Eindhoven University of Technology

MASTER

Computational effort of BDD-based supervisor synthesis of extended finite automata

Thuijsman, Sander B.

Award date:
2019

Link to publication

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 20. Apr. 2025

Computational Effort of BDD-based Supervisor Synthesis of Extended Finite Automata

Master Thesis

Sander Benjamin Thuijsman
0850788

Master:	Mechanical Engineering
Department:	Mechanical Engineering
Research Group:	Control Systems Technology
In collaboration with:	ASML
Supervisor (TU/e):	dr.ir. M.A. Reniers
Supervisor (ASML):	dr.ir. R.R.H. Schiffelers
Report ID:	CST2019.018

Eindhoven, April 2019

Abstract

We consider supervisor synthesis of Extended Finite Automata that are represented using Binary Decision Diagrams (BDDs). Peak used BDD nodes and BDD operation count are introduced as platform independent and deterministic metrics that quantitatively indicate the computational effort needed to synthesize a supervisor. The use of BDD operation count is novel with respect to expressing supervisor synthesis effort. The (dis-)advantages of using these metrics to state of practice metrics such as wall clock time and worst case state space size are analyzed. The supervisor synthesis algorithm is initiated with a certain event- and variable order. It is already known from literature that variable order influences synthesis performance. We show that the event order is also relevant to consider. We discuss how these orders influence the synthesis effort and, by performing an experiment on a set of models, we show the extent of this influence. The effectiveness of computational effort reduction is evaluated for variable ordering heuristic algorithms that are in established usage.

Preface

This thesis, 'Computational Effort of BDD-based Supervisor Synthesis of Extended Finite Automata', concludes the research that I have performed as part of my graduation project. This project serves to finalize my Master Mechanical Engineering study, to obtain the Master of Science title. I have partaken in this Master's program with great joy, as part of the Control Systems Technology research group, starting September 2016, until now, April 2019. Working on the graduation project, starting July 2018, has sparked an interest in me to continue research in the field of Supervisory Control Theory.

I would like to thank my supervisors of this graduation project, Michel and Ramon. You have given me incredible amounts freedom. This did not necessarily make my work any easier, but it made the final product extra rewarding to me. You have enabled me to develop skills valuable to performing independent research. Michel, I would notice you checking in on the Overleaf project at random times and you were willing to come over to ASML for the progress meetings. This shows how devoted you are. Thank you for this. Ramon, thank you for providing me with a workplace at ASML, it has been a great experience to get an insight in the company and the research colloquia were valuable for getting acquainted with many research topics.

Thank you to all colleagues at ASML (student and non-student) for making it a pleasant working environment during my graduation period. Dennis and Rolf, thank you for your discussions, they were of great value to my research. I only had to introduce a topic, and listen to the discussion you two had on it afterwards, in order to gain excellent insights. This greatly accelerated my research process.

I would like to thank my friends from study, in particular Thijs, Joey, Maarten, Job and Mathijs, being close friends from day one, starting together as first year Bachelor students. You have provided me with much needed distractions from the study and many great memories, of which there are surely a lot more to come. I would like to thank my friends and colleagues from Squadra Veloce and Applied Micro Electronics for their companionship during my extracurricular activities.

Last, but definitely not least, I would like to express my greatest thankfulness to my mother, Annelies, father, Frank, and my dearest sister, Eva, for their unconditional love and support.

I wish you enjoy the pages to come.

Sander

Contents

1. Introduction

Supervisory Control Theory (SCT) [1,2] is a model-based approach to control (cyber-physical) systems. Given a plant (a model that defines all possible system behavior) and a specification (a model that defines what behavior is forbidden), a supervisor can be computed algorithmically (synthesized) that restricts the plant's behavior so that it is in accordance with the specification. Depending on the synthesis algorithm, the supervised system has some useful properties by construction, such as safety, nonblockingness, controllability and maximal permissiveness. There are a number of formal modeling frameworks to which SCT can be applied. The framework of Extended Finite Automata (EFA) [3] is an extension to Finite State Automata by augmenting them with variables, guard expressions and updates, which enables more convenient modeling of systems.

The power of SCT has been demonstrated in literature [4–9]. Despite the beneficial properties of SCT, industrial acceptance is scarce. The exponential state space explosion that occurs during supervisor synthesis is a major hurdle [10]. A way to mitigate this is by symbolically representing the EFA using Binary Decision Diagrams (BDDs) [11–14]. This approach is considered state of the art to handle industrial sized systems [15].

The synthesis complexity depends on the amount of BDD nodes required to represent the system during synthesis [13]. It is well known that this number is largely dependent on the *variable order* [16]. A contribution of this paper is showing that in addition to the variable order, the *event order* is relevant to consider for the effort that is needed to synthesize a supervisor (supervisor synthesis effort). This is the order in which the synthesis algorithm iterates through the events when performing reachability/fixed point searches. By performing supervisor synthesis to a set of models using a large number of random variable- and event orders, we demonstrate to what extent these orders influence the supervisor synthesis effort. We also evaluate the effectiveness of computational effort reduction for variable ordering heuristic algorithms that are in established usage.

Even when restricting ourselves to only BDD-based supervisor synthesis of EFA, there is no consensus in literature on how to express the supervisor synthesis effort. Several metrics are used, such as: wall clock time, peak random access memory and state space sizes [6,15,17–19]. These metrics give some intuitive indication on the supervisor synthesis effort, but there is no exact method on how to interpret them. In this paper, we present *peak used BDD nodes* and *BDD operation count* as deterministic, platform independent metrics that provide a quantitative indication of the supervisor synthesis effort. Peak used BDD nodes has been used to express supervisor synthesis effort [13, 17, 18, 20]. The contribution of BDD operation count is novel in this context. We show the relevance of these BDD-based metrics to express supervisor synthesis effort, and compare them to state of practice metrics.

2. BDD-Based Supervisor Synthesis of EFA

2.1 Supervisor synthesis of EFA

We consider EFA A defined as *7-tuple*

$$A = (L, D, \Sigma, E, L^0, D^0, L^m)$$

where L is the domain of locations, $D = D_1 \times ... \times D_p$ is the domain of variables and Σ is the set of events, usually called the alphabet. L^0 is the set of initial locations, $D^0 = D_1^0 \times ... \times D_p^0$ is the set of initial variable values and L^m is the set of marked locations. E is a set of edges where an edge $e_k \in E$ is defined as *5-tuple*

$$e_k = (l_k^o, l_k^t, \sigma_k, g_k^e, f_k^e)$$

where l_k^o and l_k^t are the origin and target location in L, σ_k is an event in Σ, $g_k^e : D \rightarrow \{false, true\}$ is the guard evaluation function and $f_k^e : D \rightarrow D$ is the update function that assigns new values to the variables.

The *state* of the EFA specifies the automaton location and the value of each of the variables as a pair $(l, d) \in L \times D$. Consequently the initial states are pairs $(l^0, d^0) \in L^0 \times D^0$ and the marked states are pairs $(l^m, d) \in L^m \times D$.

An edge e_k is *enabled* if the current location is l_k^o and g_k^e evaluates to true for the current variable values. Only enabled edges can be executed. Upon execution, the state is updated according to l_k^t and f_k^e. Thus we can speak of a origin and target state that relate to an edge.

The set of events Σ is split into two disjoint subsets, Σ_c and Σ_u, denoting *controllable* and *uncontrollable* events. We speak of controllable and uncontrollable edges respective to the event they are labeled with. Controllable edges can be enforced by the supervisor, uncontrollable edges can not. Thus, the supervisor can only disable an edge e_k when $\sigma_k \in \Sigma_c$.

This paper is based on the supervisor synthesis algorithm for EFA as presented in [21]. An EFA of the plant that models all possible behavior and an EFA of the specification, where all forbidden behavior of the system ends in blocking states, are given. The algorithm produces a supervisor EFA by iteratively strengthening guards resulting in forbidden or blocking states finally becoming unreachable. This supervisor EFA contains all behavior of the system that is safe (forbidden states can never be reached), nonblocking (a marked state can always be reached), controllable (only controllable edges are restricted by the supervisor) and it is maximally permissive (the restrictions are minimal in order to guarantee safety, nonblockingness and controllability). A high level pseudo code description of the algorithm is given in Algorithm 1. A more rigorous description can be found in [21]. *Flagging* a state by true, denotes adapting a predicate so that the new predicate evaluates to true for that state.

