DSM-based variable ordering heuristic for reduced computational effort of symbolic supervisor synthesis

DSM-based variable ordering heuristic for reduced computational effort of symbolic supervisor synthesis

Sam Lousberg [*] **Sander Thuijsman** [*] **Michel Reniers** [*]

[*] *Eindhoven University of Technology, Department of Mechanical Engineering, Eindhoven, the Netherlands (e-mail: samlousberg@live.nl and {s.b.thuijsman, m.a.reniers}@tue.nl).*

Abstract: We consider the influence that the variable order of Binary Decision Diagrams (BDDs) has on the computational effort that is required for symbolic supervisor synthesis. In recent research it has been shown that improving the variable order can result in a substantial decrease of synthesis effort. We propose the combined use of the Dependency Structure Matrices (DSMs) and the matrix reordering heuristics Cuthill-McKee and Sloan to minimize the Weighted Event Span (WES) of the variable order. This is done by placing variables that often appear together in transition relations near each other. By performing benchmark experiments, we measure the reduction in synthesis effort by utilizing a variable order with minimized WES. The experiments show that our approach is competitive in reducing computational effort compared to FORCE, a state of practice variable ordering heuristic. Moreover, the best improvements in effort reduction are shown for the most computationally demanding models tested.

Keywords: Binary decision diagrams, Control system synthesis, Heuristic algorithms, Supervisory control, Variable order

1. INTRODUCTION

Supervisory Control Theory (SCT) as introduced by Ramadge and Wonham (1989) is a model-based approach to control Discrete Event Systems. In the framework of SCT we compute a supervisor that is safe (unsafe or undesirable states are not reachable), non-blocking (marked states are reachable), controllable (only controllable events are disabled), and maximally permissive (restrictions are minimal with regard to the three aforementioned criteria). The supervisor is synthesized from plants (models of the uncontrolled system) and requirements (specifications of allowed or desired behavior).

Despite the fact that SCT has been successfully applied to some examples of industrial size, e.g., for a magnetic resonance imaging scanner (Theunissen et al., 2014) and a waterway lock (Reijnen et al., 2017), industrial acceptance is still scarce. This is mainly a result of the computational complexity of synthesis, the exponential state-space explosion that occurs is a limiting factor (Wonham et al., 2018). In case the plants and requirements are provided by means of Extended Finite Automata (EFAs) (Skoldstam et al., 2007), they can symbolically be expressed by Binary Decision Diagrams (BDDs) (Miremadi et al., 2012). Synthesis can be applied directly to these BDD representations. Essential to utilizing BDDs is finding an efficient *variable order*. This order has an enormous influence on the amount of computation time and computer memory required (Minato, 1996; Somenzi, 1999). Computing the most efficient order is NP-complete as shown by Bollig and Wegener (1996) and therefore heuristic algorithms are often used to find an acceptable order.

The importance of the variable order has been well described in literature. More specifically, in recent research Thuijsman et al. (2019) have shown the impact of BDD variable order on symbolic supervisor synthesis. Even after applying FORCE (Aloul et al., 2003), which is a variable ordering heuristic that can be regarded as state of the art as shown by Meijer and van de Pol (2016), they show that the computational effort can still further be reduced by order of magnitude.

This paper proposes a heuristic algorithm named DSM-based Cuthill-McKee-Sloan variable ordering Heuristic (DCSH) to find a variable order that further reduces the computational effort required for symbolic supervisor synthesis compared to current implementations. This heuristic is based on two matrix ordering heuristics (Cuthill and McKee, 1969; Sloan, 1989) that are used to minimize the Weighted Event Span (WES) metric (Siminiceanu and Ciardo, 2006). These matrix reordering heuristics are applied to a Dependency Structure Matrix (DSM) that stores the number of times BDD-variables appear together in transition relations, to find a new variable order. Meijer and van de Pol (2016) have shown that these heuristics are able to reduce the WES, and thereby the computational effort for symbolic model checking. Directly minimizing the WES has been shown to be NP-complete by Siminiceanu and Ciardo (2006).

[⋆] Research leading to these results has received funding from the EU ECSEL Joint Undertaking under grant agreement n° 826452 (project Arrowhead Tools) and from the partners national programs/funding authorities.

Utilizing a DSM for computing a variable order for symbolic supervisor synthesis is the first novel contribution of this paper. The second contribution is an analysis of synthesis in which we point out why applying these heuristics results in less synthesis effort. DCSH is compared to FORCE in benchmark experiments to measure the effort reduction that is achieved. Moreover, experiments are conducted where DCSH and FORCE are used in sequence to compute a variable order applied to symbolic synthesis.

In Section 2 preliminary knowledge of BDD-based supervisor synthesis and related subjects are elaborated upon. Section 3 shows an analysis of the backwards reachability search, which leads to the proposal of the new variable ordering heuristic found in Section 4. This heuristic is compared to FORCE in benchmark experiments as shown in Section 5. Finally, conclusions are found in Section 6.

2. PRELIMINARIES

2.1 Extended Finite Automata

An EFA is a 7-tuple $(L, V, \Sigma, E, L_m, L_0, V_0)$ with finite set of locations L, bounded domain of discrete variables $V = V^1 \times \ldots \times V^n$ where n is the number of variables, set of events Σ, set of edges E, set of marked locations $L_m \subseteq L$, set of initial locations $L_0 \subseteq L$, and set of initial variable valuations $V_0 = V_0^1 \times \ldots \times V_0^n$. Each edge $e \in E$ is a 5-tuple $e = (o_e, t_e, \sigma_e, g_e, u_e)$ with origin location $o_e \in L$, terminal location $t_e \in L$, event label $\sigma_e \in \Sigma$, guard $g_e : V \to \{\mathit{true}, \mathit{false}\}$, and variable update function $u_e : V \to V$ (Ouedraogo et al., 2011).

