \{\{\{a, c\}, \{c, b\}\} \mid \{\{a, c\}, \{c, b\}\} \subseteq E\} \subseteq \binom{V^T}{2}$ is the set of edges, and $O^T \subseteq O \cup V^{T^2}$, i.e. we add all possible vertex-edge edges, edge-vertex edges and edge-edge edges and assign some order to V^T .

Throughout this work we use the order on G^T where all edges from G are larger than all vertices from G and all edges are mutually ordered in lexicographic order.

3 Background

Model checking involves answering questions such as whether or not a system can enter a particular state. Consider Figure 2, which is an example of a (1-safe) Petri net. A Petri net is a bipartite graph where its vertices are places and transitions. Places can contain a positive number of tokens. The graph's edges are called arcs. An outgoing arc means that tokens will be consumed and incoming arc means that tokens will be produced. In Figure 2, after transition t_1 fires, p_4 will have no token, while both p_2 and p_5 get one token. A reachability question is whether or not p_1 will eventually have a token, which it will, after firing t_1 followed by t_4 . A Petri net can model many kinds of systems or protocols such as Lamport's mutual exclusion algorithm, or they can even model biological processes.

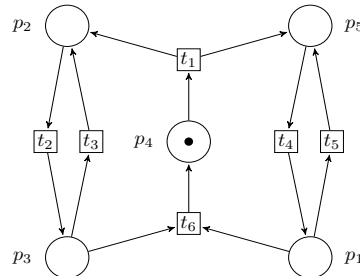


Fig. 2: Example 1-safe Petri net.

The key issue with storing all reachable states in decision diagrams is the ordering of variables, i.e. the order of places p_1, p_2, p_3, p_4, p_5 . Figure 3 shows particular decision diagrams, namely List Decision Diagrams (LDDs) [12], representing the five reachable states of the Petri net, but with different variable orders. Every path from the top left node to the **true** node represents a reachable state. The value in a node indicates the number of tokens. One can see that Figure 3b, whose variable order is computed using Cuthill McKee, has fewer nodes than Figure 3a with the default alphanumeric variable order. Thus to store the decision diagram in Figure 3b requires less memory.

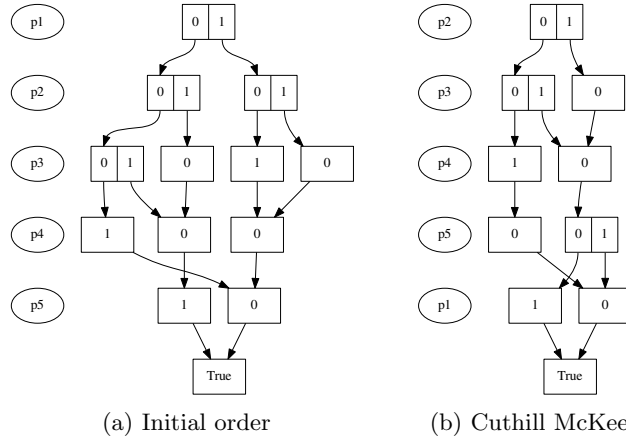


Fig. 3: Reachable states

3.1 Symbolic Reachability Analysis

Symbolic reachability analysis is a strategy that repeatedly applies the transition relation until a fixed point is reached. For efficient symbolic reachability analysis the transition relation can be partitioned [6]. The relation can be partitioned conjunctively, for synchronous systems, or disjunctively for asynchronous systems. The latter approach is used for the Petri net in Figure 2. The following notion of a Partitioned Transition System (PTS) can be used to represent any asynchronous system. Furthermore, PTSs allow us to describe event locality in terms of write dependencies and read dependencies.

Definition 5 (Partitioned Transition System). A Partitioned Transition System (PTS) is a structure $\mathcal{P} = (S^N, \rightarrow^M, \mathbf{s}^0)$, where

- $S^N = \prod_{i=1}^N S_i^N$ is the set of states $\mathbf{s} \in S^N$, which are vectors of N values,
- $\rightarrow^M = \bigcup_{i=1}^M \rightarrow_i$ is the transition relation, which is a union of the M transition groups $\rightarrow_i \subseteq S^{N^2}$ (for $1 \leq i \leq M$), and
- $\mathbf{s}^0 = (s_1^0, \dots, s_N^0) \in S^N$ is the initial state.

We write $\mathbf{s} \rightarrow_i \mathbf{t}$ when $(\mathbf{s}, \mathbf{t}) \in \rightarrow_i$ for $1 \leq i \leq M$, $\mathbf{s} \rightarrow_{\mathcal{P}} \mathbf{t}$ when $(\mathbf{s}, \mathbf{t}) \in \rightarrow^M$ and $S^N = \{S_1^N, \dots, S_N^N\}$ for the set of state variables.

Both write and read independence are expressed in terms of unordered pairs between variables and transitions. In a Petri net a transition is both write and read independent from a place when it neither consumes nor produces a token in that place. If a Petri net is 1-safe, a transition can always safely produce a token when the transition is enabled and all places always have at most 1 token. Thus in 1-safe Petri nets transitions are read independent from places where tokens are produced.

Definition 6 (Write independence). Given a Partitioned Transition System (PTS) $\mathcal{P} = (S^N, \rightarrow^M, \mathbf{s}^0)$, transition group i is write-independent from state variable j , if: $\forall \mathbf{s}, \mathbf{t} \in S^N: (s_1, \dots, s_j, \dots, s_N) \rightarrow_i (t_1, \dots, t_j, \dots, t_N) \implies (s_j = t_j)$, i.e., state variable j is never modified in transition group i . The write set is $\mathcal{W} = \{ \{ \rightarrow_i^M, S_j^N \} \mid \rightarrow_i^M \text{ is not write-independent from } S_j^N \} \subseteq (\rightarrow^M \cup S^N)$.