2.2 CIF toolset

There are several tools that allow modelling of plants and requirements with the ability to synthesize a supervisor. Of the tools considered in [22], the tools Supremica [15] and CIF [23] allow for the use of EFA. Both tools base their EFA supervisor synthesis algorithm on the use of BDDs. The syntheses in this paper are performed using the EFA supervisor synthesis tool

Algorithm 1 Supervisor synthesis of EFA

Input: Plant P and specification R represented as EFA
Output: Maximally permissive nonblocking, safe and controllable EFA G respective to P and satisfying R

1: Create refined EFA G that has the same behavior as P, where the disallowed behavior by R ends in a set of forbidden states in G
2: Create bad state predicate $\mathcal{B}$: flag all forbidden states of G by true and all others by false
3: **repeat** Create new nonblocking predicate $\mathcal{N}$: flag all marked states of G by true and all unmarked states by false
4: **repeat** Flag all states in $\mathcal{N}$ by true that have an enabled edge to a state already flagged true in $\mathcal{N}$
5: **until** $\mathcal{N}$ did not change
6: Flag all states in $\mathcal{B}$ that are currently set to false in $\mathcal{B}$ by their negation in $\mathcal{N}$
7: **repeat** Flag all states in $\mathcal{B}$ by true that have an enabled uncontrollable edge to a state already flagged true in $\mathcal{B}$
8: **until** $\mathcal{B}$ did not change
9: Set guards in G to false so that all controllable edges that lead to a state flagged true in $\mathcal{B}$ are disabled
10: **until** Guards did not change

of CIF[1]. CIF has been used to synthesize supervisors for industrial sized systems [4–6, 24]. This EFA supervisor synthesis tool is an implementation based on the supervisor synthesis algorithm as sketched in Section 2.1, barring some minor differences that are irrelevant to our interpretation of the algorithm. The tool is instrumented to extract the metrics as introduced in Section 3.

2.3 Binary Decision Diagrams

Boolean functions can be used to symbolically represent EFA [25]. An efficient manner to store and make adaptations to Boolean functions is using BDDs [11, 12]. These are directed acyclic graphs that contain decision nodes that all have two child nodes which can be reached by taking a low/0 or high/1 decision edge from the parent node. At the bottom of the graph there are two terminal nodes representing true and false. (Groups of) nodes represent variable values or automata locations. When referring to BDDs here, we consider Reduced Ordered BDDs [26] which impose some extra restrictions, guaranteeing a canoncial form. This representation is better suited for computer manipulation [13]. After converting the EFA to a BDD structure, the synthesis operations are directly applied to the BDDs. When synthesis is complete, the BDDs can be converted back to the EFA structure. Several synthesis algorithms that employ BDDs exist. Their efficiency usually depends on their handling of the BDDs [13, 14, 18].

[1]The CIF tooling and documentation is open source and freely available at `cif.se.wtb.tue.nl`

2.4 Variable order

Because *Ordered* BDDs are used, a total ordering is imposed over the set of variables. If in the variable order a precedes b (denoted $a < b$), then no path of decision edges can exist in the BDD from the nodes representing b to a [26]. This variable order has a major effect on the amount of BDD nodes that are necessary to represent a system [16]. In turn, it has a large influence on the efficiency of supervisor synthesis for EFA [13, 18]. Finding the optimal variable order is NP-hard [27], thus CIF uses heuristics to come up with a good order. First, the FORCE algorithm [28] is applied: All model variables are indexed vertices in some intial order. A hyperedge is placed between two vertices if their respective variables appear together in a guard or update expression of an edge. The span of a hyperedge is the difference between the index values of the connected vertices, which indicates their distance. After this, new tentative locations for the vertices are found by taking the sum of the spans of all connected hyperedges to the vertex. Then, the vertices get reindexed in ascending order of their tentative location values and the algorithm can iterate again. Iterations stop once the average span per hyperedge does not decrease anymore. After applying the FORCE algorithm, the sliding-window algorithm is applied where a small window is slid over all the variable order returned by FORCE and exhaustively searches for the variable order within the window resulting in the lowest span [28]. Orders that have a low average span for the hyperedges are considered good, because variables that are often considered together are near each other in the diagram. This makes a relatively small BDD representation more likely. Note that these algorithms come up with local optima, providing them with different initial orders, will lead to different resulting variable orders. In CIF, the default initial variable order is an alphabetic ordering of the names of all variables and automata locations.

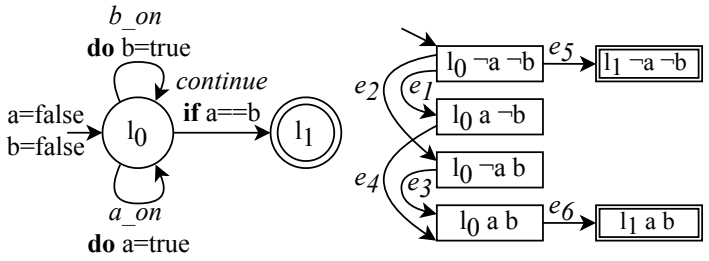
2.5 Edge order

Next to variable order, the edge order is also relevant when considering supervisor synthesis effort. In lines 4 and 7 of Algorithm 1, a backwards reachability search is performed to find edges of which the target state is flagged true in predicate $\mathcal{N}$ or $\mathcal{B}$, and the origin state is not. If such an edge is found, the predicate will be adapted on the fly; The iteration over edges does not restart, and the new predicate will immediately be considered for the next edge that is checked. Applying a different edge order will not influence the resulting predicate when exiting the reachability loop. However, the BDD sizes required to represent the intermediate predicates, and the rate at which states get flagged in the predicate may differ. The construction of the guards in line 9 of Algorithm 1 is also influenced by the edge order. Unlike the variable order, no reordering algorithm is applied to the edge order in CIF.

Example

We consider the EFA of Figure 2.1a. This EFA consists out of two locations $L = \{l_0, l_1\}$ of which l_1 is marked: $L^m = \{l_1\}$, as indicated in Figure 2.1a by a double circle. We have Boolean variables a and b, forming variable space $D = \{false, true\} \times \{false, true\}$. Both variables are initially set to false: $D^0 = \{(false, false)\}$. Events a_on and b_on can occur at l_0, variables a and b will then respectively update to $true$. These updates are denoted in Figure 2.1a by the keyword 'do'. An edge with event label *continue* can be taken from origin

location l_0 to target location l_1. This can only happen if the guard $a == b$ evaluates to $true$. The guard is denoted by the keyword 'if'. The reachable state space of this EFA is displayed in Figure 2.1b. The edges have been enumerated e_1 to e_6. We use a or $\neg a$ to respectively denote the values $a = true$ or $a = false$, same holds for b.



(a) EFA (b) State space

Figure 2.1: *Example EFA and respective reachable state space*

Let us consider the construction of the nonblocking predicate $\mathcal{N}$. Initially, only the marked states are flagged as true. As our example only has two locations, a single Boolean variable can be used to indicate the location. We use location specifier l_s to indicate l_0 and $\neg l_s$ to indicate l_1. The BDD representing the initial predicate is shown in Figure 2.2a. Solid decision edges denote that the origin node evaluates to $true$, dashed decision edges denote $false$. The square T and F vertices represent the $true$ and $false$ terminal nodes.

Let us examine the case that we continue performing backwards reachability by first considering edge e_6. The target state of this edge is part of the nonblocking predicate, thus we can flag its origin state (l_s a b) in the predicate, resulting in the BDD of Figure 2.2b. All BDDs in Figure 2.2 have variable order $l_s < a < b$. A caption 'B.w. $e_x; e_y$' denotes backwards reachability by first applying edge e_x followed by edge e_y. Applying edge order $e_6; e_5; e_3$ results in the same nonblocking predicate as applying $e_6; e_3; e_5$. The resulting BDD is shown in Figure 2.2e. However, the intermediate BDDs to represent the predicate after applying $e_6; e_5$ or $e_6; e_3$ are of different sizes, seen when comparing Figure 2.2c to 2.2d. This illustrates how the edge order can influence the BDD sizes required to represent the intermediate predicates.