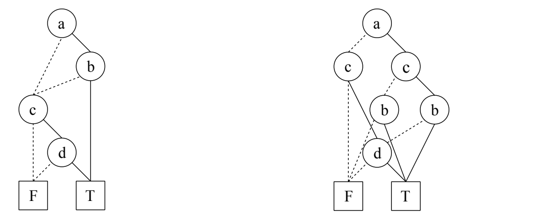
The set of events Σ is divided into two disjoint subsets Σ_c and Σ_u, respectively the set of controllable and uncontrollable events. The guards of controllable events can be adapted by synthesis, opposed to the guards of uncontrollable events. By referring to controllable or uncontrollable *edges*, we refer to whether σ_e is controllable or uncontrollable for $e \in E$. An edge $e \in E$ from o_e to t_e with g_e, σ_e, and u_e can only be taken if g_e evaluates to *true* for the current variable valuation. After the transition, the variable valuation is updated according to u_e. The *state* of an EFA is a valuation from $L \times V$.

2.2 Binary Decision Diagrams

An EFA can symbolically be described by Boolean expressions (Miremadi et al., 2012). Any Boolean expression f containing Boolean variables $b \in B$, can be expressed as

$$(b \wedge f|_{b=true}) \vee (\neg b \wedge f|_{b=false}) \,. \tag{1}$$

In this notation $f|_{b=true}$ denotes an assignment of *true* to b in expression f. By recursively applying (1) for all $b \in B$, a Binary Decision Diagram (BDD) (Akers, 1978) can be constructed to express f. BDDs are directed acyclic graphs that consist out of two types of nodes: decision- and terminal nodes. Each decision node is labeled by a Boolean variable $b \in B$ and has two edges leading to child nodes, one edge labeled *true* and the other *false*. When evaluating b to *true* or *false* we take the respective edge. Visually we represent a *true* (*false*) edge by a solid (dashed) line. At the leaves of the BDD are terminal nodes, that are labeled by *true* or *false*. When referring to BDDs in this paper, we indicate reduced ordered BDDs (Bryant, 1992).



(a) Variable order $a < b < c < d$ (b) Variable order $a < c < b < d$

Fig. 1. Two variable orders for $f : (a \wedge b) \vee (c \wedge d) = \mathit{true}$.

This type of BDD imposes some additional restrictions such that the BDD is minimal in the number of decision nodes and canonical for a given order of the variables. This order is strictly imposed over all the variables in the BDD and is called the *variable order*. A variable order is denoted as $<$, where $b_1 < b_2$ indicates that decision node b_1 is placed closer to the root node than b_2. Furthermore, there are only two terminal nodes in the BDD, one labeled *true* and the other *false*. The variable order can have a major influence on the number of decision nodes required to encode a Boolean expression, see Fig. 1 for an example. The size of a BDD is defined by the number of decision nodes and in worst-case this size can be exponential in the number of Boolean variables (Bryant, 1992).

The BDD size affects the amount of computer memory required. Furthermore, it also has a major effect on the computation time of synthesis. Common BDD operations are those of computing the conjunction (`And` operation) and the disjunction (`Or` operation) of two BDDs. These operations are both based on the recursive expansion following from (1) of BDDs f and g with operation **op** for variable b:

$$\begin{aligned} f \ \mathbf{op} \ g = {} & b \wedge (f|_{b=true} \ \mathbf{op} \ g|_{b=true}) \vee \\ & \neg b \wedge (f|_{b=false} \ \mathbf{op} \ g|_{b=false}) \,. \end{aligned} \tag{2}$$

The two sub-operations are recursively expanded according to the equation shown above, starting from the top node(s), as imposed by the variable order. This is repeated until all recursive operations lead to terminal cases (Somenzi, 1999). Applying operations to smaller BDDs commonly results in fewer recursive operations and thereby to reduced computation time. Next to the `And` and `Or` operations, the existential quantification is an operation that can be applied to a BDD (Bryant, 1992). This operation is applied during the reachability search that is required to compute a supervisor.

2.3 Compositional Interchange Format

The Compositional Interchange Format (CIF) (van Beek et al., 2014) and Supremica (Malik et al., 2017) both allow for BDD-based symbolic supervisor synthesis of EFAs. Moreover, both modeling tools use FORCE as variable ordering heuristic to find a variable order before synthesis commences that often performs better compared to a random order. FORCE optimizes the *span* of highly related variables (Aloul et al., 2003). A global optimization is performed by repeatedly placing highly related variables closer to each other, until the span does not reduce anymore. Afterwards, a window is slid over the order produced by FORCE where the variables within the window are reordered locally according to the same placement criteria as the global order. In this paper we refer to applying FORCE as applying first the global and subsequently the local (window-based) reordering. In this paper all experiments are conducted using CIF.

2.4 Metrics for computational effort

We express the computational effort required for synthesis by two BDD-based metrics: peak used BDD nodes and BDD operation count (Thuijsman et al., 2019). The first metric is the peak size of all BDDs combined during synthesis. As computer memory is always finite, this is the main limiting factor for successful synthesis. The latter is the number of times a recursive call is made to any BDD operation and mainly influences the computation time. These metrics allow to measure the required computational effort in a deterministic, platform-independent way and include no overhead in their measurements, opposed to more traditional metrics such as computer memory usage and wall-clock time. Performing synthesis with constant parameter settings results in the same measurement each time for these BDD-based metrics.

3. THE BACKWARDS REACHABILITY SEARCH

Symbolic supervisor synthesis as applied in this paper requires a backwards reachability search. By analysis of this search we show which characteristics of the variable order influence the size of intermediate BDDs and to what extent this contributes to the total synthesis effort. The computations of the non-blocking N and bad-state B predicates are essential to this process. The non-blocking predicate expresses what locations and discrete variable values can be reached backwards from the marked states, with respect to the current guards and updates. The bad-state predicate expresses what locations and discrete variable values lead to undesirable states (Ouedraogo et al., 2011). These are states that can lead to blocking states by a sequence of uncontrollable events.

