New Metrics for Static Variable Ordering in Decision Diagrams*

Radu I. Siminiceanu[1] and Gianfranco Ciardo[2]

[1] National Institute of Aerospace, Hampton, Virgina 23666
[2] University of California, Riverside, CA 92521

Abstract. We investigate a new class of metrics to find good variable orders for decision diagrams in symbolic state-space generation. Most of the previous work on static ordering is centered around the concept of minimum variable span, which can also be found in the literature under several other names. We use a similar concept, but applied to event span, and generalize it to a family of metrics parameterized by a moment, where the metric of moment 0 is the combined event span. Finding a good variable order is then reduced to optimizing one of these metrics, and we design extensive experiments to evaluate them. First, we investigate how the actual optimal order performs in state-space generation, when it can be computed by evaluating all possible permutations. Then, we study the performance of these metrics on selected models and compare their impact on two different state-space generation algorithms: classic breadth-first and our own saturation strategy. We conclude that the new metric of moment 1 is the best choice. In particular, the saturation algorithm seems to benefit the most from using it, as it achieves the better performance in nearly 80% of the cases.

1 Introduction

In automated system verification, the performance of symbolic model checking algorithms based on binary decision diagrams (BDD) [4] is strongly influenced by the variable ordering of the model. While the boundaries of what is now amenable to BDD technology have been constantly pushed, many industrial-size applications are still out of reach. A critical factor is that finding the optimal BDD variable order is an NP-complete problem. Not knowing what the optimum BDD performance could be, leaves the issue of what is actually achievable by this method still uncertain.

Various heuristics have been proposed to tackle the variable order issue. One direction is to attempt to find a good variable order statically [1, 2, 3, 20, 23], i.e., *prior* to generating the state space, hoping to keep the peak size of the BDD as small as possible. The other direction is to dynamically alter the variable ordering *during* state-space generation [24], usually when the size of the BDD becomes too large, to reduce the current BDD size.

* Work supported in part by the National Aeronautics and Space Administration under grant NCC1-02043 and by the National Science Foundation under grants CNS-0501747 and CNS-0501748.

Most of the previous efforts on static ordering is centered around the concept of *minimum variable span*, variants of which have been described as normalized average lifetime [22], smallest communication graph [1], and diagonal dependency matrix [15]. At its core is the idea that clustering variables that are interrelated (in the transition relation expression, combinatorial circuit design, dependency matrix, etc.) yields better results. This was hinted as early as in [6] and also supports the idea of event locality [9], which ultimately produced the saturation strategy [10], an efficient state-space generation algorithm.

However, simply minimizing the span as a metric does not always guarantee good results in practice. Indeed, variable orders with the same span may produce drastically different results. In the process of developing the saturation strategy, we observed that the dynamics of the BDD growth can be significantly different than for the classic breadth-first iterations. In general, the complexity of symbolic state-space generation depends not only on the overall number of BDD nodes, but also on the location of the region of BDD levels affected by each event. In most cases, some regions tend to grow much larger than the rest. Precisely pinpointing where those levels are concentrated cannot be done in advance, as this is largely model-dependent, but experience indicates that the BDDs grow larger mostly in the middle or middle-bottom area. Therefore, BDD node operations tend to be more costly if performed at the top levels, as the recursive calls propagate downstream. In this paper, we propose a metric focused on the *event span* (over variables), rather than the *variable span*, and propose a generalized version of this metric that takes into account the location of the span with respect to the range of state variables.

The remainder of the paper is structured as follows. Section 2 contains a brief digest of previous work on variable ordering. Section 3 recalls the background on symbolic state-space generation and introduces the new metrics. Section 4 discusses the experimental results to evaluate the significance of these metrics in the context of the saturation and breadth-first iteration strategies. Section 5 concludes and discusses future work.

2 Related Work

The importance of clustering interdependent variables was first pointed out by Burch, Clarke, and Long [6]. Fujita et al. [15] provided an early overview and evaluation of BDD variable orderings. On a closely related subject, a first static heuristic for image computation with a partitioned transition relation was proposed by Geist and Beer [13], based on the idea of ordering the conjuncts depending on the number of affected variables. IWLS95, another successful but quite elaborate heuristic, was proposed in [25] and is still widely used in various BDD packages. Aziz et al. [1] also suggested clustering variables that depend on each other based on an underlying communication graph.

Moon et al. [22] discusses the normalized average lifetime metric in the context of efficiently applying the transition relation of a system using BDDs, either disjunctively or conjunctively. They report significant improvements over previous heuristics in performing image computation within a unified framework that combines conjunctive and splitting methods. In the same context, Chauhan et al. [7] studied different algorithms for optimizing the lifetime metric and concluded that the simulated annealing algorithm achieves the best results. They also proved that the problem of minimizing the normalized average lifetime metric is NP-complete.

Closer to our approach, [3] attempts to alter the minimum event span method. Variables are assigned different weights, according to how many events in the model affect them, then they are statically arranged in decreasing weight order. MINCE [2] is a similar heuristic in the context of both BDD and SAT-based verification, which exploits information from the conjunctive normal form.

Solving the BDD minimization problem by means of genetic algorithms is seen in [14, 23]. One drawback in this type of work is that the evaluation function of the chromosomes is the actual size of the resulting BDD for that order, hence the optimization process is extremely time consuming. In Section 4.5, we propose a much faster approach, where the fitness function is the value of the weighted event span metric, which can be computed statically from the model information for each order.

Other techniques for variable ordering that do not directly employ optimizing a metric are found in [20], which proposes a sampling heuristic, [18], which introduces the scatter search, and [17], which studies a learning based method.