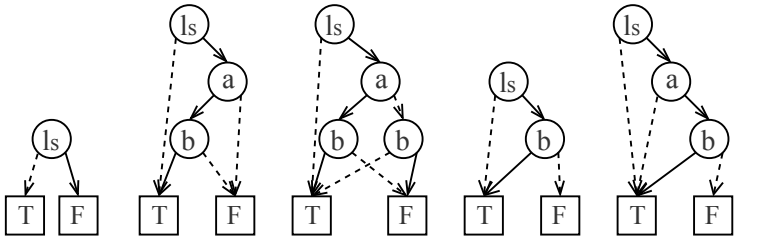


Figure 2.2: *Different BDD sizes for varying edge orders*

2.6 Ordering granularity

The state spaces of supervisory control models often contain even more edges than states. When analyzing edge orders, instead of working with these very large sets, we use a less granular approach and group all edges with the same event label together. In the remainder of this paper we will consequently refer to these orders by *event* orders rather than *edge* orders. Likewise, when we say variable order, we will consider the variables and automata locations as they are defined in the model, instead of the order of the bits in the BDD. We also do not interleave [16] variables with each other. Although not presented in this paper, using these lower granularities allows for more intuitive interpretation of why one certain order results in a more reduced supervisor synthesis effort than some other.

3. Metrics for Computational Effort

Algorithms are typically judged by their space- and time complexity [29]. We propose two metrics; *peak used BDD nodes* and *BDD operation count* to quantitatively express the required space- and time effort required for supervisor synthesis. These metrics have some advantages over wall clock time and peak random access memory usage. First, they are deterministic; performing a supervisor synthesis twice with the same input and algorithm configuration will result in exactly the same result. This determinism also holds when doing the synthesis on two different platforms, even if one is a supercomputer and the other is a personal computer. As a result, it becomes easier to compare results from different publications, even if performed decades apart. Second, there is no overhead in the measurement, loaded-in Java classes or other computer processes will not influence the measurement. After introducing these BDD-based metrics, an elaboration on their relation to the conventional metrics will be given in Section 3.3.

We purposefully distinguish *complexity* from *effort*. Complexity defines the generic trend of the (space/time) resources an algorithmic computation requires for inputs of different sizes, an is often expressed using the 'Big O' notation [30]. Effort specifies the amount of (space/time) resources of a particular algorithmic computation. The complete input is considered rather than only its size. This includes algorithm configuration settings and variable- and event order.

3.1 Peak used BDD nodes

The number of nodes in a BDD during the synthesis procedure is important to the space complexity [13, 29]. During supervisor synthesis, the number of BDD nodes to describe the system generally fluctuates. Since *Reduced* Ordered BDDs are used, which are minimal representations, the peak used BDD nodes is also the absolute minimal amount of BDD nodes that are necessary to represent the predicates needed to solve the supervisor synthesis problem.

In CIF, BDD nodes are stored in a hash table. Each new node is allocated to an entry in the hash table. Once the hash table reaches a certain fill rate, garbage collection is employed to free no longer used entries. We only count the *used* BDD nodes, i.e. hash table entries that still contain relevant information for the BDDs that are still in use. Garbage collection is performed by means of a standard mark-and-sweep algorithm. Functions from the implementation of this algorithm in the JavaBDD library[2] are reused to count the BDD nodes that are in use. Peak used BDD nodes is a reproducible metric; Performing a supervisor synthesis twice with the same input yields exactly the same peak used BDD nodes.

3.2 BDD operation count

The time complexity can be expressed in number of steps/operations of an algorithm [29]. The time complexity of performing operations on BDDs is dependent on the number of nodes in the BDDs. For example, performing a Boolean operation $(a \otimes b)$ has a worst-case time complexity $\mathcal{O}(\#a \times \#b)$, where $\#a$ denotes the number of nodes in BDD a [31]. As the supervisory

[2]The JavaBDD library is available at `javabdd.sourceforge.net`

synthesis is done by performing operations on BDDs, we use BDD operation count to express the time effort of performing supervisor synthesis. Since BDD operations (such as `and`, `or` and `not`) are implemented as functions that employ structural recursion on BDD nodes, the number of invocations of such functions can be used to express time effort. Since the functions are deterministic, the results are reproducible.

Generally, these functions consist of three parts. First, a few checks are made to see whether the requested calculation is a terminal case. Second, if it is a non-terminal case, it is checked whether the calculation has already been performed, and is still in the cache. Note that we do not mean hardware cache here, but a table actively storing results of previous calculations. If both previous cases did not occur, the function performs recursive expansion over the child nodes. For more details about terminal cases, cache lookup and recursive expansion over child nodes we refer to [32].

Checking whether there is a terminal case, or looking for some solution in the cache can be done with relatively low computational effort. Thus, the time effort is mainly influenced by the third part of the algorithm, where the actual operations are applied to the BDD [32]. Therefore, we only increase the BDD operation count when we reach this part of the algorithm, including all recursive invocations that reach this part.

3.3 Relevance of metrics

In order to compare the BDD-based metrics to the conventional metrics, we perform a number of supervisor syntheses and extract these metrics. The data presented in this paper is acquired by performing supervisor syntheses to the models shown in Table 3.1[3]. The models are selected to have a wide range of model sizes. Table 3.1 shows the worst case state space size of the uncontrolled plant for each model, which is the product of all location and variable domain sizes.

Table 3.1: *Case study models*

Name	**Worst case state space**
Robotic swarm aggregation [33]	$1.0 \cdot 10^{0}$
Robotic swarm clustering [33]	$1.0 \cdot 10^{0}$
Robotic swarm segregation [33]	$6.4 \cdot 10^{1}$
Robotic swarm formation [33]	$8.0 \cdot 10^{1}$
Power substation system [34]	$2.1 \cdot 10^{10}$
Advanced driver assistance system [35]	$3.4 \cdot 10^{9}$
Multi agent formation [36]	$1.0 \cdot 10^{3}$
Ball sorting system [37]	$7.4 \cdot 10^{4}$
Production cell [38]	$7.5 \cdot 10^{8}$
FESTO production line [6]	$1.3 \cdot 10^{28}$
Waterway lock [24]	$6.0 \cdot 10^{32}$

For a supervisor synthesis of the Waterway lock model, Figure 3.1 shows how the number

[3]Some of these models are readily available at `github.com/magoorden/T-AC2018`, others can be extracted from the listed references.