We utilize current-state variables $x \in X$ that specify the state of the system, i.e., specify all locations of the automata and all variable valuations, before a transition is taken. Next-state variables $x^+ \in X^+$ specify the state of the system after a transition is taken. Therefore, the BDD of each update u_e is expressed in both current- and next-state variables (the update adapts the state after a transition based on the state before a transition). The BDD of each guard g_e is only expressed in current-state variables (the evaluation of the guard only depends on the state before the transition). During the search one transition at a time is added to the predicates. This is described by the transition relation $T_e(X, X^+) = g_e(X) \wedge u_e(X, X^+)$; each edge $e \in E$ has a single $T_e(X, X^+)$.

Fig. 2 shows a global overview of the steps that are required for symbolic synthesis. Before synthesis, all plants and requirements are combined by linearization such that a single EFA results, see Fig. 3 for an example. For a more thorough elaboration we refer to Nadales Agut and Reniers (2011). The initial non-blocking predicate $N_0(X)$ is set equal to the marked state predicate $N_m(X)$ where locations $l \in L_m$ are set to *true*, otherwise to *false*. To compute the non-blocking predicate, recursively apply

$$N_k(X) = N_{k-1}(X) \vee \bigvee_{e \in E} \exists_{X^+}\big[N_{k-1}(X^+) \wedge T_e(X, X^+)\big], \quad (3)$$

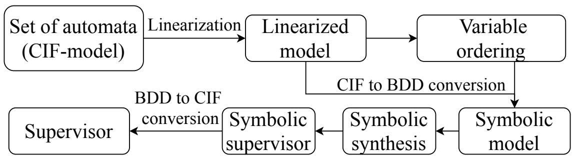


Fig. 2. Flowchart of symbolic synthesis.

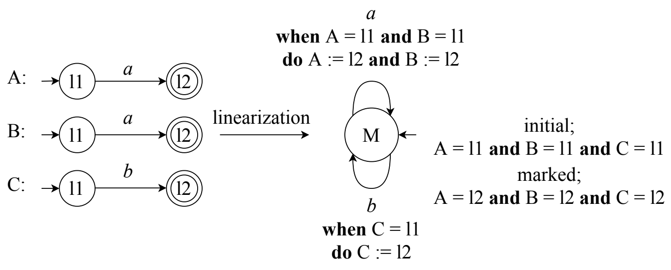


Fig. 3. EFA M that results from the linearization of three automata A, B and C.

until the fixed-point $N_k(X) = N_{k-1}(X)$ is reached. Next, the BDD of the initial bad-state predicate is set equal to $B_0(X) = \neg N_k(X)$ and a similar fixed-point computation, now with only uncontrollable edges, is performed using

$$B_j(X) = B_{j-1}(X) \vee \bigvee_{(e \in E \mid \sigma_e \in \Sigma_u)} \exists_{X^+}\big[B_{j-1}(X^+) \wedge T_e(X, X^+)\big]. \quad (4)$$

Next, the initial non-blocking predicate is set to $N_0(X) = N_k(X) \wedge \neg B_j(X)$. The previous steps starting from $N_0(X)$ are repeated, unless $B_j(X)$ is unchanged after a new iteration. If this is the case, the backwards reachability search is finished and the predicate of the controlled system is set to $P_c(X) = N_k(X)$, where $N_k(X)$ is the most recently computed non-blocking predicate. Finally, the supervisor is computed by strengthening the guards of all controllable edges, using

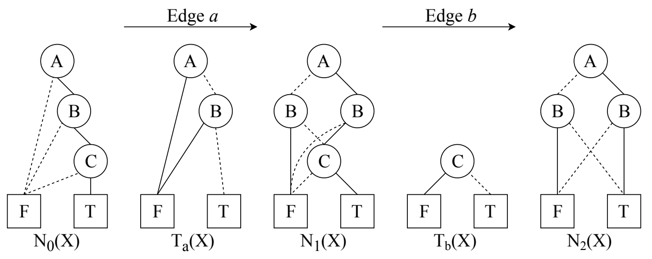
$$g_e(X) = g_e(X) \wedge \exists_{X^+}\left[P_c(X^+) \wedge T_e(X, X^+)\right]. \quad (5)$$

3.1 The relational product

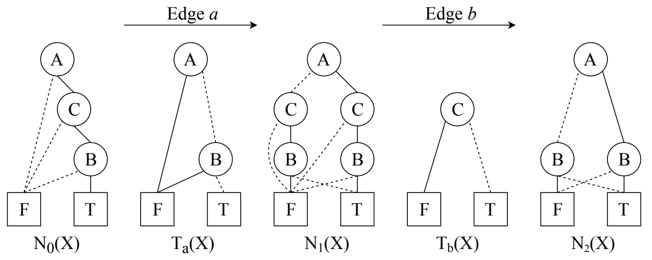
A frequently applied operation during the backwards reachability search is $\exists_v\,[f \wedge g]$, where f and g are both BDDs and v is the set of existentially quantified variables, as found in (3), (4), and (5). This operation can be executed by first computing the `And` for f and g and later quantifying over v. However, this results in a large intermediate result of $f \wedge g$. Therefore, both the conjunction and existential quantification are computed in a single recursive pass over f and g by utilizing the relational product operation (Burch et al., 1994). This operation prevents computing the entire BDD $f \wedge g$ and quantifies early over v, thereby reducing memory usage and number of required operations. The worst-case complexity of the relational product is $\mathcal{O}(|f| \times |g| \times 2^{2|b|})$, where $|f|$ and $|g|$ are the sizes of the BDDs and $|b|$ is the total number of BDD-variables in the model (McMillan, 1993). Nonetheless, computing the relational product is known to be an expensive computation during the backwards reachability search (Burch et al., 1994).