3 Variable Ordering in Symbolic State-Space Generation

We focus on the important problem of symbolically generating the state-space $\mathcal{S}$ of a discrete-state model. We assume a high-level description of the model where each state $\mathbf{i}$ is a K-tuple of integer variables, $\mathbf{i} = (i_K, ..., i_1)$. Each of these variables i_k is in some range $\mathcal{S}_k = \{0, 1, ..., n_k - 1\}$, so that the *potential state space* of the model is $\widehat{\mathcal{S}} = \mathcal{S}_K \times \cdots \times \mathcal{S}_1$. The model has an initial state, or, in full generality, an initial set of states $\mathcal{S}^{init} \subseteq \widehat{\mathcal{S}}$. A *next-state function* of the form $\mathcal{N} : \widehat{\mathcal{S}} \rightarrow 2^{\widehat{\mathcal{S}}}$ specifies the set of states reachable from each state, we can also think of it as a *transition relation* of the form $\mathcal{R} \subseteq \widehat{\mathcal{S}} \times \widehat{\mathcal{S}}$, where $\mathbf{j} \in \mathcal{N}(\mathbf{i}) \Leftrightarrow (\mathbf{i}, \mathbf{j}) \in \mathcal{R}$. We are interested in computing and storing the state space $\mathcal{S}$, which can be defined as the smallest set containing $\mathcal{S}^{init}$ and satisfying the fixed-point equation $\mathcal{X} = \mathcal{X} \cup \mathcal{N}(\mathcal{X})$.

Symbolic methods to compute $\mathcal{S}$ use *decision diagrams.* We consider *quasi-reduced ordered multi-valued decision diagrams (MDDs)* [19], formally defined as a directed acyclic edge-labeled multi-graph where:

- Each node p belongs to a *level* $k \in \{K, ..., 1, 0\}$, denoted *p.lvl*.
- There is a single *root* node r at level K
- Level 0 can only contain the two *terminal* nodes *Zero* and *One*.
- A node p at level $k > 0$ has n_k outgoing edges, labeled from 0 to $n_k - 1$. The edge labeled by i_k points to a node q at level $k - 1$; we write $p[i_k] = q$.
- Given nodes p and q at level $k > 0$, if $p[i_k] = q[i_k]$ for all $i_k \in \mathcal{S}_k$, then $p = q$, i.e., there are no *duplicates.*

The set of states encoded by an MDD is $\mathcal{B}(r)$, defined recursively as

$$\mathcal{B}(p) = \begin{cases} \bigcup_{i_k \in \mathcal{S}_k} \{i_k\} \times \mathcal{B}(p[i_k]) & \text{if } \ p.lvl = k > 1 \\ \{i_1 : p[i_1] = One\} & \text{if } \ p.lvl = 1 \end{cases} .$$

A basic *breadth-first-search* (BFS) algorithm to generate $\mathcal{S}$ implements exactly the fixed-point definition of $\mathcal{S}$, by initializing $\mathcal{S}$ to $\mathcal{S}^{init}$, then repeatedly updating it to include the states reachable from it in one (more) application of $\mathcal{N}$, until no more new states are found, i.e., until $\mathcal{N}(\mathcal{S}) \subseteq \mathcal{S}$. A $2K$-level MDD, fixed for the duration of the iterations, is used to store the next-state function $\mathcal{N}$, while a K-level MDD, which grows and shrinks during the iterations, is used to store $\mathcal{S}$. The *peak size* of this second MDD is critical, as it can exceed the available memory.

To reduce the peak memory requirements of symbolic state-space generation, especially for *globally-asynchronous locally-synchronous systems (GALS)*, we proposed an alternative algorithm called *saturation* [10]. At its core is the recognition that, in GALS, most events exhibit strong *locality*, i.e., they affect only a small subset of the state variables, while the other state variables are subject to *identity transformations*, i.e., they do not change.

Saturation requires a next-state function *disjunctively-partitioned* according to a set $\mathcal{E}$ of (asynchronous) *events* in the high-level model, $\mathcal{N} = \bigcup_{e \in \mathcal{E}} \mathcal{N}_e$. As initially defined, saturation also requires that each $\mathcal{N}_e$ be *conjunctively-partitioned* into K *local* functions, $\mathcal{N}_e = \mathcal{N}_{K,e} \times \cdots \times \mathcal{N}_{1,e}$, each one describing the interaction between event e and a state variable k, $\mathcal{N}_{k,e} : \mathcal{S}_k \to 2^{\mathcal{S}_k}$. Such a decomposition always exists. However, for Petri nets, for example, the $\mathcal{N}_{k,e}$ functions always exist regardless of how many places are grouped into a single state variable while, in other formalisms, this conjunctive decomposition might exist only if we merge state variables or split events, potentially leading to exponential growth of the node sizes or of the number of events. A more recent version of saturation allows the conjuncts to be functions of multiple state variables [12], but we limit our discussion to the original version for simplicity (the findings of this paper are equally applicable to this general version of saturation).