In Figure 2, t_1 is write independent from p_1 , but write dependent from p_4 .

Definition 7 (write graph). Given a PTS $\mathcal{P} = (S, \rightarrow, \mathbf{s}^0)$ and a write set \mathcal{W} , a write graph is a partially ordered bipartite graph $G^{\mathcal{W}} = (\rightarrow \cup \mathcal{S}, \mathcal{W}, O)$, where $O \subset \rightarrow^2 \cup \mathcal{S}^2$. The biadjacency matrix of the write graph is called the write matrix, denoted $\mathbf{W} \in \{0, 1\}^{|\rightarrow| \times |\mathcal{S}|}$, an element of \mathbf{W} is w_{ij} .

Figure 4 shows the write graph and the write matrix of the Petri net, using the default alphanumeric (partial) order \mathcal{A} .

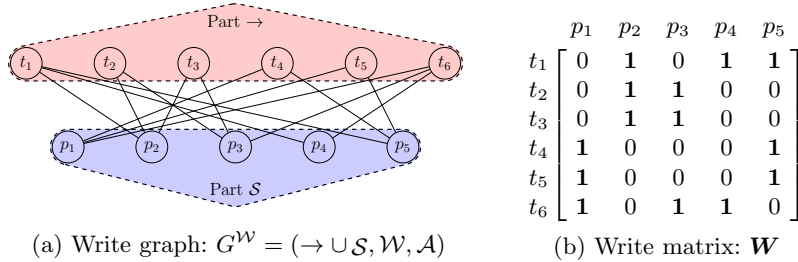


Fig. 4: Write dependencies

Analogous to the definitions regarding write dependence we can define read dependence [24]. Furthermore we can define the *combined graph* and *combined matrix* of which the edges and nonzeros are the elements in the union of the write set and *read set*. Siminiceanu et al. [27] use a similar concept as the combined matrix to define the WES metric. The concept of the *dependency matrix* in earlier papers on LTSMIN [3, 24] is identical to the combined matrix. In this paper we focus on write dependence, because only write dependencies can create nodes in decision diagrams. We do not focus on read dependence, because in Petri nets, there exists no net with only a read dependency on a place (consuming or producing tokens always requires a write dependency). In languages such as PROMELA, DVE and mCRL2 read dependencies do exist without write dependencies, e.g. as variables on the right hand side of assignments.

3.2 Saturation and Peak Nodes

Figure 3 shows that the final set of reachable states benefits from a good variable ordering. In general however, the number of nodes in DDs during reachability analysis is much higher than at the end of reachability. The number of nodes as a function of reachable states is of parabolic form and its maximum – and thus limiting factor in terms of memory usage – is called *peak nodes*. Although good variable orders allow the number of peak nodes to remain low, an advanced reachability strategy is also mandatory for low peak nodes. A reachability algorithm can be expressed as a sequence of applying transitions, i.e. let t_i^- be the application of transition i including the identity. Then $\text{BFS} = ((t_1 \cup \dots \cup t_M)^=)^+$, $\text{CHAINING} = (t_1^- \cdot t_2^- \cdot \dots \cdot t_M^-)^+$ and $\text{SAT-LIKE} = (((t_M^- \cdot t_{M-1}^-)^+ \cdot t_{M-2}^-)^+ \cdot t_{M-1}^-)^+ \cdot t_1^-)^+$. The SAT-LIKE algorithm saturates the bottom of the DD first, to keep peak

nodes low, like the *saturation* algorithm by Ciardo et al. [8]. In LTSmin, the option `--sat-granularity` allows to group multiple transitions for saturation. Within a group of transitions, CHAINING is used. By default, LTSmin creates $N/10$ transitions groups.

3.3 Metrics for Undirected Ordered Graphs

The *bandwidth* of a row in a matrix is the maximal distance to the diagonal of a nonzero in that row. The *span* of a row in a matrix is the distance between the minimal and maximal nonzero.

Definition 8 (bandwidth). *Given a graph $G = (V, E, O)$, the vertex bandwidth is a function $b_G: V \rightarrow \mathbb{N}_{<|V|}^0$, such that*

$$v \mapsto \begin{cases} 0 & \text{if } n_G(v) = \emptyset, \\ \max_{w \in n_G(v)} |p_O(v) - p_O(w)| & \text{otherwise.} \end{cases} \quad \begin{matrix} \text{(a)} \\ \text{(b)} \end{matrix}$$

Definition 9 (span). *Given a graph $G = (V, E, O)$, the vertex span is a function $s_G: V \rightarrow \mathbb{N}_{\leq|V|}^0$, such that*

$$v \mapsto \begin{cases} 0 & \text{if } n_G(v) = \emptyset, \\ \max_{w \in n_G(v)} p_O(w) - \min_{w \in n_G(v)} p_O(w) + 1 & \text{otherwise.} \end{cases} \quad \begin{matrix} \text{(a)} \\ \text{(b)} \end{matrix}$$

In Figure 4 the bandwidth and span of t_4 are $b_{G^w}(t_4) = 3$ and $s_{G^w}(t_4) = 5$.

The *wavefront* of a column in a matrix is the number of rows that have nonzeros within columns smaller or equal than that column.