3.2 Propagation of transition relations

Recall (3) and (4), notice that each computation depends on the previous result. As the number of BDD-variables n



(a) Variable order $A < B < C$.



(b) Variable order $A < C < B$.

Fig. 4. BDDs during the backwards reachability search.

and the size of each transition relation is equal throughout synthesis, we know that the BDD that determines the computational effort of the following relational product operation is the previously computed non-blocking or bad-state predicate. Thus, if $N_{k-1}(X)$ is small, the effort required for the computation of $N_k(X)$ is low. Furthermore, note that during each step of the computation of the non-blocking predicate first $\exists_{X^+} [N_{k-1}(X^+) \wedge T_e(X, X^+)]$ is computed. The same applies to the bad-state predicate. Both consist only out of current-state variables, as the next-state variables are existentially quantified. In a sense, each iteration we add assignments that are imposed by the current-state variables of the transition relation. Furthermore, we known that the variables in each transition relation are strongly related in the Boolean expressions the BDDs represent. Moreover, BDDs are overall small if strongly related BDD-variables are placed near each other (Minato, 1996; Somenzi, 1999). An example of this effect is shown in Fig. 1, where keeping BDD-variables a and b as well as c and d near each other results in a smaller BDD.

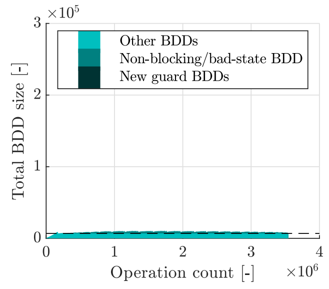
To summarize our observation, if we keep all current-state variables of each transition relation near each other, it is likely that the resulting non-blocking and bad-state predicates are small, as we know that these variables are strongly related. We denote the transition relation of current-state variables by

$$T_e(X) = \exists_{X^+} T_e(X, X^+). \tag{6}$$

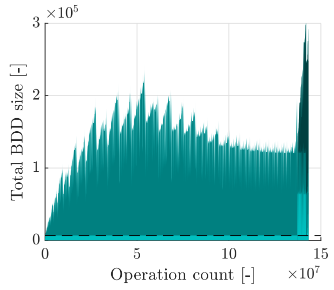
In the following section we shown an example of this effect.

3.3 Relation between transition relations, variable order and computational effort

In Fig. 4 the first two steps of synthesis are shown applied to EFA M given in Fig. 3. Note that keeping A and B strictly next to each other in the order results in the least number of decision nodes. To further analyze this behavior the same experiment is executed for a model of relevant complexity. In this experiment we measure the size of all BDDs combined, non-blocking (3) and bad-state predicates (4) and new guard predicates (5) during



(a) Order with variables appearing in $T_e(X)$ placed near each other.



(b) Order with variables appearing in $T_e(X)$ placed apart from each other.

Fig. 5. Evolution of the total BDD size for two orders with opposing characteristics.

synthesis, where two variable orders are chosen such that variables appearing in transition relations are placed near each other for the first order and far apart from each other for the second order. In Fig. 5 the results of this experiment are shown for the Cluster tool model (Su et al., 2010). Notice that the total size of the BDDs prior to the computations of the non-blocking and bad-state predicates in the backwards reachability search is relatively equal for both orders. For the first order the total BDD size only slightly increases during synthesis, for the second order the total BDD size becomes substantially larger, more so, during the computation of the new guards the total BDD size reaches its peak. For the first order this effect is hardly noticeable. As larger BDDs require more recursive calls when operations are applied to them, this also results in a large difference in operation count.

4. VARIABLE ORDERING HEURISTIC

We introduce a variable ordering heuristic to find an order where the variables often appearing together in transition relations are placed near each other. The heuristic utilizes a Dependency Structure Matrix (DSM) to store pairs of CIF-variables that appear together in $T_e(X)$. Subsequently, the DSM is manipulated utilizing two matrix reordering heuristics, resulting in several viable orders.

4.1 Dependency Structure Matrix

A DSM is a square $n \times n$ matrix representing dependencies between n aspects of a system or model (Browning, 2016). We capture variables of SCT models along the rows and columns where each index represents a single CIF-variable. In this paper we utilize static Numerical DSMs (NDSMs). The off-diagonal elements can be non-negative integers, where the value indicates the number of times the respective variables appear together in $T_e(X)$. In our use, the diagonal elements are always zero. Furthermore, all dependencies in the NDSM are regarded as undirected, thus providing a symmetric matrix. Subsequently, the NDSM is manipulated by two matrix ordering heuristic algorithms that reorder the row and column indices such that non-zero values are placed towards the diagonal. The order in which the variables appear along the rows/columns is used as variable order for synthesis. Essentially we are creating a variable order such that variables that often appear together in pairs in $T_e(X)$, are placed near each other in the variable order.

4.2 Construction of the NDSM

Before synthesis we extract the variables that appear in $T_e(X)$ for all $e \in E$, recall (6). For all occurrences of pairs of variables per $T_e(X)$ we increment the element in the NDSM by one, thus a higher value indicates a stronger dependency between the variables. The increment is executed for both combinations of the pair such that the resulting NDSM is symmetric.

The use of DSMs in SCT is not new, Goorden et al. (2017) used DSMs to find clusters of highly interactive components for the purpose of applying multilevel synthesis. However, we are not looking for clusters, but interested in reordering the row and column indices such that higher valued elements are placed as close as possible towards the diagonal relative to lower valued elements. By finding such an order, we also find an order where variables that often appear together in transition relations are placed near each other.