We say that level k does not *depend* on event e, and vice-versa, if $\mathcal{N}_{k,e} = \mathcal{I}_k$, the identity function, i.e., $\mathcal{N}_{k,e}(i_k) = \{i_k\}$ for every local state $i_k \in \mathcal{S}_k$. Then, we define $Top(e) = \max\{k : \mathcal{N}_{k,e} \neq \mathcal{I}_k\}$ and $Bot(e) = \min\{k : \mathcal{N}_{k,e} \neq \mathcal{I}_k\}$ to be the highest and lowest levels that depend on event e. Letting $\mathcal{N}_k = \bigcup_{e:\, Top(e)=k} \mathcal{N}_e$ and $\mathcal{N}_{\leq k} = \bigcup_{e:\, Top(e) \leq k} \mathcal{N}_e$, saturation applies $\mathcal{N}_1$ to each node p at level 1 in the MDD encoding of $\mathcal{S}^{init}$, by modifying it *in place*, until it has reached a fixed-point, i.e., $\mathcal{B}(p) = \mathcal{B}(p) \cup \mathcal{N}_{\leq 1}(\mathcal{B}(p))$; then it moves to each node q at level 2 and applies $\mathcal{N}_2$ to it, and $\mathcal{N}_1$ to any node at level 1 created by this application, so that $\mathcal{B}(q) = \mathcal{B}(q) \cup \mathcal{N}_{\leq 2}(\mathcal{B}(q))$; then it moves to the nodes at level 3, and so on. Once the root r is saturated in this manner, it encodes the desired state space $\mathcal{S}$. Saturation has been shown to have memory and time requirements several orders of magnitude smaller than those of BFS in many models of GALS.

3.1 BDD vs. MDD Variables and Their Order

It is well-known that the variable order can greatly affect the size of a BDD, thus the efficiency of the symbolic iterations. Moreover, finding the optimal order that minimizes the size of a BDD (or of multiple BDDs stored in a BDD forest to share nodes) is an NP-complete problem [5]. The same applies to MDDs, of course, but, in addition, the MDD variables themselves offer a greater degree of freedom, thus more opportunities to introduce improvements, but also inefficiencies. For example, we can choose to partition the P places of a Petri net into $K \leq P$ groups, each one corresponding to a state variable. We do not address the issue of defining these groups of MDD variables, but simply observe that it can be seen as an improvement to be applied *after* having decided the order of the finest possible partition (in the case of Petri nets, this means assigning a different place to each level of the MDD, i.e., $K = P$). Thus, since we focus on the problem of finding a good order for the finest set of MDD state variables, the results that follow are applicable to BDDs as well.

3.2 Event Span Metrics

A variable ordering is a permutation π of the K state variables $(i_K, \ldots, i_1)$, so that variable i_k is assigned to level $\pi(k)$ of the MDD. In the following, we write $Top_\pi(e)$ and $Bot_\pi(e)$ to mean the value of $Top(e)$ and $Bot(e)$ when we use the permutation π. We can also envision a boolean matrix describing the dependence between levels and events, $\mathbf{A} \in \{0,1\}^{|\{K,\ldots,1\}| \times |\mathcal{E}|}$ where $\mathbf{A}(k,e) = 1$ iff $\mathcal{N}_{k,e} \neq \mathcal{I}_k$ and, for a given permutation π, let $\mathbf{A}_\pi$ be matrix obtained by permuting the rows of $\mathbf{A}$ according to π, i.e., row k of $\mathbf{A}$ equals row $\pi(k)$ of $\mathbf{A}_\pi$.

For a given variable ordering π, we define the Normalized Event Span (*NES*) metric as

$$NES(\pi) = \sum_{e \in \mathcal{E}} \frac{Top_\pi(e) - Bot_\pi(e) + 1}{K \cdot |\mathcal{E}|}$$

The *NES* metric computes the average span of all events (the span is then normalized by K) and its value is always between 0 and 1. A low *NES* indicates that the event spans are small, i.e., that most events affect only state variables close to each other in the order π.

We generalize this concept by introducing the Weighted Event Span metric of moment i, $WES^{(i)}$ for variable ordering π as:

$$WES^{(i)}(\pi) = \sum_{e \in \mathcal{E}} \left(\frac{Top_\pi(e)}{K/2} \right)^i \cdot \frac{Top_\pi(e) - Bot_\pi(e) + 1}{K \cdot |\mathcal{E}|}$$

We observe that $WES^{(0)}$ is exactly equivalent to *NES*. The $WES^{(1)}$ metric, instead, adds to it a component that reflects the *location* of the affected region, by assigning higher weights to locations closer to the top. This takes into account that operations applied to nodes in the lower portion of the MDD tend to have lower cost than those applied to higher nodes. Therefore the span of an event is scaled by $\alpha_\pi(e) = \frac{Top_\pi(e)}{K/2}$, the relative position of the topmost level compared to the average level, $K/2$. The weight of an event is thus between $(2/K)^i$ and 2^i, but the average over all events, if their tops were uniformly distributed over the MDD, should have an expected value of 1 for $WES^{(1)}$, like for *NES*. For larger moments i, the emphasis on the location grows, as the weight multiplies in powers of 2, while strong clustering is relatively less important.

The Normalized Average Lifetime (*NAL*) metric introduced in [22] is very similar to our *NES*, but it is employed in a different context: that of finding a good ordering of the conjuncts in the transition relation expression when performing symbolic image computations. In essence, the target in [22] is still to minimize the average span of rows, but computed on the transpose of our dependence matrix. Therefore, the object of optimizing *NAL* can be ultimately viewed as clustering *events* (the rows in our matrix), as opposed to variables (the columns).

3.3 NP-Completeness of Our Metric

Intuitively, our $WES^{(i)}$ metric arises from two components, the size of the span for each event, and the (i^{th} power of the) position of the span for each event.