Definition 10 (wavefront). *Given a graph $G = (V, E, O)$, the vertex frontwidth or vertex wavefront is a function $f_G: V \rightarrow \mathbb{N}_{\leq|V|}$, such that $u \mapsto |\{u\} \cup \{v \mid v \in V \wedge u \neq v \wedge \exists w \in V: \{w, v\} \in E \wedge w \leq u\}|$.*

In Figure 4 the wavefront of p_3 is $f_{G^w}(p_3) = 7$, because all transitions depend on a place in $\{p_1, p_2, p_3\}$. The wavefront of p_1 is $f_{G^w}(p_1) = 4$.

4 Approach

Our approach consists of transforming the partially ordered adjacency graphs of the dependency matrix to totally ordered graphs. This is a requirement for the sparse matrix algorithms in Boost and ViennaCL. Then we provide metrics that can measure the quality of orders on a graph level, rather than on vertex level (Definitions 8 to 10). We then explain how nodal ordering algorithms work on totally ordered graphs and how to apply the resulting permutations to partially ordered graphs. But first, we prove Theorem 1 with the triangle inequality. Theorem 1 shows that span is limited by twice the bandwidth plus the diagonal. This gives the intuition of why bandwidth reduction can be used to reduce span. Reducing span is known to be good for variable orderings in symbolic reachability analysis.

Theorem 1 (bandwidth limits span). *Given a graph $G = (V, E, O)$, we have $\forall v \in V: s_G(v) \leq 2 \cdot b_G(v) + 1$.*

Proof. Take an arbitrary $v \in V$, we have two cases for v :

Case 1 ($n(v) = \emptyset$). By Definition 8a and 9a we have $0 \leq 2 \cdot 0 + 1 \iff s_G(v) \leq 2 \cdot b_G(v) + 1$.

Case 2 ($|n(v)| > 0$). We have two symmetric cases.

Case 2.1 ($|\min_{w \in n(v)} p(w) - p(v)| \geq |p(v) - \max_{w \in n(v)} p(w)|$).

$$\begin{aligned} s_G(v) &= \max_{w \in n(v)} p(w) - \min_{w \in n(v)} p(w) + 1 = |\max_{w \in n(v)} p(w) - \min_{w \in n(v)} p(w)| + 1 \\ &= |\max_{w \in n(v)} p(w) - p(v) + p(v) - \min_{w \in n(v)} p(w)| + 1 \\ &\leq |\max_{w \in n(v)} p(w) - p(v)| + |p(v) - \min_{w \in n(v)} p(w)| + 1 \leq 2 \cdot |p(v) - \min_{w \in n(v)} p(w)| + 1 \\ &= 2 \cdot \max_{w \in n(v)} |p(v) - p(w)| + 1 = 2 \cdot b_G(v) + 1. \end{aligned}$$

Case 2.2 ($|p(v) - \max_{w \in n(v)} p(w)| \geq |\min_{w \in n(v)} p(w) - p(v)|$).

Symmetric to 2.1.

Since v is arbitrary we are done. \square

4.1 Total Orders for Dependency Graphs

The bandwidth and wavefront reduction algorithms discussed in this paper can only be run on the underlying graphs of symmetric matrices. The method of symmetrizing suggested by Reid et al. [25] is to create the matrix \hat{A} from Definition 3, of which an example is shown in Figure 5. The key difference between the symmetric matrix in Figure 5 and asymmetric matrix in Figure 4 is that

	t_1	t_2	t_3	t_4	t_5	t_6	p_1	p_2	p_3	p_4	p_5
t_1	0	0	0	0	0	0	0	1	0	1	1
t_2	0	0	0	0	0	0	0	0	1	1	0
t_3	0	0	0	0	0	0	0	0	1	1	0
t_4	0	0	0	0	0	0	1	0	0	0	1
t_5	0	0	0	0	0	0	1	0	0	0	1
t_6	0	0	0	0	0	0	1	0	1	1	0
p_1	0	0	0	1	1	1	0	0	0	0	0
p_2	1	1	1	0	0	0	0	0	0	0	0
p_3	0	1	1	0	0	1	0	0	0	0	0
p_4	1	0	0	0	0	1	0	0	0	0	0
p_5	1	0	0	1	1	0	0	0	0	0	0

Fig. 5: Symmetrized write matrix

symmetric matrix implies a total order on the underlying graph, while the asymmetric matrix implies a partial order. The total order implied is indeed $T = t_1 < t_2 < t_3 < t_4 < t_5 < t_6 < p_1 < p_2 < p_3 < p_4 < p_5$, while the partial order is $P = t_1 < t_2 < t_3 < t_4 < t_5 < t_6 \cup p_1 < p_2 < p_3 < p_4 < p_5$. Technically we have two undirected ordered graphs of the form $G = (V, E, T)$ and $H = (V, E, P)$. On both graphs we can define aggregation and normalization functions, which allow us to measure the quality on a graph level, instead of vertex level. A key observation on this approach is that we can actually choose many total orders as long as the partial order remains the same (i.e. $P \subseteq T$). This implies different results for computing bandwidth, span and wavefront on G , but not on H . Although Definitions 8 and 10 allows measuring bandwidth and wavefront on H , it is currently not known whether these metrics are actually appropriate for symbolic model checking. We therefore compute those two

metrics on G , because this is known to be appropriate for sparse matrix solvers and is already supported by Boost and ViennaCL. To see whether bandwidth and wavefront computed on H is appropriate for symbolic reachability analysis requires the same approach taken by Ciardo et al. in [27], where many samples of a metric are generated of a specific model and analyzed. The results from our benchmark do show however, that metrics computed on G and H are related.