We utilize existing node ordering heuristics, that have been designed for bandwidth, profile, and/or wavefront reduction of symmetric sparse matrices. The three aformentioned metrics indicate the closeness of non-zero elements to the diagonal in a matrix, for an elaboration on these metrics we refer to Cuthill and McKee (1969) for bandwidth and Sloan (1989) for profile and wavefront. By minimizing any of these metrics we also achieve our desired order. We experienced that the NDSMs constructed in our approach are also sparse. The effective use of these heuristics for static variable order optimization for decision diagrams is shown in Meijer and van de Pol (2016), where several node ordering heuristics have been compared. These heuristics apply a reordering to the adjacency graph that can directly be extracted from the NDSM. As we utilize an NDSM we append the graph's edges by weights resulting in a *weighted* adjacency graph. For an NDSM with row index i and column index j, we denote elements by $e_{i,j}$. For each row i we generate a node labeled by i. Subsequently, each non-zero element $e_{i,j}$ results in an undirected edge with weight $e_{i,j}$ between nodes i and j. This results in a weighted adjacency graph where the node labels are reordered using the following two heuristics.

4.3 Weighted Cuthill-McKee ordering

The Cuthill-McKee (CM) ordering is a bandwidth reducing node ordering heuristic introduced by Cuthill and McKee (1969). The standard algorithm places non-zero elements near the diagonal to result in a matrix with a lower bandwidth. We introduce an adjustment to the standard algorithm, such that it is able to differentiate between non-zero elements. Higher valued elements are prioritized in being placed close to the diagonal over lower valued elements. We will refer to this algorithm as the *weighted* CM ordering, which is shown in Algorithm 1. Lines 5 and 6 are an adjustment of the standard algorithm found in Cuthill and McKee (1969). The algorithm starts by finding a peripheral node that is defined as a node whose shortest distance to the node furthest away is equal to the diameter of the graph. However, finding a true peripheral node is often very expensive to calculate and therefore a heuristic approach is taken to find a pseudo-peripheral node (George and Liu, 1979). Nodes can be unconnected as in not having any edges connected to any other nodes. Those nodes are stored in a list of unconnected nodes E.

Algorithm 1 Weighted Cuthill-McKee ordering

```
Input NDSM $M$
Output Order $R$
 1: Initialize empty list $R$, compute weighted adjacency
    graph $A$ of $M$ and initialize list of unconnected nodes
    $E$ in any order
 2: Compute pseudo-peripheral node $p$ of $A$
 3: Mark $p$ and append $p$ to $R$
 4: while Unmarked nodes exist in $A$ do
 5:     Find list of unmarked neighbors $C$ of $p$ and sort $C$
        in descending weight
 6:     Sort nodes in $C$ with equal weight in ascending
        degree
 7:     Append $C$ to $R$ and mark all nodes in $C$
 8:     Set the next node in $R$ as $p$
 9: end while
10: Set $R$ equal to $R$ appended to $E$
```

4.4 Sloan's ordering

Sloan's ordering is a profile and wavefront reducing node ordering heuristic introduced by Sloan (1989). It places non-zero elements near the diagonal to result in a lower profile of the matrix. In this paper the standard algorithm is not adjusted to be able to differentiate between non-zero elements, although this is of interest for future work.

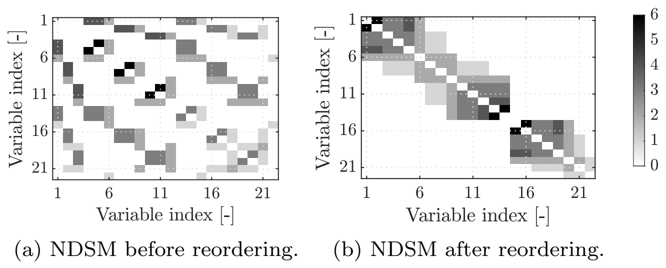
4.5 Weighted Event Span

We apply both ordering heuristics to the NDSM indicating related pairs of variables. This results in two orders, furthermore, we notice that reversing the order can sometimes lead to significant differences in synthesis effort. Siminiceanu and Ciardo (2006) noticed that placing variables that result in more costly operations towards the bottom of the BDD resulted in less effort required in a similar application of decision diagrams. This resulted in the Weighted Event Span (WES) metric. Furthermore, the WES has extensively been tested by Meijer and van de Pol (2016), where a correlation is shown between peak BDD nodes, computation time, and the WES for several types of decision diagrams applied to symbolic model checking. The WES is found by

$$\text{WES} = \sum_{e \in E} \frac{2x_b}{|x|} \cdot \frac{x_b - x_t + 1}{|x||E|} \tag{7}$$

where $|E|$ indicates the total number of edges, $|x|$ the number of current-state variables, and x_b and x_t are respectively the index of the bottom and top BDD-variable that occur in $T_e(X)$ where the index is numbered $(1, ..., |x|)$ starting at index 1 for x_t. For the computation of the WES of a given variable order, we take the bottom and top BDD-variables that appear in each $T_e(X)$ after the reordering has been applied. The first term in (7) increases as x_b is placed later in the variable order. The second term increases the WES when $x_b - x_t$ is large.

To estimate which of the four orders (two orders resulting from two different ordering heuristics and two reversed



(a) NDSM before reordering. (b) NDSM after reordering.

Fig. 6. NDSM before and after reordering for the Power substation model (Chao et al., 2017).

orders) should be applied to synthesis, the WES is computed for each of the orders. The order that has the lowest WES is applied to synthesis. This results in the proposed variable ordering heuristic, named DSM-based Cuthill-McKee-Sloan variable ordering Heuristic (DCSH), for ease of reference. Fig. 6 shows an example of an NDSM before and after reordering.