Given our matrix $\mathbf{A} \in \{0,1\}^{|\{K,\ldots,1\}|\times|\mathcal{E}|}$ and considering all the matrices $\mathbf{A}_\pi$ obtained by permuting its rows according to π, the question (*SUM-OF-SPANS*, i.e., *NES*)

$$\text{“Is there an } \mathbf{A}_\pi \text{ s.t. } \sum_{e\in\mathcal{E}} \frac{Top_\pi(e) - Bot_\pi(e) + 1}{K\cdot|\mathcal{E}|} \leq T\text{”},$$

was proven in [7] to be NP-complete by reducing the *directed optimal linear arrangement* problem (GT43 in [16]) to it.

Focusing on the position of the spans alone, the question (*SUM-OF-TOPS*)

$$\text{“Is there an } \mathbf{A}_\pi \text{ s.t. } \sum_{e\in\mathcal{E}} Top_\pi(e) \leq T\text{”}$$

can also be shown to be NP-complete [26] by reducing the *interval graph completion* problem (GT35 in [16]) to it.

The corresponding question for our more general metric $WES^{(i)}$,

$$\text{“Is there an } \mathbf{A}_\pi \text{ s.t. } \sum_{e\in\mathcal{E}} \left(\frac{Top_\pi(e)}{K/2}\right)^i \cdot \frac{Top_\pi(e) - Bot_\pi(e) + 1}{K\cdot|\mathcal{E}|} \leq T\text{”}$$

is clearly solvable in non-deterministic polynomial time, by simply evaluating the metric for each non-deterministically chosen permutation π, but, while we strongly suspect that it is NP-complete, just like *SUM-OF-SPANS* and *SUM-OF-TOPS*, we have not yet been able to prove its NP-hardness so far. The major obstacle in achieving the completeness result is posed by the non-linearity of the target function for higher moments.

4 Results

Extensive tests were performed to shed light on the properties of the *WES* metrics. We attempted as many exhaustive experiments as we could afford, given that the number of runs required to determine the optimal ordering can be huge.

4.1 Methodology

We designed three set of experiments, which we ran on a 2.4GHz Linux workstation with 1GB of memory. Our goal is to look for a connection between optimizing one of the metrics, i.e., finding the variable ordering that results in the smallest value for the metric, and optimizing the MDD performance, i.e., having the smallest peak number of MDD nodes during state-space generation. Since the runtime and memory consumption are strongly related for MDD-based algorithms, we can restrict ourselves to peak memory as a measure of the overall performance. In all the experiments, we limited ourselves to comparing the metrics for the first three moments: $\mathit{WES}^{(0)}$, $\mathit{WES}^{(1)}$, and $\mathit{WES}^{(2)}$.

Finding the optimum peak MDD size among all possible K-variable orders requires $K!$ runs. This becomes infeasible for relatively small values of K. In our first set of experiments, then, we tried this exhaustive search on a set of five random models with $K = 6$ variables (for a total of $5 \times 6! = 3600$ runs). The question we wanted to answer was whether variable orders that minimize the MDD size coincide with orders that minimize any of the WES metrics, and how often. This test was completed only for the saturation strategy, due to the enormous amount of time required to finish the same tests on BFS.

The second set of experiments considered the inverse question: which metric we should choose to minimize in order to achieve the best MDD performance. We randomly generated 900 models of different size for which the minimum value of the metric (not the state-space) for all $K!$ variable orders can be computed in a reasonable amount of time. We stopped at a maximum of $K = 10$ variables, since after evaluating the $10! = 3,628,800$ possible permutations, the optimum one is run by BFS, for each model and each $\mathit{WES}^{(i)}$, in roughly ten minutes on average. For all 900 models, the total runtime for BFS was 29 days. In contrast, the same experiments took only 16 hours when running saturation, which is more than 40 times faster.

The next value of $K = 11$ would have taken an estimated two hours per run for BFS, for a grand total of close to one year. Even so, computing the value of the *WES* functions statically for 10! orders and then generating the state space for (one of) the order(s) that minimizes each metric takes much less time than executing 6! different MDD-based state-space generation runs. While it was not the purpose of this study, it would be of interest to generate *all* distinct models of a given size K. The total number of such models is $2^{K(K-1)}$, since, in our setting, this is the number of all sub-digraphs of the complete digraph of K nodes, as described in more detail in Section 4.2 (without considering any symmetries, equivalences, or other possible reductions). At the same time, the 900 models used in this experiment are still relevant, precisely because they are randomly generated: they represent an unbiased statistical sample and offer a reasonably even coverage of all possible models.

The first two sets of experiments can only be performed on small models. The last set is instead taken from larger, more practical models. The methodology in this case is different, as generating all models or all orders is out of the question. We generated various orders, ranging from nearly optimal to nearly random, by running a basic genetic algorithm for permutations [21]. The algorithm is stopped after a varying number of generations, and the fittest chromosome (order) at the end is fed into the state-space generator. The resulting MDD size is then used to compare how the variations in the metric value relate to the MDD performance.

4.2 The Random Model Generator

A small program written in C++ generates models as bounded Petri nets to be fed to the SMART [8] tool. The user specifies the number of Petri net places (state variables) and transitions (model events), and the maximum number of tokens allowed in each place (range of each state variable). Each transition is adjacent to one input and one output arc. Hence, this technique of "filling out" the Petri net with transitions is very similar to randomly filling a directed graph with arcs between its nodes. The program rejects disconnected models, but allows sinks, traps, and deadlocks.

4.3 Experiments Where the MDD Optimum Is Known

Table 1 presents a synopsis of the results from the first set of experiments. We generate all possible 720 permutations of six variables and report the number of permutations ("per") that led to the smallest peak MDD size. The next three pairs of columns report the minimum value of each *WES* metric on these orders, and also how many of those reached the MDD optimum. For comparison, we also list, in the last three pairs of columns, the overall optimum of each metric and in how many instances this was reached.