4.2 Aggregation and Normalization for graph metrics

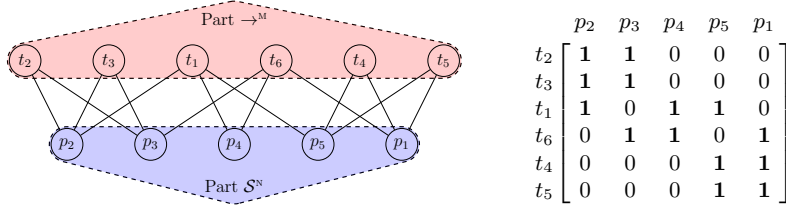
name	aggregation	normalization	value
Bandwidth	$\max_{v \in V} b_G(v)$	$M + N$	10 (.91)
Profile	$\sum_{v \in V} b_G(v)$	$(M + N)^2$	87 (.72)
Span	$\sum_{v \in V} s_G(v)$	$(M + N)^2$	44 (.36)
Average wavefront	$\sum_{v \in V} f_G(v)/(M + N)$	$M + N$	4.3 (.39)
Event Span	$\sum_{\rightarrow \in \rightarrow^M} s_H(\rightarrow)$	$M \cdot N$	22 (.73)
Weighted Event Span	$\sum_{\rightarrow \in \rightarrow^M} s_H(\rightarrow) \cdot \frac{N - \min_{v \in n(\rightarrow)} p_P(v)}{N/2}$	$M \cdot N$	41 (1.4)

Fig. 6: Aggregation and normalization

Figure 6 shows a list of aggregated and normalized metrics for bandwidth, span and wavefront we compute in our benchmarks. The first four metrics are from the literature on sparse matrix solvers. The last two are by Siminiceanu et al. [27]. Given a PTS $\mathcal{P} = (S^N, \rightarrow^M, s^0)$, all three type of metrics can be computed on the totally ordered write graph G in Boost and ViennaCL or the partially ordered write graph H in LTSMIN. The column *normalization* indicates the factor which should be divided by to obtain a number between zero and one, except for WES. The column *value* contains the computed values (and normalized values) for the Petri net with the total order from Figure 5 (graph G) and the partial order from Figure 4 (graph H). Note that the aggregation functions for G^T are the same as for G , but are not computed here.

4.3 Nodal Ordering

Cuthill Mckee [11] is a nodal ordering algorithm for bandwidth reduction. The algorithm is a simple breadth-first graph traversal algorithm that visits neighbors of a vertex in increasing order of degree. For picking a starting vertex, there are several options. The implementation in the Boost graph library uses a pseudo-peripheral pair heuristic [13] for each disconnected component. A simpler approach is to take the smallest vertex with minimum degree. Using the latter option, the order in which Cuthill McKee visits vertices is $O^\pi = t_2 < p_2 < p_3 < t_3 < t_1 < t_6 < p_4 < p_5 < p_1 < t_4 < t_5$. The reordered write graph, (symmetrized) write matrix and metrics of the Petri net are shown in Figure 7. The *band* and low *event span* in Figure 7b is clearly visible. Note that indeed the partial order for the reordered write graph is $O_{S^N}^\pi \cup O_{\rightarrow^M}^\pi$. If a nodal ordering



$$\begin{matrix}
 & p_2 & p_3 & p_4 & p_5 & p_1 \\
 \begin{matrix} t_2 \\ t_3 \\ t_1 \\ t_6 \\ t_4 \\ t_5 \end{matrix} & \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}
 \end{matrix}$$

(a) Write graph: $(\rightarrow^M \cup \mathcal{S}^N, \mathcal{W}, \mathcal{O}_{\mathcal{S}^N}^\pi \cup \mathcal{O}_{\rightarrow^M}^\pi)$

(b) Write matrix

Bandwidth	3 (.27)
Profile	40 (.33)
Span	48 (.40)
Avg wavefront	3.2 (.29)
ES	16 (.53)
WES	26 (.87)

(c) Metrics

$$\begin{matrix}
 & t_2 & p_2 & p_3 & t_3 & t_1 & t_6 & p_4 & p_5 & p_1 & t_4 & t_5 \\
 \begin{matrix} t_2 \\ p_2 \\ p_3 \\ t_3 \\ t_1 \\ t_6 \\ p_4 \\ p_5 \\ p_1 \\ t_4 \\ t_5 \end{matrix} & \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}
 \end{matrix}$$

(d) Symmetrized write matrix

Fig. 7: Reordered Petri net

Algorithm	Package	Time complexity	Type	Graph
Cuthill McKee	Boost	$\mathcal{O}(\hat{D} \cdot \log \hat{D} \cdot V)$	bandwidth	totally ordered
King [20]		$\mathcal{O}(\hat{D}^2 \cdot \log \hat{D} \cdot E)$	bandwidth, profile	undirected
Sloan		$\mathcal{O}(\hat{D} \cdot \log \hat{D} \cdot V)$	profile, wavefront	graph
Cuthill McKee	ViennaCL	n/a	bandwidth	totally ordered
adv. Cuthill McKee		n/a	bandwidth	directed graph
GPS [14]		n/a	bandwidth, profile	
Column Swap	LTSMIN	$\mathcal{O}(M^2 \cdot N^4)$	event span	asymmetric matrix
Notation: $\hat{D} = \max_{v \in V} n(v) $ (maximum degree)				

Fig. 8: List of reordering algorithms

algorithm is run on the total graph of the write graph to obtain $O^{T\pi}$, the partial order for the write graph is simply $\mathcal{O}_{\mathcal{S}^N}^{T\pi} \cup \mathcal{O}_{\rightarrow^M}^{T\pi}$.