5. BENCHMARK EXPERIMENTS

To measure the effectiveness of DCSH in effort reduction, we perform two experiments for each of the eleven benchmark models shown in Table 1. For each model 10,000 random variable orders are generated and synthesis is applied using CIF for each random order. To measure the synthesis effort we extract the peak used BDD nodes and BDD operation count. This experiment is used to form a baseline of required effort when no heuristics are applied to the models. The results of the Bridge and FESTO models are excluded from this experiment as synthesis was unable to compute a supervisor for some of the random orders with 16 GB of memory allocated. Moreover, DCSH is used to generate a single order per model, this order is used for synthesis and the effort is measured.

To measure the effectiveness of DCSH compared to FORCE in synthesis effort reduction, a second experiment is conducted: the same random orders are used, however in this case used as initial order for FORCE. These results are used as a baseline. As FORCE is the state-of-the-art heuristic used for variable ordering for supervisor synthesis, this baseline is used to compare it against DCSH. In this experiment we use FORCE as it currently is implemented in CIF; FORCE performs an optimization of the span based on other dependencies compared to DCSH. The order that results from FORCE is used for synthesis. Lastly, we notice that applying FORCE as post-ordering to the order computed by DCSH can be noticeably beneficial.

Table 1. Benchmark models.

Model
Adv. Driver Assist. System (ADAS) (Korssen et al., 2018)
Power substation (Chao et al., 2017)
Theme park (Forschelen et al., 2012)
Automated Vehicle Guidance (AVG) (Wonham and Cai, 2019)
Multi Agent Formation (MAF) (Cai and Wonham, 2015)
Cluster tool (Su et al., 2010)
Ball system (Čengić and Åkesson, 2008)
Bridge (Reijnen et al., 2018b)
Production cell (Feng et al., 2009)
Waterway lock (Reijnen et al., 2017)
FESTO (Reijnen et al., 2018a)

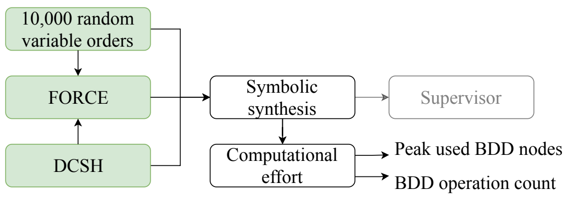


Fig. 7. Flowchart of the benchmark experiments.

Thus, we use the orders computed by DCSH as initial order for FORCE and apply the order that results to synthesis accordingly. For ease of reference, we refer to this method of applying the two heuristics in sequence as DCSH-FORCE. A flowchart of these experiments is shown in Fig. 7. For each model a normalized histogram is derived and shown along with the means, and results of DCSH and DCSH-FORCE in Fig. 8. For each model two histograms derived from the 10,000 measurements are shown per metric, one histogram for applying FORCE and one for no heuristics applied. The markers indicate the resulting effort from applying DCSH and DCSH-FORCE as well as the mean of the randomized measurements. The solid black lines indicate the best-case (bottom) and worst-case (top) effort as measured using the 10,000 randoms orders.

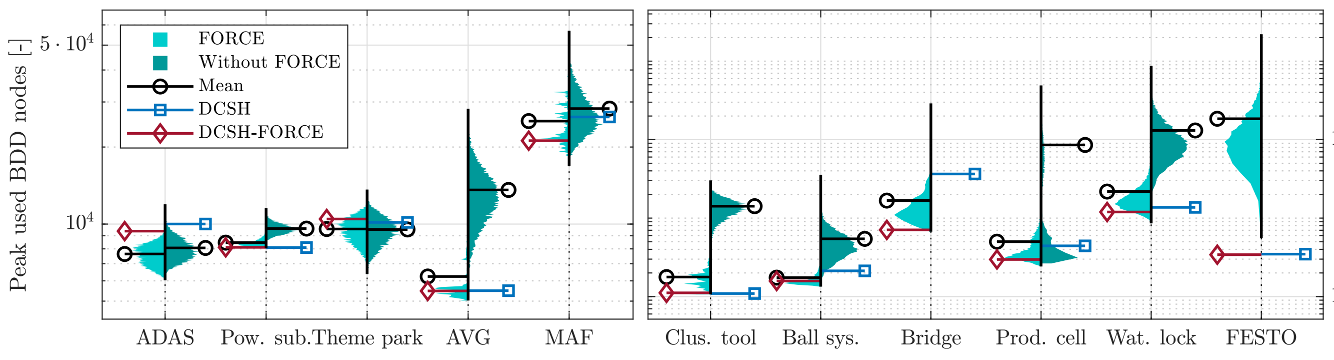
It can be seen from the means in Fig. 8 that FORCE reduced average effort for all models except the Theme park model. By applying DCSH the effort reduces compared to no heuristics applied for all models except the Theme park and ADAS models. Applying DCSH-FORCE shows a further reduction of effort compared to DCSH for most models. If the effort increased, this is only marginal compared to the reduction that is achieved for other models.

To quantify the effort reduction of DCSH and DCSH-FORCE against FORCE, we compute the number of random initial variable orders n_{ran} used for FORCE that resulted in less effort compared to DCSH or DCSH-FORCE, and compute its fraction out of all measurements n_{tot} by $f = n_{ran}/n_{tot}$. We compute this fraction for DCSH f and DCSH-FORCE f^F, where subscripts B and O indicate the peak used BDD nodes and total operation count, respectively. These fractions are shown in Table 2. If $f < 0.50$ for both metrics, DCSH or DCSH-FORCE outperforms FORCE.