Figure 1 illustrates the five random models used in this experiment, as directed graphs (instead of Petri nets). An arrow represents a transition that removes a token from (i.e., decreases the value of) the source place (variable) and adds it to (i.e., increments) the target place (variable). The initial value of p_0 is written inside the place p_0.

Table 1. Experiment 1: MDD optimum vs. *WES* optimums

	smallest MDD							overall metric optimum					
	per	metric value on smallest MDDs						$WES^{(0)}$		$WES^{(1)}$		$WES^{(2)}$	
		$WES^{(0)}$	per	$WES^{(1)}$	per	$WES^{(2)}$	per	min	per	min	per	min	per
model 1	10	0.444	1	0.648	1	0.977	1	0.426	2	0.623	1	0.949	1
model 2	10	0.407	1	0.574	1	0.819	1	0.407	2	0.574	1	0.819	1
model 3	2	0.438	1	0.674	1	1.111	1	0.396	4	0.542	1	0.806	1
model 4	4	0.467	4	0.689	2	1.081	2	0.467	16	0.689	4	1.081	2
model 5	4	0.467	2	0.661	1	1.046	1	0.450	4	0.661	1	1.046	1

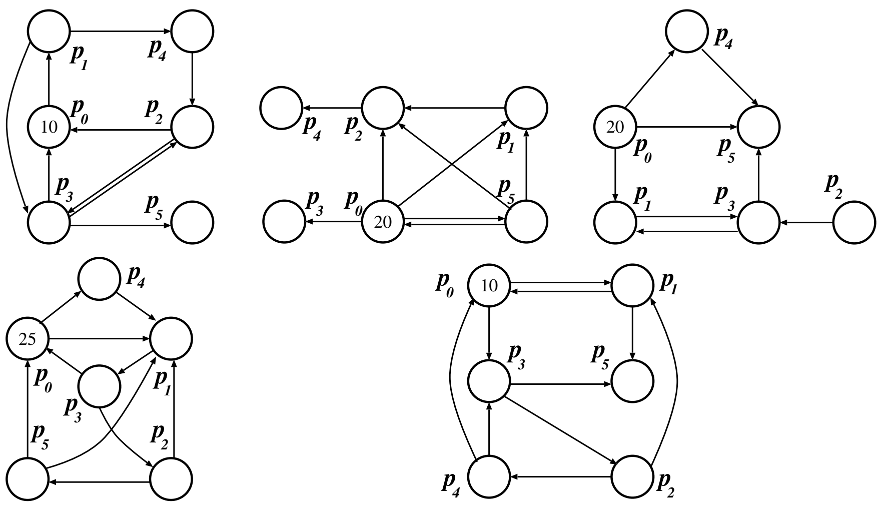


Fig. 1. The five random models used in the first set of experiments

Note that the minimums for the metrics differ substantially, depending on the moment. This is because our assumption about the expected value of the scalars $\alpha_\pi(e)$ was imprecise. The average on the top level of all events is actually higher than $K/2$ as the width of the affected region pushes this value up. Similarly, the average bottom level would sit lower than $K/2$. To achieve a better common ground when comparing the metrics, we should consider the middle level as a scalar:

$$\alpha_\pi(e) = \frac{(Top_\pi(e) + Bot_\pi(e))/2}{K/2}$$

However, that would be a completely different metric, which will not capture the effect we were targeting: the top level is most important, because that is where the recursive calls in MDD operations start. Therefore, we will forgo the property of having common expected values for the metrics in our present study.

The results show that in two of the five models (1 and 3), none of the metrics' optimums led to a MDD optimum. In two other models (2 and 4), all metrics reach the MDD optimum. However, for the $WES^{(0)}$ metric, there are multiple orderings that have minimum value and only a fraction of them are also among those that coincide with the smallest MDD (1/2 and 4/16). For the other two metrics this proportion is better (1/1 and 2/4). For the last model, $WES^{(0)}$ does not reach the MDD optimum, while the others do, and they do so for a single value. Moreover, we observe that overall (and this trend continues in the next batch of experiments) $WES^{(0)}$ has multiple minimums, making it difficult to choose the particular order among them that might lead to the MDD optimum. With the other metrics, the number of minimums is much smaller, thus the selection has a greater chance to succeed in matching the MDD's best. Most encouraging are cases such as models 2 and 5, where $WES^{(1)}$ has a *unique* minimum which *coincides* with the MDD optimum.

4.4 Experiments Where the *WES* Optimums Are Known

This set of experiments used three parameters for generating the random nets:

- number of variables P: from 8 to 10;
- number of transitions T: from 11 to 25;
- number of tokens in the initial marking (i.e., the range of variables): from 5 to 100, in increments of 5.

for a total of 900 cases.

In many instances, the best order was the same for two, or even all three metrics. A synopsis of the results is presented in Figure 2, as the percentage of runs where each metric performed best among the three (there are many ties, thus the sum of the three plots is over 100%). While for BFS the choice of metric does not appear to have a large impact, this is not the case for saturation. The table in the left of Figure 3 (left) presents a digest of the results for saturation, where it can be seen that $WES^{(1)}$ clearly performs the best.