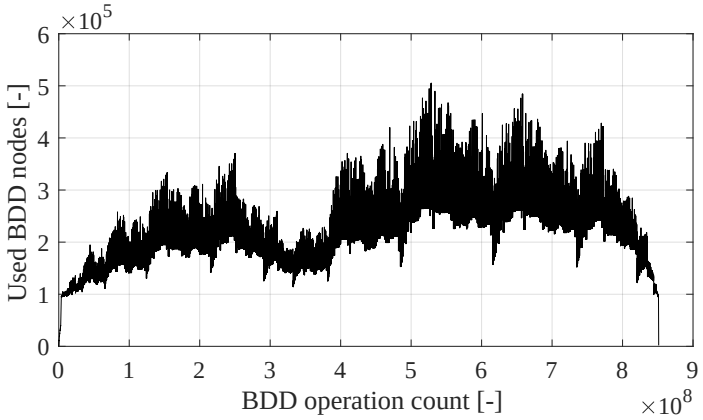


Figure 3.1: *Evolution of used BDD nodes during synthesis*

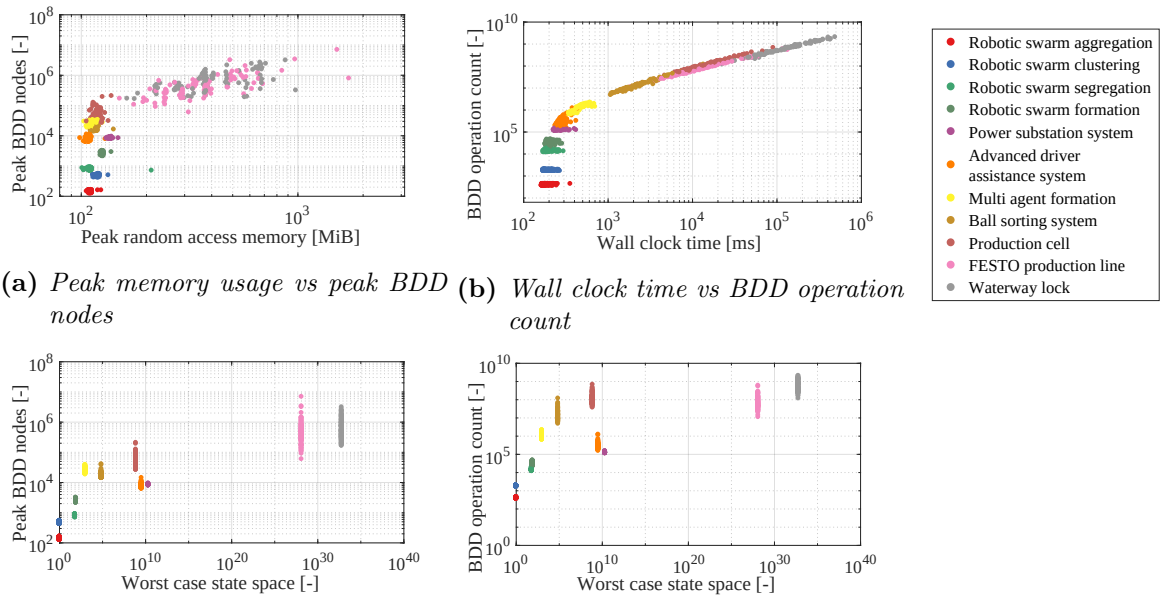
of used BDD nodes evolves, and how many BDD operations were used thus far to come up with those BDD nodes. The metrics presented in this paper are the maxima along both axes in this plot; the peak used BDD nodes and the final BDD operation count.

Figures 3.2a and 3.2b show how peak random access memory and wall clock time relate to peak used BDD nodes and BDD operation count. A supervisor was synthesized for each model of Table 3.1 for 100 pairs of random variable- and event orders. Note that the heuristic variable ordering algorithms were turned on for this test. The measurements for random access memory and wall clock time were done separately from the measurements of the BDD-based metrics to avoid them from interfering. It can be seen that for small models, peak random access memory and wall clock time can not indicate a difference in synthesis effort, as all results of the small models are grouped around the same result $\sim (1 \cdot 10^2$ MiB, $2 \cdot 10^2$ ms). Influences like loaded-in Java classes for the peak random access memory and reading the input- and writing the output file for wall clock time dominate these metrics. The BDD-based metrics enable a distinction in effort for the actual synthesis part of the computation.

For larger computations, a linear relation is visible between wall clock time and BDD operation count. The threshold at which this relation starts, and its slope, are dependent on the used hardware. The scattering that is seen for larger computations in Figure 3.2a is a result of the manner in which the BDD space allocation takes place; When the current table is full, it gets doubled in size, the new free entries in this table will have an influence on the memory, but are not measured when counting the used BDD nodes. Also, when performing computations that require more memory, Java (Java Virtual Machine) will perform garbage collection in the background to free memory. For separate measurements this will happen at different times, which impacts the peak random access memory, not the amount of used BDD nodes.

An advantage of wall clock time and peak random access memory is that a user performing supervisor synthesis is more likely to be familiar with these metrics. It gives a better idea whether their computer is able to perform the synthesis in an acceptable amount of time given the available amount of memory.

The advantage of using worst case state space size of the uncontrolled system over BDD-based metrics to indicate the synthesis effort, is that no supervisor synthesis or reachability



(a) *Peak memory usage vs peak BDD nodes*

(b) *Wall clock time vs BDD operation count*

(c) *Worst case state space size vs peak BDD nodes*

(d) *Worst case state space size vs BDD operation count*

Figure 3.2: *BDD-based metrics against conventional metrics*

computations are required to calculate this number. Figures 3.2c and 3.2d show how this state space size relates to the BDD-based metrics. There is some general trend of a larger state space size suggesting more supervisor synthesis effort, but it is clearly not a very accurate indicator.

4. Impact of Variable- and Event Order on Computational Effort

We have presented two metrics that indicate the computational effort of a single supervisor synthesis procedure. These metrics are fairly easy to extract when performing a computation. However, we have to take care when making conclusions on this synthesis effort. The variable- and event order as introduced in Section 2 have an influence on the results. This can also be seen in Figure 3.2, where the results are scattered due to using different variable- and event orders. Recall that the BDD-based metrics are deterministic, re-performing a synthesis with the same variable- and event orders would provide the exact same result.

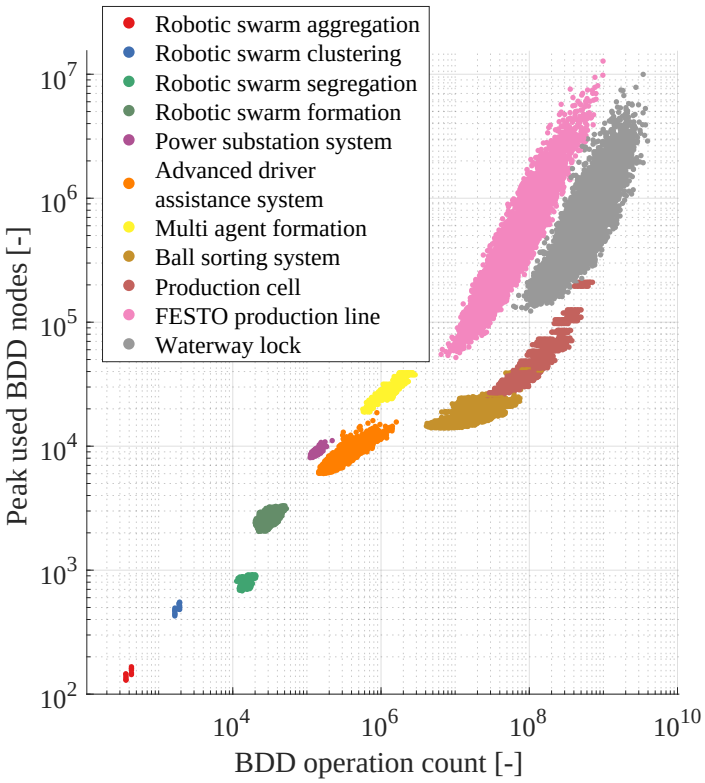


Figure 4.1: *Supervisor synthesis effort for all combinations of 100 event- and 100 variable orders for each model*

4.1 Extent of order influence

We investigate to what extent the initial variable order and event order influence the supervisor synthesis effort. For each of the models of Table 3.1, a supervisor has been synthesized for all combinations of 100 random event orders and 100 random initial variable orders. The effort of performing each synthesis is shown in Figure 4.1. It can be seen that there are major differences in computational effort by using different orders. For the FESTO production line, the highest peak used BDD nodes is 246 times larger than the lowest peak used BDD nodes. For BDD operations this factor is 153. This is purely a result of changing the variable- and

event orders; all other algorithm configurations were the same for all measurements. The results are especially surprising when considering that the heuristic variable order algorithms described in Section 2.4 are being applied to the initial variable order.

Figure 4.1 also shows that measuring both peak used BDD nodes and BDD operation count is relevant. It would be difficult to distinguish the computational effort between some of the models, if only one of the metrics was used. For example, if we only measured the peak used BDD nodes, we would not see much difference for the efforts of the Ball sorting system and Multi agent formation. Additionally considering the BDD operation count enables us to conclude that less effort is needed for Multi agent formation.