Figure 8 lists all the reordering algorithms that we have considered and their attributes. There are four categories of algorithms, those that reduce bandwidth, bandwidth and profile, reduce wavefront and profile, and those that reduce event span. In both Boost and ViennaCL the Cuthill McKee algorithm is implemented, which use an undirected and directed graph respectively, as datastructures. Section 5 confirms that the Cuthill McKee implementation differ in both tools. The GPS algorithm is only implemented in ViennaCL and the time complexity of algorithms in ViennaCL is not precisely known, but must be in the order of similar BFS algorithms. One special algorithm in the list is the column swap algorithm, which is a heuristic algorithm in LTSMIN. Its key feature is that column swap generates permutations that give low event span. In general it produces good ES, but is unpractical for large matrices.

Figures 9a and 9b show what dependency matrices can look like when reordered with some algorithms from Figure 8 including their weighted event span.

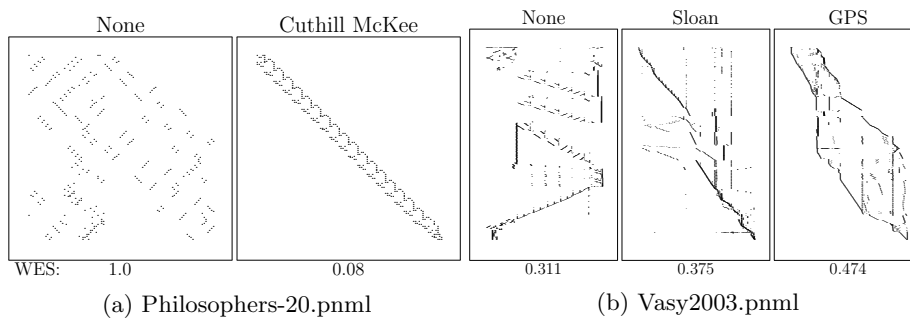


Fig. 9: Example reorderings

The first two matrices are of a model with 20 dining philosophers, one of the best results achieved in our benchmarks. Even on instances with 5000 philosophers (*25.000 variables*) we get very small event span. The order is computed within milliseconds and the resulting number of peak nodes is several orders of magnitude smaller than without reordering. The matrices from the Vasy2003 model show the more typical structure of dependency matrices, e.g. the *band* the GPS algorithm produces is clearly visible. The Sloan algorithm produces the best order for Vasy2003, probably because of the large concentration of many nonzeros on the bottom end of the diagonal. We believe this concentration of nonzeros is produced due to wavefront reduction and is beneficial to the saturation algorithm.

When benchmarking with LTSMIN, we actually need the banded structure of permuted matrices to appear from the top right to bottom left. This can be done by inverting the permutation of transitions (rows) or permutation of variables (columns), these operations are known as *horizontal flip* and *vertical flip* respectively. A flip operation is necessary, because it influences the result of the CHAINING part in SAT-LIKE. It is yet unclear which flip operation works best, thus in the benchmark we try both.

5 Results

We have benchmarked the sparse matrix ordering algorithms on 361 different Petri net models from the 2015 model checking contest³. Reproduction instructions can be found online⁴. The benchmark consists of 6 different algorithms from Figure 8, computed on the write graph and its total graph. Additionally, the column swap algorithm is run, as well as no reordering algorithm. Furthermore we added the options *Horizontal Flip* and *Vertical Flip* which invert the permutation on rows and columns, respectively. We do not perform a flip operation when no reordering is done. Our benchmark consists of 53 categories, but we show only 27 ($= ((6 \cdot 2) + 1) \cdot 2 + 1$) categories, since we omit the results of the combined graph. In total we did 114798 ($= 53 \cdot 361 \cdot (1 + 5)$) experiments, which contains 1 run for each category to obtain statistics (e.g. peak nodes and metrics) and 5 runs to measure time. Our results include only those models, which

³ <http://mcc.lip6.fr>

⁴ <https://github.com/utwente-fmt/BW-TACAS-2016>

all categories were able to compute within 30 minutes and 4 GB of memory. This resulted in 110 usable models. We ran 1 experiment on 1 core on 12 single and 32 dual socket AMD Opteron 4386 processors, with 64 GB of memory and Ubuntu 14.04 LTS. To be able to complete the entire benchmark in two days we ran 608 experiments simultaneously. The advantage of this approach is that we were able to gather lots of data on different algorithms and models. The disadvantage is that time measurements are less reliable. However, the measurement of peak nodes is more important than time measurements.