At first glance, $WES^{(1)}$ is the best choice among the three metrics. It also appears that models that are not "dense" with transitions favor higher moment metrics ($WES^{(1)}$ and $WES^{(2)}$). Since $WES^{(1)}$ seems to be consistently slightly better than $WES^{(2)}$, it is then interesting to examine when it also "beats" $WES^{(0)}$. Figure 3 (right) shows the percentage of runs where $WES^{(1)}$ is at least as good as $WES^{(0)}$, as a function of T, for the choices $P = 8$ and $P = 10$. The overall percentage is 79%.

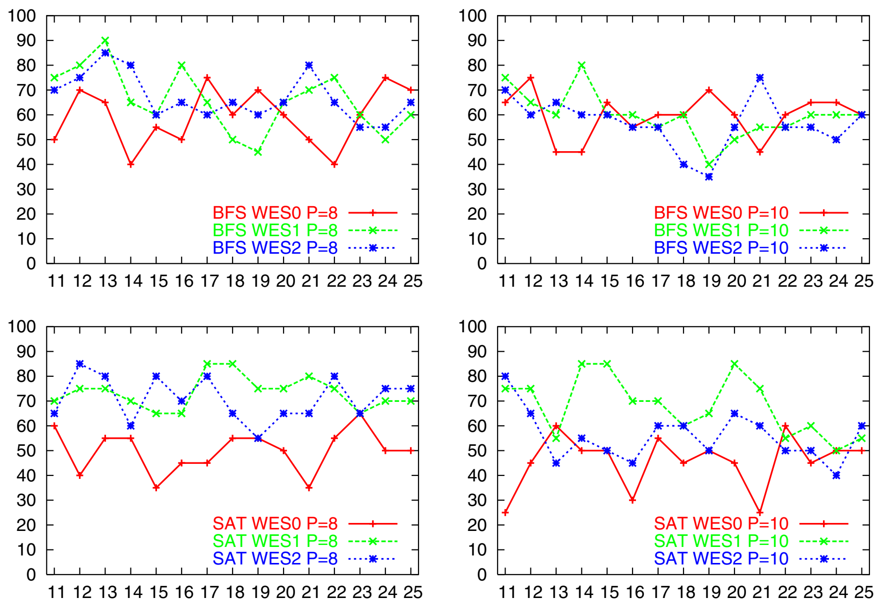
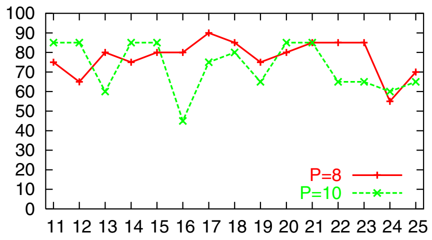


Fig. 2. Experiment 2: % of runs where $WES^{(0)}$, $WES^{(1)}$, or $WES^{(2)}$ is best, as a function of T (x-axis) and P, for BFS (top) and saturation (bottom)

	$WES^{(0)}$	$WES^{(1)}$	$WES^{(2)}$
$P = 8, T = 15$	35%	80%	65%
$P = 8, T = 20$	50%	75%	65%
$P = 8, T = 25$	50%	70%	60%
$P = 8$, total	51%	73%	70%
$P = 10, T = 15$	50%	85%	50%
$P = 10, T = 20$	45%	85%	65%
$P = 10, T = 25$	50%	55%	60%
$P = 10$, total	48%	66%	55%

% of runs where each metric is best (for saturation)



% of runs where $WES^{(1)}$ beats $WES^{(0)}$ as a function of the number of events

Fig. 3. Experiment 2 results: focus on saturation

4.5 Experiments Where Optimums Are Not Known

The collection of experimental results presented here is trying to answer the question: "Is $WES^{(1)}$ more appropriate than $WES^{(0)}$, i.e., *NES*, to evaluate good variable orderings in large models?". We compare the effect of *NES* and $WES^{(1)}$ in generating the state space with the two algorithms, saturation and BFS, and measure the runtime and peak number of nodes in the MDD (final number of nodes, as well as peak and final memory consumption are also collected, but not shown in the graphs for conciseness). The experiments are set up in SMART for three models: dining philosophers of size 10, slotted ring with 6 slots, and round robin mutex with 8 processes (see [11] for a description of these models).

A genetic algorithm computes the variable order and evaluates the metrics *NES* and $WES^{(1)}$ (as the actual fitness function for the chromosomes). To cover as many values of the metrics as possible, the genetic algorithm is run for a limited number of generations, before it converges to a good solution. As the convergence happens relatively fast, the genetic algorithm is stopped after at least 10 and no more than 1000 generations. The population size is also varied from 10 to 100 chromosomes.

We stress that a single data point in the scatter plots corresponds to one run of SMART, which can take up to an hour (the script aborts a run if the one hour timeout has expired). To increase the density of the data points in the top-right corners of the graphs would require months, spent running bad orders, so more data in those sections is hard to come by.

The scatter plots in Figure 4 reveal a few interesting facts. First and foremost, static variable ordering based on event span works: as a trend, higher values of the metric tend to correspond to larger peak number of nodes. However, there is some dispersion for both metrics, showing that neither metric is completely accurate in predicting the effect of a particular order. For example, with a variable order of *NES* 0.35 the state space of the slotted ring model can be built with BFS as fast as in 80 seconds, but also as slow as in 550 seconds. Bad orderings can have low *NES*, and also a good *NES* can yield poor results. An important question is then: are the $WES^{(1)}$ plots less scattered? The answer is yes, even if not impressively so. Nonetheless, a conclusion is that, using the $WES^{(1)}$ metric