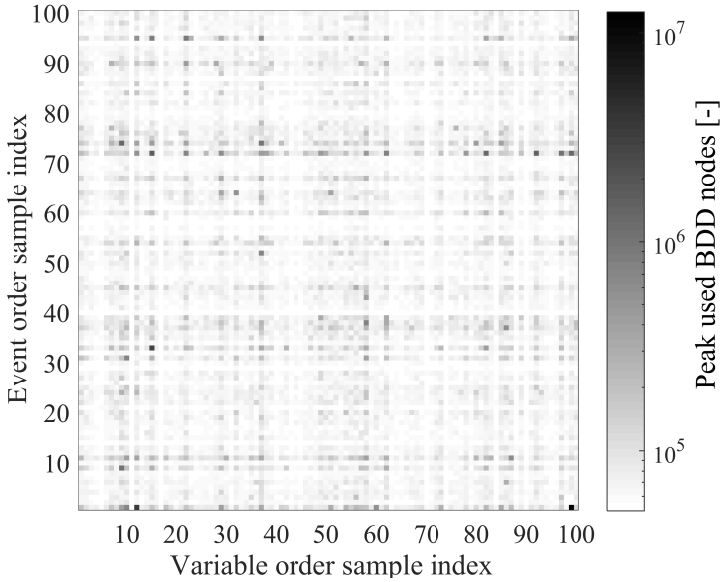


Figure 4.2: *Peak used BDD nodes for all supervisor syntheses of FESTO production line*

Table 4.1: *Sample means and variances of all models*

Name	$\mu_G(f)$	$\sigma^2_G(f)$	$\overline{\sigma^2_v}(f)$	$\overline{\sigma^2_e}(f)$	$\mu_G(g)$	$\sigma^2_G(g)$	$\overline{\sigma^2_v}(g)$	$\overline{\sigma^2_e}(g)$
Robotic swarm aggregation	$1.4 \cdot 10^2$	$1.3 \cdot 10^2$	$3.9 \cdot 10^1$	$4.0 \cdot 10^{-2}$	$4.1 \cdot 10^2$	$1.1 \cdot 10^3$	$9.9 \cdot 10^1$	$1.2 \cdot 10^3$
Robotic swarm clustering	$4.8 \cdot 10^2$	$9.2 \cdot 10^2$	$3.1 \cdot 10^2$	$1.3 \cdot 10^0$	$1.8 \cdot 10^3$	$1.5 \cdot 10^4$	$7.0 \cdot 10^2$	$1.5 \cdot 10^4$
Robotic swarm segregation	$8.1 \cdot 10^2$	$3.5 \cdot 10^3$	$4.1 \cdot 10^2$	$4.9 \cdot 10^5$	$1.4 \cdot 10^4$	$2.9 \cdot 10^6$	$3.4 \cdot 10^3$	$2.7 \cdot 10^6$
Robotic swarm formation	$2.6 \cdot 10^3$	$6.5 \cdot 10^4$	$8.2 \cdot 10^3$	$8.1 \cdot 10^6$	$3.3 \cdot 10^4$	$3.3 \cdot 10^7$	$6.1 \cdot 10^4$	$2.8 \cdot 10^7$
Power substation system	$8.4 \cdot 10^3$	$5.8 \cdot 10^4$	$2.0 \cdot 10^4$	$4.7 \cdot 10^7$	$1.3 \cdot 10^5$	$6.6 \cdot 10^7$	$4.8 \cdot 10^4$	$3.6 \cdot 10^7$
Advanced driver assistance system	$7.9 \cdot 10^3$	$1.4 \cdot 10^6$	$6.6 \cdot 10^5$	$1.2 \cdot 10^{10}$	$3.1 \cdot 10^5$	$1.6 \cdot 10^{10}$	$9.4 \cdot 10^5$	$7.6 \cdot 10^9$
Multi agent formation	$2.6 \cdot 10^4$	$2.4 \cdot 10^7$	$6.8 \cdot 10^1$	$7.4 \cdot 10^9$	$1.2 \cdot 10^6$	$2.1 \cdot 10^{11}$	$2.5 \cdot 10^7$	$2.1 \cdot 10^{11}$
Ball sorting system	$1.7 \cdot 10^4$	$1.9 \cdot 10^7$	$2.3 \cdot 10^5$	$1.5 \cdot 10^{13}$	$1.8 \cdot 10^7$	$2.5 \cdot 10^{14}$	$1.9 \cdot 10^7$	$2.4 \cdot 10^{14}$
Production cell	$5.2 \cdot 10^4$	$8.8 \cdot 10^8$	$4.2 \cdot 10^5$	$2.6 \cdot 10^{14}$	$1.4 \cdot 10^8$	$1.0 \cdot 10^{16}$	$8.8 \cdot 10^8$	$1.0 \cdot 10^{16}$
FESTO production line	$6.2 \cdot 10^5$	$4.8 \cdot 10^{11}$	$3.8 \cdot 10^{11}$	$4.6 \cdot 10^{15}$	$9.0 \cdot 10^7$	$6.3 \cdot 10^{15}$	$3.4 \cdot 10^{11}$	$4.2 \cdot 10^{15}$
Waterway lock	$7.6 \cdot 10^5$	$4.0 \cdot 10^{11}$	$2.6 \cdot 10^{11}$	$1.0 \cdot 10^{17}$	$6.8 \cdot 10^8$	$1.9 \cdot 10^{17}$	$2.8 \cdot 10^{11}$	$1.3 \cdot 10^{17}$

Figure 4.2 shows the peak used BDD nodes for all syntheses of the FESTO production line model. Darker squares indicate a higher amount of peak used BDD nodes. All variable- and event orders were given an index. A row shows the peak used BDD nodes of all syntheses that

were performed with the same event order and varying variable orders, and a column shows the same for a fixed variable order with varying event orders. We see rows and columns where the elements are similarly colored, indicating that variable order and event order both have a reasonable impact on the peak used BDD nodes for this model. There were models where only the elements in columns were similarly colored, indicating that the variable order mainly influenced the synthesis effort. We observe similar results for the BDD operation count.

If we define the peak used BDD nodes for a certain model as a deterministic function $f(o_{v,i}, o_{e,j})$, where $o_{v,i}$ is the $i'th$ sample random variable order and $o_{e,j}$ the $j'th$ sample random event order, the global sample mean [39] of the peak used BDD nodes $\mu_G(f)$ is given by Equation 4.1.

$$\mu_G(f) = \frac{1}{N \cdot M} \sum_{i=1}^{N} \sum_{j=1}^{M} f(o_{v,i}, o_{e,j}) \tag{4.1}$$

where N and M respectively are the total number of sampled variable- and event orders. For our experiment, $N = M = 100$ for each model.

The global (unbiased) sample variance [39] of the peak used BDD nodes $\sigma^2_G(f)$ is given by Equation 4.2.

$$\sigma^2_G(f) = \frac{1}{N \cdot M - 1} \sum_{i=1}^{N} \sum_{j=1}^{M} \big(f(o_{v,i}, o_{e,j}) - \mu_G(f)\big)^2 \tag{4.2}$$

The sample variance $\sigma^2_{v,i}(f)$ of the peak used BDD nodes for the event orders tested with a particular variable order $o_{v,i}$, is given by Equation 4.3.

$$\sigma^2_{v,i}(f) = \frac{1}{M - 1} \sum_{j=1}^{M} \big(f(o_{v,i}, o_{e,j}) - \mu_{v,i}(f)\big)^2 \tag{4.3}$$

where $\mu_{v,i}(f) = \frac{1}{M} \sum_{j=1}^{M} f(o_{v,i}, o_{e,j})$ is the mean peak used BDD nodes of the event orders tested with variable order $o_{v,i}$. The mean sample variance for fixed variable orders $\overline{\sigma^2_v}(f)$ is computed by Equation 4.4.

$$\overline{\sigma^2_v}(f) = \frac{1}{N} \sum_{i=1}^{N} \sigma^2_{v,i}(f) \tag{4.4}$$

Equations 4.3 and 4.4 can analogously be applied to compute the sample variance of peak used BDD nodes for variable orders tested with particular event orders $\sigma^2_{e,j}(f)$, and the mean sample variance for fixed event orders $\overline{\sigma^2_e}(f)$. Likewise, we can define a function $g(o_{v,i}, o_{e,j})$ for the BDD operation count of a model and apply above computations to this.

When relating these characteristics to what we see in Figure 4.2, a low mean sample variance for fixed variable orders $\overline{\sigma^2_v}(f)$ would indicate a similar amount of peak used BDD nodes for a given variable order. This would be visible in Figure 4.2, as elements located in the

same column would be similarly colored. This would indicate that the variable order mainly influences the peak used BDD nodes, and the event order has little influence.