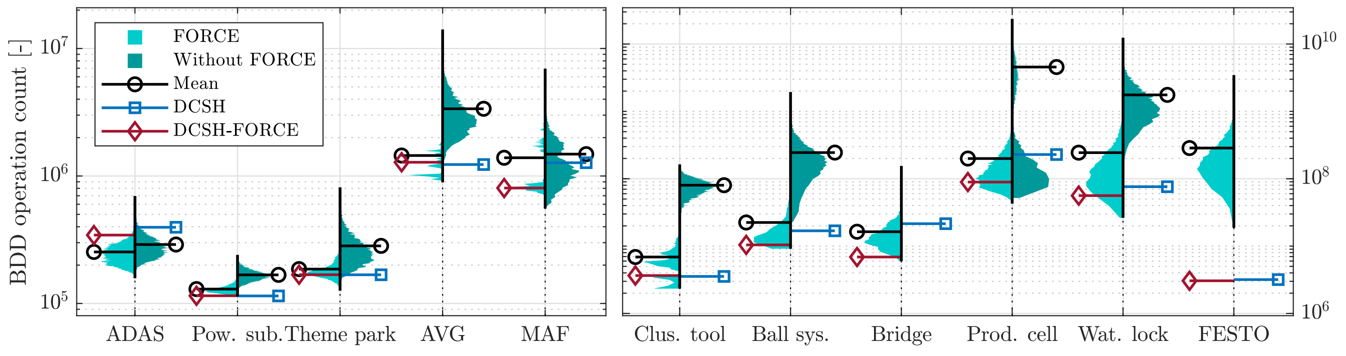
For five out of eleven models, DCSH performed better than FORCE. For these models, both metrics show that it resulted in less effort than 50% of the random orders

Table 2. Fractions and WES for DCSH and DCSH-FORCE.

Model	DCSH			DCSH-FORCE		
	f_B	f_O	WES	f_B^F	f_O^F	WES
ADAS	1.00	1.00	0.044	1.00	0.97	0.061
Pow. sub.	0.01	0.00	0.063	0.02	0.00	0.065
Theme park	0.70	0.28	0.025	0.80	0.28	0.080
AVG	0.37	0.45	0.166	0.37	0.51	0.170
MAF	0.60	0.50	0.017	0.19	0.19	0.037
Cluster tool	0.06	0.17	0.055	0.12	0.18	0.050
Ball system	0.87	0.54	0.043	0.50	0.10	0.042
Bridge	0.97	0.81	0.090	0.02	0.01	0.086
Prod. cell	0.55	0.72	0.070	0.09	0.16	0.065
Wat. lock	0.16	0.13	0.078	0.05	0.05	0.075
FESTO	0.00	0.00	0.041	0.00	0.00	0.041



(a) Peak used BDD nodes.



(b) BDD operation count.

Fig. 8. Synthesis effort of applying DCSH, FORCE and DCSH-FORCE.

where FORCE is applied, see Table 2. Applying DCSH-FORCE shows this improvement for seven out of eleven models. Furthermore, it can be seen in both Table 2 and Fig. 8 that DCSH performed noticeably better for the two largest models tested in this case study, the FESTO and Waterway lock models. The lowest overall effort is shown for DCSH-FORCE. This can be explained by the WES shown in Table 2. For most cases, applying FORCE to the order computed by DCSH resulted in a further decrease of the WES that also resulted in a decrease of synthesis effort. This can be explained as FORCE's span and the WES are closely related metrics (Meijer and van de Pol, 2016). By providing an initial order for FORCE with a low WES, we also provide an order that initially has a low span. Subsequently, FORCE optimizes the span which also results in a lower WES, this was the case for the five most complex models. For some cases the WES did increase as well as the effort. The only models that did not show this correlation between effort and the WES are the ADAS, MAF and Cluster tool models. The authors note that even for models describing the same system, different modeling strategies can result in varying results of DCSH. Overall, DCSH chose the weighted CM six times and Sloan five times. The chosen variable ordering with minimal WES did not always result in the lowest effort, but using the WES rather than always weighted CM or Sloan was still the better strategy.

6. CONCLUSION

We present a variable ordering heuristic for the reduction of computational effort of BDD-based supervisor synthesis. We show why certain variable orderings result in effort that is higher in order of magnitude. The relation between transition relations, variable order and computational effort has been studied in literature, however, to the best of the authors' knowledge, no detailed explanation for the existence of this relation had been given in prior work. Benchmark experiments are performed to compare DCSH to FORCE which shows that DCSH performs competitively with FORCE. For the most computationally demanding models, DCSH shows better results. By using the two heuristics in sequence, the largest effort reduction is observed.

As the computation time for running both heuristics is in the order of hundreds of milliseconds and the benefits for the computational effort that can be obtained is orders of magnitude larger, use of the method shown in this paper is recommended. The best results are shown for the model with the highest worst-case peak used BDD nodes, the FESTO model, where the effort reduced by over a factor of 50 compared to the mean of applying FORCE to random orders.

For future work it would be of interest how DCSH performs for different, albeit similar, applications such as model checking. Sloan's ordering can be adjusted such that it is able to sort a numerical DSM. Moreover, it would be interesting to investigate whether additional (matrix) ordering schemes could result in a further decrease of synthesis effort.

REFERENCES

Akers, S. (1978). Binary decision diagrams. *IEEE Trans. Comput.*, 27(6), 509–516.

Aloul, F., Markov, I., and Sakallah, K. (2003). FORCE: A fast and easy-to-implement variable-ordering heuristic. In *Proc. ACM Great Lakes Symp. VLSI*, 116–119.

Bollig, B. and Wegener, I. (1996). Improving the variable ordering of OBDDs is NP-complete. *IEEE Trans. Comput.*, 45(9), 993–1002.

Browning, T. (2016). Design structure matrix extensions and innovations: A survey and new opportunities. *IEEE Trans. Eng. Manag.*, 63(1), 27–52.

Bryant, R. (1992). Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3), 293–318.

Burch, J., Clarke, E., Long, D., McMillan, K., and Dill, D. (1994). Symbolic model checking for sequential circuit verification. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 13(4), 401–424.

Cai, K. and Wonham, W. (2015). New results on supervisor localization, with case studies. *Discrete Event Dynamic Syst.*, 25(1-2), 203–226.