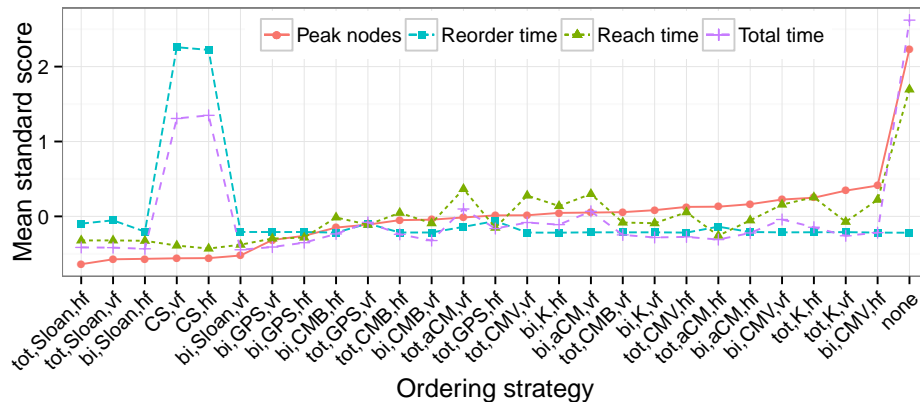


Fig. 10: Performance of reordering algorithms

Figure 10 shows which algorithm produces the best result. The results are ordered in ascending number of peak nodes. The category *tot,Sloan,hf* means we performed the **Sloan** algorithm on the **total** graph of the write graph. Then we computed all metrics in Figure 6, performed a **horizontal flip** and ran the reachability algorithm. The *Mean Standard Score* (MSS), also known as mean z-score, indicates how well a category performs relative to the mean (μ) observed value of models divided by the standard deviation (σ). For example, the *tot,Sloan,hf* category has the lowest score for peak nodes (it appears on the left), while *none* has the lowest score for reorder time (since no reordering is done). More precisely, let C be the set of categories, m a metric such as peak nodes of a category and a model, and P the set of models with completed runs, the MSS of a category $c \in C$ is: $\sum_{p \in P} \frac{m(p,c) - \mu_{c' \in C} m(p,c')}{\sigma_{c' \in C} m(p,c')} / |P|$. Other abbreviations in Figure 10 that require explanation are: CS = **C**olumn **S**wap, bi = **b**ipartite graph, CMB = **C**uthill **M**cKee in **B**oost, aCM = **a**dvanced **C**uthill **M**cKee, K = **K**ing and CMV = **C**uthill **M**cKee in **V**ienna**C**L.

Figure 11 shows the MSS for the matrix metrics with the same order for categories as Figure 10. The totally ordered graph is not created with *column swap* and *none*, thus bandwidth, profile, span and wavefront are omitted for those categories. Furthermore the μ and σ for bandwidth, profile, span and wavefront are computed per graph type, i.e. for all bipartite graphs with total order and for all total graphs, because those two graph types have different number of vertices and edges. A few observations can be made, such as that the WES metric

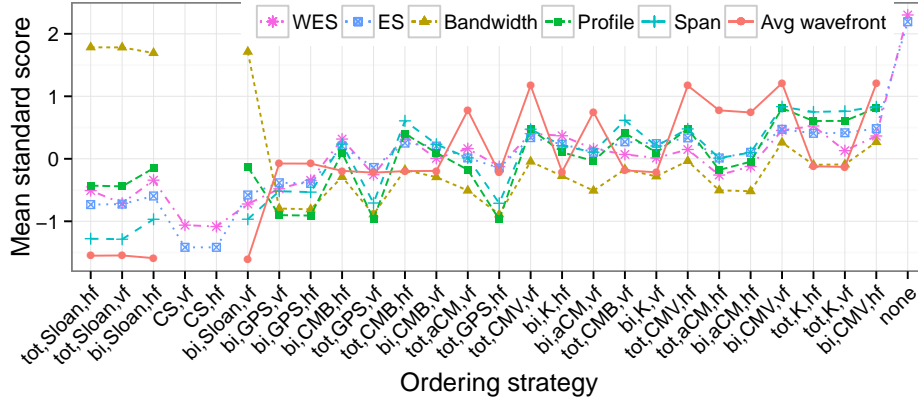
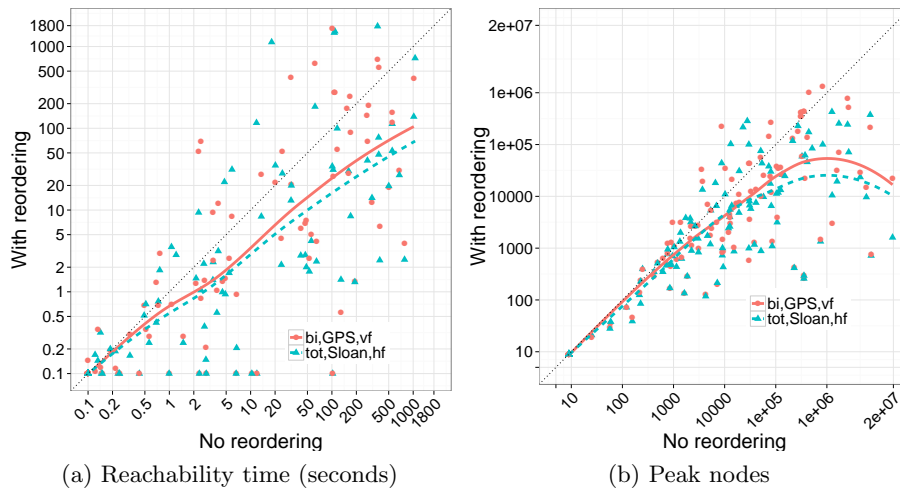


Fig. 11: Metrics produced with reordering algorithms

is a good predictor for the number of peak nodes in our benchmark; the WES increases with the number of peak nodes. It is a good idea to perform any reordering algorithm rather than none. The *Sloan* algorithm performs well for all metrics, but bandwidth. The GPS algorithm performs second best to Sloan. The wavefront is higher for GPS algorithms and can thus explain why Sloan performs better than GPS in terms of peak nodes.