For each model, the global sample mean μ_G, global sample variance σ_G^2, mean sample variance for fixed variable orders $\overline{\sigma_v^2}$ and mean sample variance for fixed event orders $\overline{\sigma_e^2}$ are given for peak used BDD nodes (f) and BDD operation count (g) in Table 4.1. For most of the models, the mean sample variance for fixed variable orders is smaller than the mean sample variance for fixed event orders. This indicates that the variable order has a larger influence on the supervisor synthesis effort than the event order. However, the mean variance for fixed variable orders is large enough that the event order is still of considerable influence to the supervisor synthesis effort.

Models that require a relatively large amount of supervisor synthesis effort, also have a relatively large variance in effort. This would also be observed if we were to normalize the variance to the mean values of the models (σ^2/μ). This indicates that applying a good variable- and event order becomes more beneficial when considering models that require more supervisor synthesis effort.

5. Effectiveness of CIF's Variable Ordering Heuristics

For the syntheses of which the results are presented in Section 4, the variable ordering heuristics of CIF as introduced in Section 2.4 have been applied. To get an insight in the effectiveness of the variable ordering heuristics, the same experiment has been executed, only now without applying the heuristics to the initial variable order. The same models and the same event orders and (initial) variable orders were used for this experiment as before. Only the Waterway lock model was omitted, as for some combinations of variable- and event order supervisor synthesis could not run to completion due to an out of memory error with 32GB memory allocated. The results are shown in Table 5.1 and Figure 5.1.

Table 5.1: *Sample means and variances without heuristic variable ordering*

Name	$\mu_G(f)$	$\sigma^2_G(f)$	$\overline{\sigma^2_v}(f)$	$\overline{\sigma^2_e}(f)$	$\mu_G(g)$	$\sigma^2_G(g)$	$\overline{\sigma^2_v}(g)$	$\overline{\sigma^2_e}(g)$
Robotic swarm aggregation	$1.4 \cdot 10^2$	$9.5 \cdot 10^1$	$6.5 \cdot 10^1$	$1.8 \cdot 10^{-2}$	$4.0 \cdot 10^2$	$3.9 \cdot 10^2$	$4.8 \cdot 10^1$	$4.0 \cdot 10^2$
Robotic swarm clustering	$4.5 \cdot 10^2$	$8.2 \cdot 10^2$	$5.2 \cdot 10^2$	$1.4 \cdot 10^0$	$1.7 \cdot 10^3$	$5.1 \cdot 10^3$	$4.4 \cdot 10^2$	$5.1 \cdot 10^3$
Robotic swarm segregation	$8.1 \cdot 10^2$	$8.2 \cdot 10^3$	$1.2 \cdot 10^2$	$7.1 \cdot 10^5$	$1.7 \cdot 10^4$	$9.7 \cdot 10^6$	$8.3 \cdot 10^3$	$9.3 \cdot 10^6$
Robotic swarm formation	$2.6 \cdot 10^3$	$7.5 \cdot 10^4$	$4.2 \cdot 10^3$	$8.9 \cdot 10^6$	$3.5 \cdot 10^4$	$3.7 \cdot 10^7$	$7.4 \cdot 10^4$	$3.0 \cdot 10^7$
Power substation system	$9.0 \cdot 10^3$	$1.5 \cdot 10^5$	$1.4 \cdot 10^5$	$1.9 \cdot 10^8$	$1.4 \cdot 10^5$	$1.9 \cdot 10^8$	$4.9 \cdot 10^4$	$4.0 \cdot 10^7$
Advanced driver assistance system	$8.4 \cdot 10^3$	$1.8 \cdot 10^6$	$1.1 \cdot 10^6$	$1.9 \cdot 10^{10}$	$3.6 \cdot 10^5$	$2.6 \cdot 10^{10}$	$1.1 \cdot 10^6$	$1.2 \cdot 10^{10}$
Multi agent formation	$2.8 \cdot 10^4$	$3.3 \cdot 10^7$	$4.5 \cdot 10^1$	$8.5 \cdot 10^9$	$1.2 \cdot 10^6$	$2.9 \cdot 10^{11}$	$3.4 \cdot 10^7$	$2.9 \cdot 10^{11}$
Ball sorting system	$5.3 \cdot 10^4$	$3.9 \cdot 10^8$	$8.3 \cdot 10^6$	$7.2 \cdot 10^{14}$	$1.9 \cdot 10^8$	$1.3 \cdot 10^{16}$	$3.9 \cdot 10^8$	$1.3 \cdot 10^{16}$
Production cell	$1.0 \cdot 10^6$	$4.0 \cdot 10^{11}$	$4.4 \cdot 10^5$	$1.5 \cdot 10^{17}$	$3.7 \cdot 10^9$	$4.8 \cdot 10^{18}$	$4.0 \cdot 10^{11}$	$4.7 \cdot 10^{18}$
FESTO production line	$2.5 \cdot 10^6$	$1.1 \cdot 10^{13}$	$8.7 \cdot 10^{12}$	$1.2 \cdot 10^{17}$	$3.7 \cdot 10^8$	$1.5 \cdot 10^{17}$	$7.1 \cdot 10^{12}$	$9.7 \cdot 10^{16}$

By performing the experiment with- and without applying the variable ordering heuristics, we can analyze how effective the heuristics are at reducing computational effort for each model. We define the ratio in sample mean peak used BDD nodes from the experiments without applying the heuristics to those where the heuristics were applied $\phi_{\mu_G}(f)$

$$\phi_{\mu_G}(f) = \frac{\mu_G(f)\big|_{\text{heuristics applied}}}{\mu_G(f)\big|_{\text{heuristics not applied}}} \tag{5.1}$$

Likewise we can calculate the ratio for BDD operation count $\phi_{\mu_G}(g)$. Table 5.2 shows these ratios for each model.

We observe that the smaller models require about equal computational effort whether the variable ordering heuristics are used or not. However, the variable ordering heuristics are effective for the larger models. For the Production cell model, the heuristics managed to reduce the mean peak used BDD nodes by 95% and the BDD operation count by 96%.

Even though the reduction in computational effort by applying the heuristics is considerable, there is room for improvement. As pointed out in Section 4.1, there is still a large variance in required computational effort when applying the heuristics. We investigate what the cause of this large variance is. As introduced in Section 2.4, the heuristics find a local minimal value for span corresponding to the variable order. Figure 5.2 shows how the minimized span value relates to the computational effort required to synthesize a supervisor for the Production cell and FESTO models. For the Production cell model, the model where applying the variable ordering heuristics yielded the most reduction in computational effort, the heuristic algorithms

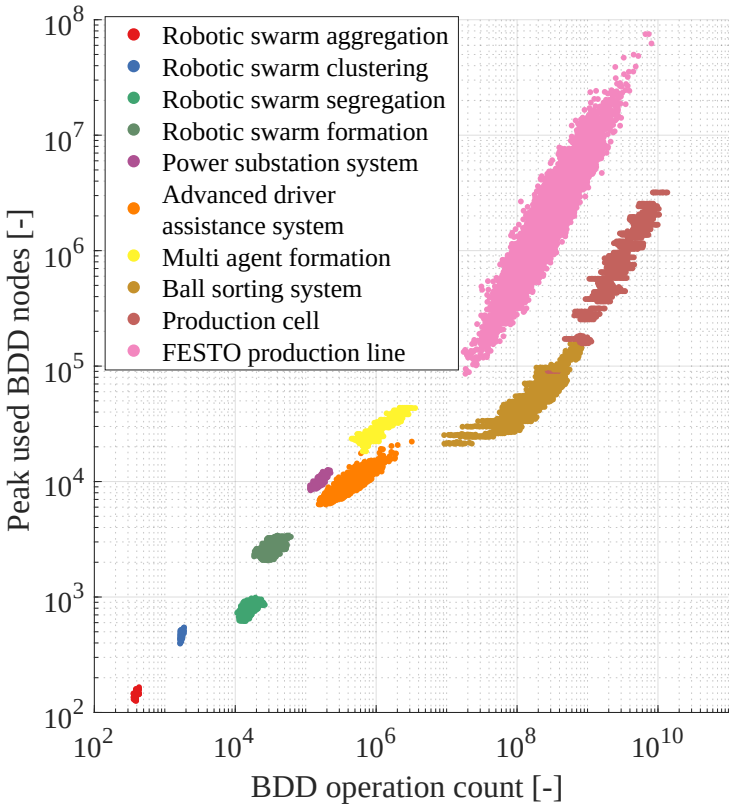
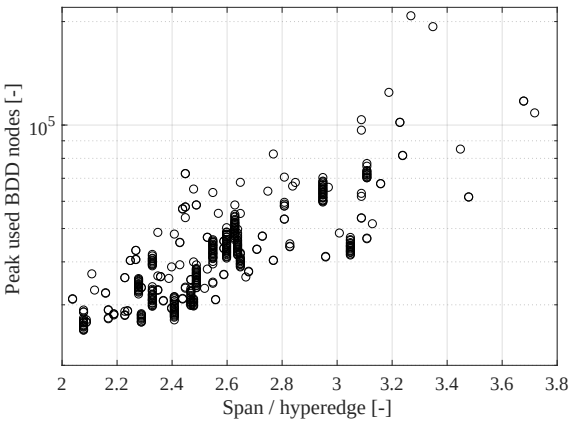