Čengić, G. and Åkesson, K. (2008). A control software development method using IEC 61499 function blocks, simulation and formal verification. *IFAC Proc. Volumes*, 41(2), 22–27.

Chao, W., Tang, Z., Lin, G., Guo, J., Lin, W., and Yu, S. (2017). Modular supervisory control of computer based preventing electric mal-operation system for multiple bays in substation. In *IEEE Inf. Technol. Netw. Electron. Autom. Control Conf.*, 1730–1739.

Cuthill, E. and McKee, J. (1969). Reducing the bandwidth of sparse symmetric matrices. In *Proc. Nat. Conf.*, 157–172.

Feng, L., Cai, K., and Wonham, W. (2009). A structural approach to the non-blocking supervisory control of discrete-event system. *Int. J. Adv. Manuf. Technol.*, 41, 1152–1168.

Forschelen, S., van de Mortel-Fronczak, J., Su, R., and Rooda, J. (2012). Application of supervisory control theory to theme park vehicles. *Discrete Event Dynamic Syst.*, 22(4), 511–540.

George, A. and Liu, J.W.H. (1979). An implementation of a pseudoperipheral node finder. *ACM Trans. Math. Softw.*, 5(3), 284–295.

Goorden, M., van de Mortel-Fronczak, J., Reniers, M., and Rooda, J. (2017). Structuring multilevel discrete-event systems with dependency structure matrices. In *IEEE Conf. Decis. Control*, 558–564.

Korssen, T., Dolk, V., van de Mortel-Fronczak, J., Reniers, M., and Heemels, M. (2018). Systematic model-based design and implementation of supervisors for advanced driver assistance systems. *IEEE Trans. Intell. Transp. Syst.*, 19(2), 533–544.

Malik, R., Åkesson, K., Flordal, H., and Fabian, M. (2017). Supremica - An efficient tool for large-scale discrete event systems. *IFAC-PapersOnLine*, 50(1), 5794–5799.

McMillan, K. (1993). *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell, MA, USA.

Meijer, J. and van de Pol, J. (2016). Bandwidth and wavefront reduction for static variable ordering in symbolic reachability analysis. In *NASA Formal Methods*, 255–271.

Minato, S. (1996). *Binary Decision Diagrrams and Applications for VLSI CAD*, volume 342 of *Springer Int. Series Eng. Comput. Sci.* Springer US.

Miremadi, S., Lennartson, B., and Åkesson, K. (2012). A BDD-based approach for modeling plant and supervisor by extended finite automata. *IEEE Trans. Control Syst. Technol.*, 20(6), 1421–1435.

Nadales Agut, D. and Reniers, M. (2011). Linearization of CIF through SOS. In *Proc. Int. Workshop Expressiveness Concurrency*, 74–88.

Ouedraogo, L., Kumar, R., Malik, R., and Åkesson, K. (2011). Nonblocking and safe control of discrete-event systems modeled as extended finite automata. *IEEE Trans. Autom. Sci. Eng.*, 8, 560–569.

Ramadge, P. and Wonham, W. (1989). The control of discrete event systems. *Proc. IEEE*, 77(1), 81–98.

Reijnen, F., Goorden, M., van de Mortel-Fronczak, J., Reniers, M., and Rooda, J. (2018a). Application of dependency structure matrices and multilevel synthesis to a production line. In *IEEE Conf. Control Technol. Appl.*, 458–464.

Reijnen, F., Goorden, M., van de Mortel-Fronczak, J., and Rooda, J. (2017). Supervisory control synthesis for a waterway lock. In *IEEE Conf. Control Technol. Appl.*, 1562–1563.

Reijnen, F., Reniers, M., van de Mortel-Fronczak, J., and Rooda, J. (2018b). Structured synthesis of fault-tolerant supervisory controllers. *IFAC-PapersOnLine*, 51(24), 894–901.

Siminiceanu, R. and Ciardo, G. (2006). New metrics for static variable ordering in decision diagrams. In *Tools Algorithms Construction Anal Syst.*, 90–104.

Skoldstam, M., Åkesson, K., and Fabian, M. (2007). Modeling of discrete event systems using finite automata with variables. In *IEEE Conf. Decis. Control*, 3387–3392.

Sloan, S. (1989). A FORTRAN program for profile and wavefront reduction. *Int. J. Numerical Methods Eng.*, 28, 2651–2679.

Somenzi, F. (1999). Binary decision diagrams. In *Calculational Syst. Design, NATO Sci. Series F: Comput. Syst. Sci.*, volume 173, 303–366.

Su, R., van Schuppen, J., and Rooda, J. (2010). Aggregative synthesis of distributed supervisors based on automaton abstraction. *IEEE Trans. Autom. Control*, 55(7), 1627–1640.

Theunissen, R., Petreczky, M., Schiffelers, R., van Beek, D., and Rooda, J. (2014). Application of supervisory control synthesis to a patient support table of a magnetic resonance imaging scanner. *IEEE Trans. Autom. Sci. Eng.*, 11(1), 20–32.

Thuijsman, S., Hendriks, D., Theunissen, R., Reniers, M., and Schiffelers, R. (2019). Computational effort of BDD-based supervisor synthesis of extended finite automata. In *IEEE Conf. Automation Sci. Eng.*, 486–493.

van Beek, D., Fokkink, W., Hendriks, D., Hofkamp, A., Markovski, J., van de Mortel-Fronczak, J., and Reniers, M. (2014). CIF 3: Model-based engineering of supervisory controllers. *Tools Algorithms Construction Anal. Syst.*, 575–580.

Wonham, W. and Cai, K. (2019). *Supervisory Control of Discrete-Event Systems*. Springer, Cham.

Wonham, W., Cai, K., and Rudie, K. (2018). Supervisory control of discrete-event systems: A brief history. *Annu. Rev. Control*, 45, 250–256.