Figure 12 shows how well the Sloan algorithm on the total graph and GPS on the bipartite graph perform relative to no reordering. Figure 12a illustrates the reachability time in seconds and Figure 12b shows the number of peak nodes. Both scatter plots have logarithmic axes and contain data of the 110 completed models for both algorithms. If a point lies below the line $x = y$, the result with reordering is better. The plots also show locally weighted regression lines. These lines indicate that for larger models the effect of reordering is more dramatic than for smaller models. In larger cases we see an improvement in peak nodes of factors 1000 to 10000, and time of factors 10 to 100. There are however some models that do not benefit from the two reordering algorithms.



(a) Reachability time (seconds)

(b) Peak nodes

Fig. 12: Comparison of Sloan and GPS to no reordering

6 Conclusion and Related Work

In Section 5 we have shown that bandwidth and wavefront reduction is clearly useful in symbolic model checking. The best algorithm for variable reordering is Sloan. Empirical observation of results presented in this paper and the results of the 2015 model checking contest (without the traditional nodal ordering algorithms in LTSMIN) show a big improvement and at least on par with other competitors in the statespace category. There are two branches of related work. The first are other model checkers, such as SMART [9], MARCIE [15] and NuSMV [10]. The SMART tool employs advanced saturation algorithms, which can be used to confirm whether bandwidth and wavefront reduction is useful in other model checkers as well. Readily available variable ordering algorithms are hard to find, however the MARCIE model checker implements the Noack [22] variable ordering algorithm, which can be used to compare our proposed algorithms with.

Our approach works for disjunctive partitioning schemes, the question remains however, whether or not our method also works for conjunctive [6] partitioning schemes, such as in NuSMV. Furthermore bandwidth and wavefront reduction may be applicable to SAT/SMT solving, where rows in the matrix are clauses and columns are variables. Of interest is whether or not our proposed method can achieve similar results as the FORCE [1] heuristic, which reduces the variable cut and *span*.

The second branch of related work is other bandwidth and wavefront reduction algorithms. Kaveh [19] discusses many different graph transformations of the adjacency graph on which nodal ordering algorithms can be run. We picked only the total graph, because running nodal ordering algorithms on total graphs does not require modifying these algorithms. Reid et al. [25] provide two more methods of symmetrizing an asymmetric matrix \mathbf{A} , namely $\mathbf{A} + \mathbf{A}^T$ and $\mathbf{A} \cdot \mathbf{A}^T$. Additionally the authors provide a modified Cuthill McKee algorithm that can be run on an asymmetric matrix directly, available in the HSL library⁵. A survey [23] covers the state of the art in bandwidth reduction, including meta-heuristic algorithms, of which many have been developed in the past decade.

Another nodal ordering algorithm which is often used in iterative sparse matrix solvers is the Approximate Minimum Degree (AMD) [2] algorithm, implemented in SuiteSparse⁶. AMD produces matrices of the form shown in Figure 13. AMD can also be applied on symmetrized matrices, but we have found AMD not applicable to symbolic model checking, judging by the form of the reordered matrices it produces. Recently, advances have been made in parallelizing [18] nodal ordering algorithms. We think however, that dependency matrices are too small to benefit greatly from these parallelized algorithms.

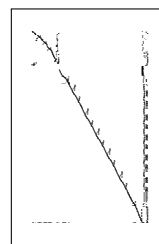


Fig. 13: AMD

Acknowledgements We would like to thank Marcus Gerhold, Erik Kemp and Alfons Laarman for making helpful contributions to this paper.

⁵ <http://www.hsl.rl.ac.uk>

⁶ <http://faculty.cse.tamu.edu/davis/suitesparse.html>

References

1. Aloul, F.A., Markov, I.L., Sakallah, K.A.: FORCE: a fast and easy-to-implement variable-ordering heuristic. In: Stan, M.R., Garrett, D., Nakajima, K. (eds.) Proceedings of the 13th ACM Great Lakes Symposium on VLSI 2003, Washington, DC, USA, April 28-29, 2003. pp. 116–119. ACM (2003), <http://doi.acm.org/10.1145/764808.764839>
2. Amestoy, P., Enseeiht-Irit, Davis, T.A., Duff, I.S.: Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.* 30(3), 381–388 (2004), <http://doi.acm.org/10.1145/1024074.1024081>
3. Blom, S., van de Pol, J.: Symbolic Reachability for Process Algebras with Recursive Data Types. In: ICTAC 2008. LNCS, vol. 5160. Springer (2008)
4. Bollig, B., Wegener, I.: Improving the Variable Ordering of OBDDs Is NP-Complete. *IEEE Trans. Comput.* 45(9), 993–1002 (Sep 1996), <http://dx.doi.org/10.1109/12.537122>
5. Bryant, R.E.: Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Computers* 35(8), 677–691 (1986), <http://doi.ieeecomputersociety.org/10.1109/TC.1986.1676819>
6. Burch, J.R., Clarke, E.M., Long, D.E.: Symbolic model checking with partitioned transition relations. In: VLSI 1991 (1991)
7. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. In: LICS 1990. IEEE (1990)
8. Ciardo, G., Marmorstein, R.M., Siminiceanu, R.: The saturation algorithm for symbolic state-space exploration. *STTT* 8(1), 4–25 (2006), <http://dx.doi.org/10.1007/s10009-005-0188-7>
9. Ciardo, G., Miner, A.S., Wan, M.: Advanced features in SMART: the stochastic model checking analyzer for reliability and timing. *SIGMETRICS Performance Evaluation Review* 36(4), 58–63 (2009), <http://doi.acm.org/10.1145/1530873.1530885>
10. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In: CAV 2002. LNCS, vol. 2404. Springer (2002)
11. Cuthill, E., McKee, J.: Reducing the Bandwidth of Sparse Symmetric Matrices. In: Proceedings of the 1969 24th National Conference. pp. 157–172. ACM '69, ACM, New York, NY, USA (1969), <http://doi.acm.org/10.1145/800195.805928>
12. van Dijk, T., van de Pol, J.: Sylvan: Multi-Core Decision Diagrams. In: Baier, C., Tinelli, C. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science, vol. 9035, pp. 677–691. Springer Berlin Heidelberg (2015), http://dx.doi.org/10.1007/978-3-662-46681-0_60
13. George, A., Liu, J.W.H.: An Implementation of a Pseudoperipheral Node Finder. *ACM Trans. Math. Softw.* 5(3), 284–295 (Sep 1979), <http://doi.acm.org/10.1145/355841.355845>
14. Gibbs, N.E., Poole, Jr, W.G., Stockmeyer, P.K.: An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM Journal on Numerical Analysis* 13(2), 236–250 (1976)
15. Heiner, M., Rohr, C., Schwarick, M.: MARCIE - Model checking And Reachability analysis done effiCIently. In: Colom, J., Desel, J. (eds.) Proc. PETRI NETS 2013. LNCS, vol. 7927, pp. 389–399. Springer (June 2013), http://link.springer.com/chapter/10.1007/978-3-642-38697-8_21