Figure 5.1: *Supervisor synthesis effort for all combinations of 100 event- and 100 variable orders for each model, without applying variable ordering heuristics*

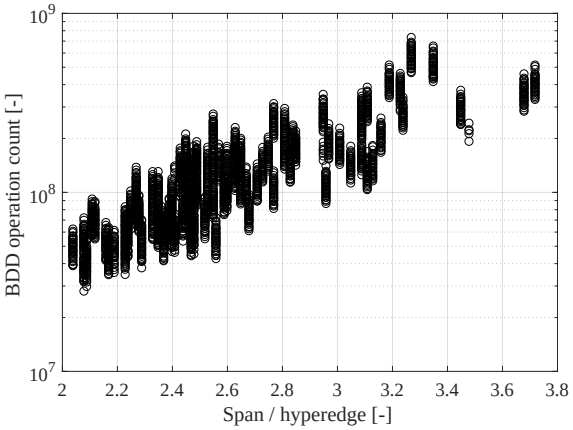
seem to work as intended; A correlation is visible of a lower span value indicating lower computational effort. However, for the FESTO model, we do not observe the same correlation. For this model, it seems nonsensical to base the variable order on the span. Span is seemingly not a good objective function [40] to minimize in order to come up with a good variable order. Another observation is the large variance of the span in the resulting variable orders. Some variable orders have a span much higher than the best-found span. If span were to be a good objective function, more effort could be put towards finding a better variable order. For instance, by instantiating the heuristic algorithms for several initial variable orders and picking the result with the lowest span.

Table 5.2: *Sample mean ratio of computational effort for applying variable ordering heuristics*

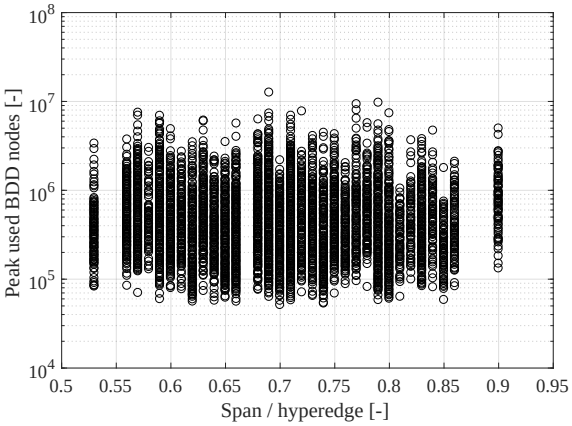
Name	$\phi_{\mu_G}(f)$	$\phi_{\mu_G}(g)$
Robotic swarm aggregation	1.01	1.00
Robotic swarm clustering	1.06	1.06
Robotic swarm segregation	1.00	0.82
Robotic swarm formation	1.01	0.93
Power substation system	0.93	0.91
Advanced driver assistance system	0.93	0.87
Multi agent formation	0.92	0.98
Ball sorting system	0.33	0.09
Production cell	0.05	0.04
FESTO production line	0.25	0.24



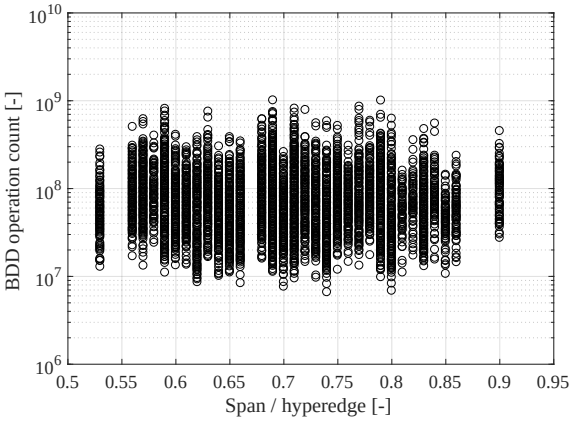
(a) *Production cell correlation between span and peak used BDD nodes*



(b) *Production cell correlation between span and BDD operation count*



(c) *FESTO correlation between span and peak used BDD nodes*



(d) *FESTO correlation between span and BDD operation count*

Figure 5.2: *Correlation between span and computational effort for the Production cell and FESTO models*

6. Conclusions

We have introduced peak used BDD nodes and BDD operation count as metrics that express computational effort of BDD-based supervisor synthesis of EFA. Unlike wall clock time and peak random access memory, the BDD-based metrics are platform independent, deterministic and include no overhead in their measurements. It has been shown that worst case state space size is not an effective indicator for supervisor synthesis effort.

We have shown why and how variable- and event orders influence the supervisor synthesis effort. We performed supervisor synthesis using a large set of random variable- and event orders on 11 models of different sizes. This experiment has shown that the choice of variable- and event orders can influence the supervisor synthesis effort by order of magnitude. The variable order has a larger impact on the supervisor synthesis effort than the event order, but both influences are considerable. The influence of the variable- and event orders increases when considering models that require more supervisor synthesis effort.

Because the BDD-based metrics allow for effective indication of effort for small-scale models, the research can be continued by investigating whether conclusions made by applying certain modeling- or abstraction techniques on small-scale models convert well when implementing them in large-scale models. If this is the case, the BDD-metrics can be used on toy examples to investigate these techniques, instead of being constrained to using large-scale examples.

The current variable reordering algorithms leave some room for improvement. Different methods are to be researched. The metrics presented in the paper can be used for this, as it allows for more precise performance comparisons and comparisons on small-scale models. Preferably a new implementation would provide the same (optimal) variable order regardless of the initial order. The same holds for finding some heuristic ordering algorithm for the events. This way, the variance in computational effort discussed in this paper could be eliminated.

The modeling tool Supremica has a BDD-based supervisor synthesis algorithm for EFA that is similar to CIF's. It can be researched whether the conclusions made in this paper, based on experiments in CIF, also apply to Supremica. For practitioners, it would be interesting to know which of these tools is able to synthesize supervisors with less effort.

References

[1] P. Ramadge and W. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control*, vol. 25, no. 1, pp. 206–230, 1987.

[2] P. Ramadge and W. Wonham, "The control of discrete event systems," *Proc. IEEE*, vol. 77, pp. 81–98, Jan 1989.

[3] M. Skoldstam, K. Akesson, and M. Fabian, "Modeling of discrete event systems using finite automata with variables," in *46th IEEE Conf. Decision and Control*, pp. 3387–3392, Dec 2007.

[4] R. Theunissen, M. Petreczky, R. Schiffelers, D. van Beek, and J. Rooda, "Application of supervisory control synthesis to a patient support table of a magnetic resonance imaging scanner," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, pp. 20–32, Jan 2014.

[5] R. Loose, B. van der Sanden, M. Reniers, and R. Schiffelers, "Component-wise supervisory controller synthesis in a client/server architecture," *14th IFAC Workshop on Discrete Event Syst.*, vol. 51, no. 7, pp. 381 – 387, 2018.