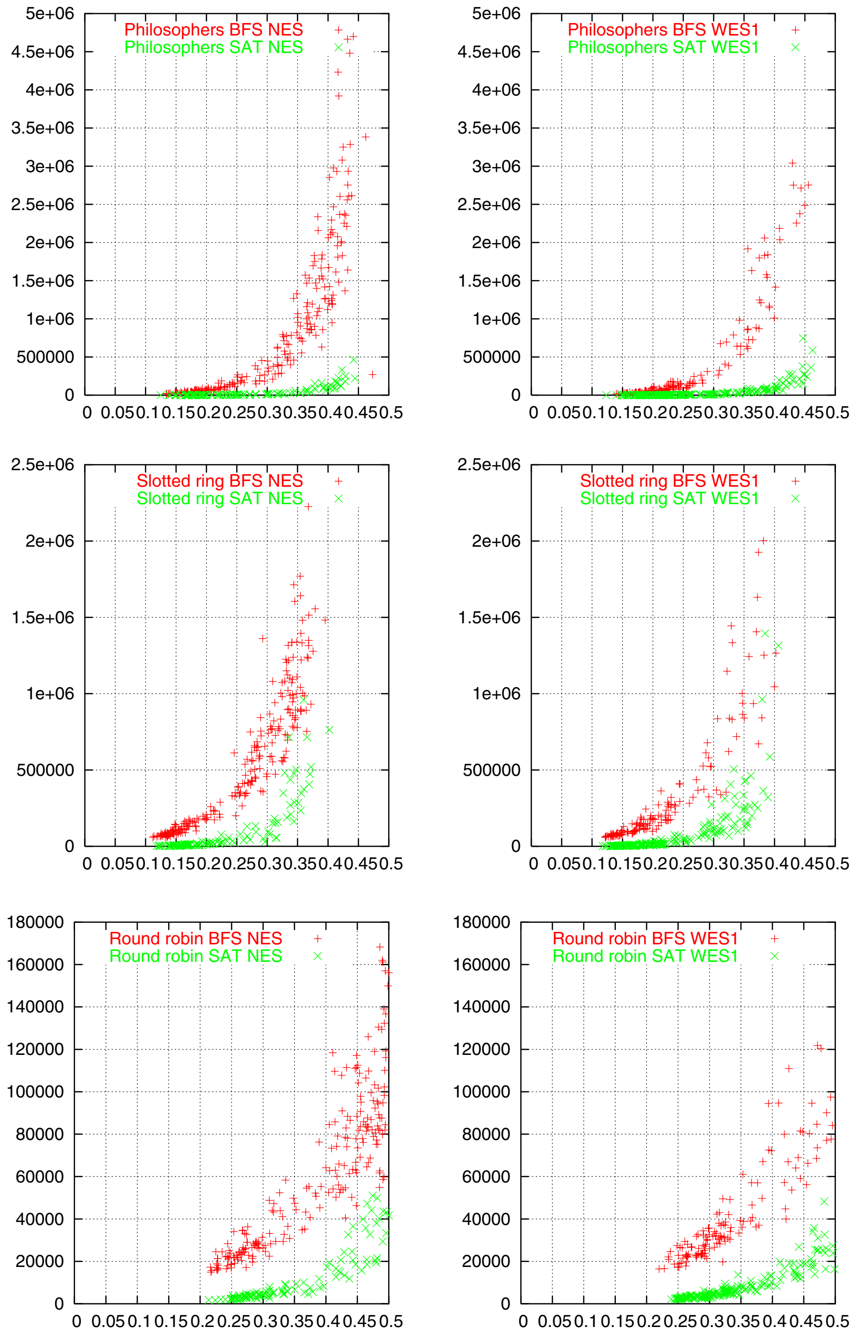


Fig. 4. Peak no. of nodes as a function of the $NES/WES^{(1)}$ metric: BFS vs. saturation

and given an input variable order of fitness 0.35, it is at least guaranteed that state-space generation will take less than 350 seconds for this model. In conjunction to this, we recall that the $WES^{(1)}$ minimums have the tendency to be up to 80% larger than *NES*'s, so the above statement is actually conservative.

Beside the connection between the metric values and the MDD size, the scatter plots also reveal a clear separation between the peak size when using BFS and saturation. Most importantly, for near optimal values the performance of saturation is consistently better. In conclusion, we can safely state that investing time in optimizing the $WES^{(1)}$ metric will result in lower runtime and memory consumption for MDD state-space generation, and that $WES^{(1)}$ is better suited for saturation than for BFS.

Finally, a remark about the genetic algorithm employed here is due. As prompted by [7], simulated annealing (essentially a degenerate form of genetic optimization) was found to be faster at finding the global optimum of certain fitness functions. We used this information to circumvent the need to compare genetic optimization with other heuristics. A more comprehensive study on this issue is due in the near future.

5 Conclusions and Future Work

We introduced a new family of metrics $WES^{(i)}$, indexed by a moment i, to be used as a guide for static variable ordering in symbolic methods. We provided sufficient evidence that the connection between minimizing $WES^{(1)}$ and minimizing the peak MDDs size in symbolic state-space generation is stronger than for the unweighted metric $WES^{(0)}$. We attribute this to the fact that the weighted metrics incorporate more specific information about the model, by rewarding what is considered a good placement for the state variables affected by an event, in addition to only a compact clustering of interdependent state variables. Another clear advantage of the metrics of higher moment is that they tend to have fewer minimums than $WES^{(0)}$. We designed extensive experiments to analyze the properties of the new metrics, including exhaustive searches for the best variable orders in small models. To the best of our knowledge, this brute-force approach had not been attempted before, yet it clearly can provide very useful insight. We have also attested once more that the saturation algorithm is vastly superior to breadth-first search, and, quite interestingly, it benefits even more from adopting the metric $WES^{(1)}$ for its variable ordering.