16. Kam, T., Villa, T., Brayton, R.: Multi-valued decision diagrams: theory and applications. *Multiple-Valued Logic* (4) (1998)
17. Kant, G., Laarman, A.W., Meijer, J.J.G., van de Pol, J.C., Blom, S.C.C., van Dijk, T.: LTSmin: High-Performance Language-Independent Model Checking. In: Baier, C., Tinelli, C. (eds.) *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2015*, London, United Kingdom. *Lecture Notes in Computer Science*, vol. 9035, pp. 692–707. Springer Verlag, London (April 2015)
18. Karantasis, K.I., Lenharth, A., Nguyen, D., Garzarán, M.J., Pingali, K.: Parallelization of Reordering Algorithms for Bandwidth and Wavefront Reduction. In: Damkroger, T., Dongarra, J. (eds.) *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2014*, New Orleans, LA, USA, November 16-21, 2014. pp. 921–932. IEEE (2014), <http://dx.doi.org/10.1109/SC.2014.80>
19. Kaveh, A.: *Ordering for Optimal Patterns of Structural Matrices*, pp. 191–271. John Wiley & Sons, Ltd (2006), <http://dx.doi.org/10.1002/9780470033326.ch5>
20. King, I.P.: An automatic reordering scheme for simultaneous equations derived from network systems. *International Journal for Numerical Methods in Engineering* 2(4), 523–533 (1970), <http://dx.doi.org/10.1002/nme.1620020406>
21. Kordon, F., Garavel, H., Hillah, L.M., Hulin-Hubard, F., Linard, A., Beccuti, M., Hamez, A., Lopez-Bobeda, E., Jezequel, L., Meijer, J., Paviot-Adet, E., Rodriguez, C., Rohr, C., Srba, J., Thierry-Mieg, Y., Wolf, K.: Complete Results for the 2015 Edition of the Model Checking Contest. <http://mcc.lip6.fr/2015/results.php> (2015)
22. Liu, F.: *Manual for Snoopy2Prism Export*. Tech. rep., Brandenburg University of Technology Cottbus, Department of Computer Science (March 2009)
23. Maftaiu-Scai, L.O.: The Bandwidths of a Matrix. A Survey of Algorithms. *Annals of West University of Timisoara-Mathematics* 52(2), 183–223 (2014)
24. Meijer, J., Kant, G., Blom, S., van de Pol, J.: Read, Write and Copy Dependencies for Symbolic Model Checking. In: Yahav, E. (ed.) *Hardware and Software: Verification and Testing - 10th International Haifa Verification Conference, HVC 2014*, Haifa, Israel, November 18-20, 2014. *Proceedings. Lecture Notes in Computer Science*, vol. 8855, pp. 204–219. Springer (2014), http://dx.doi.org/10.1007/978-3-319-13338-6_16
25. Reid, J.K., Scott, J.A.: Reducing the Total Bandwidth of a Sparse Unsymmetric Matrix. *SIAM J. Matrix Analysis Applications* 28(3), 805–821 (2006), <http://dx.doi.org/10.1137/050629938>
26. Rupp, K., Rudolf, F., Weinbub, J.: ViennaCL - A High Level Linear Algebra Library for GPUs and Multi-Core CPUs. In: *Intl. Workshop on GPUs and Scientific Applications*. pp. 51–56 (2010)
27. Siminiceanu, R., Ciardo, G.: New Metrics for Static Variable Ordering in Decision Diagrams. In: Hermanns, H., Palsberg, J. (eds.) *Tools and Algorithms for the Construction and Analysis of Systems, 12th International Conference, TACAS 2006 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006*, Vienna, Austria, March 25 - April 2, 2006, *Proceedings. Lecture Notes in Computer Science*, vol. 3920, pp. 90–104. Springer (2006), http://dx.doi.org/10.1007/11691372_6
28. Sloan, S.W.: A FORTRAN program for profile and wavefront reduction. *International Journal for Numerical Methods in Engineering* 28(11), 2651–2679 (1989), <http://dx.doi.org/10.1002/nme.1620281111>