[6] F. Reijnen, M. Goorden, J. van de Mortel-Fronczak, M. Reniers, and J. Rooda, "Application of dependency structure matrices and multilevel synthesis to a production line," in *IEEE Conf. Control Technol. Appl.*, pp. 458–464, Aug 2018.

[7] L. Swartjes, *Model-based design of baggage handling systems.* PhD thesis, Dept. Mech. Eng., Eindhoven Univ. of Tech., Sep 2018.

[8] B. van der Sanden, *Performance analysis and optimization of supervisory controllers.* PhD thesis, Dept. Elec. Eng., Eindhoven Univ. of Tech., Nov 2018.

[9] J. Bastos, *Modular specification and design exploration for flexible manufacturing systems.* PhD thesis, Dept. Elec. Eng., Eindhoven Univ. of Tech., Dec 2018.

[10] W. Wonham, K. Cai, and K. Rudie, "Supervisory control of discrete-event systems: A brief history - 1980-2015," *Proc. 20th IFAC World Congr.*, vol. 50, no. 1, pp. 1791 – 1797, 2017.

[11] S. Akers, "Binary decision diagrams," *IEEE Trans. Comput.*, vol. 27, pp. 509–516, June 1978.

[12] C. Lee, "Representation of switching circuits by binary-decision programs," *The Bell System Tech. J.*, vol. 38, pp. 985–999, July 1959.

[13] A. Vahidi, M. Fabian, and B. Lennartson, "Efficient supervisory synthesis of large systems," *Control Eng. Practice*, vol. 14, pp. 1157–1167, Oct. 2006.

[14] S. Miremadi, B. Lennartson, and K. Akesson, "A BDD-based approach for modeling plant and supervisor by extended finite automata," *IEEE Trans. Control Syst. Technol.*, vol. 20, pp. 1421–1435, Nov 2012.

[15] R. Malik, K. Akesson, H. Flordal, and M. Fabian, "Supremica-an efficient tool for large-scale discrete event systems," *Proc. 20th IFAC World Congr.*, vol. 50, no. 1, pp. 5794 – 5799, 2017.

[16] A. Aziz, S. Taşiran, and R. Brayton, "BDD variable ordering for interacting finite state machines," in *Proc. 31st Annu. Des. Autom. Conf.*, pp. 283–288, 1994.

[17] M. Shoaei, L. Feng, and B. Lennartson, "Abstractions for nonblocking supervisory control of extended finite automata," in *IEEE Int. Conf. Autom. Sci. Eng*, pp. 364–370, Aug 2012.

[18] Z. Fei, S. Miremadi, K. Akesson, and B. Lennartson, "Efficient symbolic supervisor synthesis for extended finite automata," *IEEE Trans. Control Syst. Technol.*, vol. 22, pp. 2368–2375, Nov 2014.

[19] B. van der Sanden, M. Reniers, M. Geilen, T. Basten, J. Jacobs, J. Voeten, and R. Schiffelers, "Modular model-based supervisory controller design for wafer logistics in lithography machines," in *ACM/IEEE 18th Int. Conf. Model Driven Eng. Languages and Sys.*, pp. 416–425, Sept 2015.

[20] S. Miremadi, Z. Fei, K. Akesson, and B. Lennartson, "Symbolic representation and computation of timed discrete-event systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, pp. 6–19, Jan 2014.

[21] L. Ouedraogo, R. Kumar, R. Malik, and K. Akesson, "Nonblocking and safe control of discrete-event systems modeled as extended finite automata," *IEEE Trans. Autom. Sci. Eng.*, vol. 8, pp. 560–569, July 2011.

[22] M. Reniers and J. van de Mortel-Fronczak, "An engineering perspective on model-based design of supervisors," *14th IFAC Workshop on Discrete Event Syst.*, vol. 51, no. 7, pp. 257 – 264, 2018.

[23] D. van Beek, W. Fokkink, D. Hendriks, A. Hofkamp, J. Markovski, J. van de Mortel-Fronczak, and M. Reniers, "CIF 3: Model-based engineering of supervisory controllers," *Tools and Algorithms for the Construction and Anal. of Syst.*, vol. 8413, pp. 575–580, 2014.

[24] F. Reijnen, M. Goorden, J. van de Mortel-Fronczak, and J. Rooda, "Supervisory control synthesis for a waterway lock," in *IEEE Conf. Control Technol. Appl.*, pp. 1562–1563, Aug 2017.

[25] Y. Yang and R. Gohari, "Embedded supervisory control of discrete-event systems," in *IEEE Trans. Autom. Sci. Eng.*, pp. 410–415, Aug 2005.

[26] R. Bryant, "Symbolic boolean manipulation with ordered binary-decision diagrams," *ACM Comput. Surv.*, vol. 24, pp. 293–318, Sept. 1992.

[27] B. Bollig and I. Wegener, "Improving the variable ordering of OBDDs is NP-complete," *IEEE Trans. Comput.*, vol. 45, pp. 993–1002, Sept 1996.

[28] F. Aloul, I. Markov, and K. Sakallah, "FORCE: A fast and easy-to-implement variable-ordering heuristic," *IEEE Great Lakes Symp. on VLSI*, May 2003.

[29] C. Meinel and T. Theobald, *Algorithms and Data Structures in VLSI Design*. Berlin, Heidelberg: Springer-Verlag, 1st ed., 1998.

[30] D. Knuth, "Big omicron and big omega and big theta," *ACM Sigact News*, vol. 8, no. 2, pp. 18–24, 1976.

[31] I. Wegener, "BDDs-design, analysis, complexity, and applications," *Discrete Appl. Math.*, vol. 138, pp. 229–251, Mar. 2004.

[32] F. Somenzi, "Binary decision diagrams," *NATO Sci. Ser., Ser. III*, pp. 303–368, 1999.

[33] Y. Lopes, S. Trenkwalder, A. Leal, T. Dodd, and R. Groß, "Supervisory control theory applied to swarm robotics," *Swarm Intell.*, vol. 10, pp. 65–97, Mar. 2016.

[34] W. Chao, Z. Tang, G. Lin, J. Guo, W. Lin, and S. Yu, "Modular supervisory control of computer based preventing electric mal-operation system for multiple bays in substation," in *IEEE Inform. Technol., Networking, Electron. and Autom. Control Conf.*, pp. 1730–1739, Dec. 2017.

[35] T. Korssen, V. Dolk, J. van de Mortel-Fronczak, M. Reniers, and M. Heemels, "Systematic model-based design and implementation of supervisors for advanced driver assistance systems," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, pp. 533–544, Feb 2018.

[36] K. Cai and W. Wonham, "New results on supervisor localization, with case studies," *Discrete Event Dynamic Syst.*, vol. 25, pp. 203–226, May 2014.

[37] G. Čengić and K. Åkesson, "A control software development method using IEC 61499 function blocks, simulation and formal verification," *IFAC Proc. Vol.*, vol. 41, pp. 22–27, Jan. 2008.

[38] L. Feng, K. Cai, and W. Wonham, "A structural approach to the non-blocking supervisory control of discrete-event systems," *Int. J. Adv. Manuf. Technol.*, vol. 41, pp. 1152–1168, Apr. 2009.

[39] D. Montgomery and G. Runger, *Applied Statistics and Probability for Engineers*. John Wiley and Sons, 2003.

[40] A. J. G. Schoofs, M. H. van Houten, L. F. P. Etman, and D. H. van Campen, "Global and mid-range function approximation for engineering optimization," *Math. Meth. Oper. Res.*, vol. 46, pp. 335–359, Oct 1997.

TU/e Technische Universiteit Eindhoven University of Technology

Declaration concerning the TU/e Code of Scientific Conduct for the Master's thesis

I have read the TU/e Code of Scientific Conduct[i].

I hereby declare that my Master's thesis has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct

Date

08-04-2019

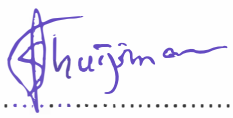
Name

Sander Thuijsman

ID-number

0850788

Signature



Submit the signed declaration to the student administration of your department.

[i] See: http://www.tue.nl/en/university/about-the-university/integrity/scientific-integrity/
The Netherlands Code of Conduct for Academic Practice of the VSNU can be found here also.
More information about scientific integrity is published on the websites of TU/e and VSNU