For future research, one question is whether there is room for more fine-tuning of the metrics or more "creative" ways to choose the scalars $\alpha_\pi(e)$. An open alternative is to scale the weights not by the index of the highest level in the decision diagram, but by some middle value so that the the expected average of the weights is 1. Of great interest would also be an exhaustive search of all models of a given size, even if such an endeavour obviously has enormous computational costs. This might enable us to classify the models into classes that are best suited to a specific choice of metric. Where exhaustive searches are not possible, data of statistical nature should be collected from more extensive experiments. The behavior of the metrics near the optimums for the metrics, and how this behavior relates to the minimization of the peak MDD size should be considered. From the algorithmic standpoint, heuristics to minimize the metrics, other than genetic optimization, and approximation methods should be studied and compared.

References

1. A. Aziz, S. Tasiran, and R.K. Brayton. BDD Variable Ordering for Interacting Finite State Machines. In *31st ACM/IEEE Design Automation Conference (DAC)*, San Diego, CA, June 1994. San Diego Convention Center. ch. 18.3.
2. F. A. Aloul, I. L. Markov, and K. A. Sakallah. MINCE: A static global variable-ordering heuristic for SAT search and BDD manipulation. *J. UCS*, 10(12):1562–1596, 2004.
3. D. Borrione and J. Vidal. Improving static ordering of BDDs for reachability analysis, Apr. 29 2002.
4. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, Aug. 1986.
5. R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comp. Surv.*, 24(3):293–318, 1992.
6. J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic model checking with partitioned transistion relations. In *VLSI*, pages 49–58, 1991.
7. P. Chauhan, E. Clarke, S. Jha, J. Kukula, H. Veith, and D. Wang. Using combinatorial optimization methods for quantification scheduling. *Lecture Notes in Computer Science*, 2144:293–302, 2001.
8. G. Ciardo, R. L. Jones, A. S. Miner, and R. Siminiceaunu. Logical and stochastic modeling with SMART. In *Proc. Modelling Techniques and Tools for Computer Performance Evaluation*, LNCS 2794, pages 78–97, Urbana, IL, USA, Sept. 2003. Springer-Verlag.
9. G. Ciardo, G. Lüttgen, and R. Siminiceanu. Efficient symbolic state-space construction for asynchronous systems. In *Proc. 21th Int. Conf. on Applications and Theory of Petri Nets*, LNCS 1825, pages 103–122, Aarhus, Denmark, June 2000. Springer-Verlag.
10. G. Ciardo, G. Lüttgen, and R. Siminiceanu. Saturation: An efficient iteration strategy for symbolic state space generation. In T. Margaria and W. Yi, editors, *Proc. TACAS*, LNCS 2031, pages 328–342, Genova, Italy, Apr. 2001. Springer-Verlag.
11. G. Ciardo, R. Marmorstein, and R. Siminiceanu. Saturation unbound. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS 2619, pages 379–393, Warsaw, Poland, Apr. 2003. Springer-Verlag.
12. G. Ciardo and J. Yu. Saturation-based symbolic reachability analysis using conjunctive and disjunctive partitioning. In *Proc. CHARME*, Saarbrücken, Germany, Oct. 2005. Springer-Verlag. To appear.
13. D. Geist and I. Beer. Efficient model checking by automated ordering of transition relation. In David L. Dill, editor, *Proceedings of the sixth International Conference on Computer-Aided Verification CAV*, volume 818, pages 299–310, Standford, California, USA, 1994. Springer-Verlag.
14. R. Drechsler, B. Becker, and N. Gockel. A genetic algorithm for variable ordering of OBDDs. In *Int'l Workshop on Logic Synthesis*. ACM/IEEE, May 1995.

15. M. Fujita, H. Fujisawa, and Y. Matsunaga. Variable ordering algorithms for ordered binary decision diagrams and their evaluation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(1):6–12, Jan. 1993.
16. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman Press, 1979.
17. O. Grumberg, S. Livne, and S. Markovitch. Learning to order BDD variables in verification. *Journal of Artificial Intelligence Research*, 18:83–116, 2003.
18. W. N. N. Hung and X. Song. BDD variable ordering by scatter search. In *ICCD*, pages 368–373, 2001.
19. T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli. Multi-valued decision diagrams: theory and applications. *Multiple-Valued Logic*, 4(1–2):9–62, 1998.
20. Y. Lu, J. Jain, E. M. Clarke, and M. Fujita. Efficient variable ordering using a BDD based sampling. In *Design Automation Conference*, pages 687–692, 2000.
21. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, New York, NY, USA, 1996.
22. I.-H. Moon, J. H. Kukula, K. Ravi, and F. Somenzi. To split or to conjoin: the question in image computation. In *Proceedings of the 37th Conference on Design Automation (DAC-00)*, pages 23–28, NY, June 5–9 2000. ACM/IEEE.
23. A. M. Moreira, D. Déharbe, and U. S. Costa. Advances in BDD reduction using parallel genetic algorithms, May 2001.
24. R. Rudell. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In *IEEE /ACM International Conference on CAD*, pages 42–47, Santa Clara, California, Nov. 1993. ACM/IEEE, IEEE Computer Society Press.
25. R. Ranjan, A. Aziz, R. Brayton, B. Plessier, and C. Pixley. Efficient BDD algorithms for FSM synthesis and verification, May 1995.
26. Y. Wu and J. Robert. Personal communication, Oct. 2005.