A Survey of Static Variable Ordering Heuristics for Efficient BDD/MDD Construction

Michael Rice
University of California, Riverside
mrice@cs.ucr.edu

Sanjay Kulhari
University of California, Riverside
skulhari@cs.ucr.edu

Abstract

The problem of finding an optimal variable ordering for Binary Decision Diagrams (BDD) or Multi-Valued Decision Diagrams (MDD) is widely known to be NP-Complete. This paper presents a survey of static heuristic techniques applied to ordering the variables of the BDD/MDD under construction in order to minimize the overall size of the resulting decision diagram.

1 Introduction

Binary Decision Diagrams (BDD) and Multi-Valued Decisions Diagrams (MDD) are commonly used structures for representing Boolean functions and multi-valued functions, respectively. Such decision diagrams are widely used within the domain of system design verification. For example, given two binary or multi-valued circuits, decision diagrams representing the function of each circuit may be used to test and compare the circuits for logical or functional equivalence.

More formally, a Binary Decision Diagram (BDD) is a directed acyclic graph used to represent/encode some Boolean function $f\colon \mathbb{B}^L \rightarrow \mathbb{B}$. As an extension to this concept, a Multi-Valued Decision Diagram (MDD) is a directed acyclic graph used to represent/encode some multi-valued function $f : \{0...k-1\}^L \rightarrow \{0...k-1\}$, based on a given range value $k \in \mathbb{N}$. It is easy to see that BDDs may be considered a special variant of MDDs, where the value of the range k for representing the input and output values for the function is equal to 2.

Given the L possible input variables to encode in each such decision diagram, the directed acyclic graph representation is designed to contain L levels of nodes, such that each variable is represented at a specific level of the graph. Each node on a given level has k outgoing edges to nodes in the next (lower) level of the graph. The final (lowest) level of the graph is represented by k terminal nodes with values in the range $\{0...k\text{-}1\}$.

The function encoded by the decision diagram may be evaluated on a given set of L input variables by traversing the graph, starting from the root (i.e., the highest-level variable in the decision diagram), and choosing the outgoing edge at each node corresponding to the input value of the variable represented at the current level of the graph. This traversal continues until a terminal node is reached. The resulting value of the function is indicated by the value of the terminal node reached along this evaluation path.

A decision diagram is considered to be *ordered* if "each variable is encountered at most once on each path from the root to a terminal" [12]. A decision diagram is considered to be *fully-reduced* if it does not contain any nodes with all k outgoing edges pointing to the same node and does not contain duplicate nodes for a given level in the directed acyclic graph [7]. For consistency, we assume that all decision diagrams discussed within the context of this paper are considered to be ordered and fully-reduced.

When constructing decision diagrams to represent functions for system verification, the size of the constructed decision diagrams can depend heavily on the ordering of the L input variables used to encode the function. Therefore, finding a good variable ordering is crucial to establishing efficient decision diagram representations for encoding such functions. The problem of finding an optimal variable ordering for constructing a minimum-size decision diagram (in terms of the number of nodes needed to encode the function) is known to be NP-complete [5].

The complexity of the variable ordering problem for decision diagram construction has forced researchers to establish many efficient heuristic and meta-heuristic approaches toward establishing near-optimal variable orderings for efficient BDD and MDD construction.

There are generally considered to be two classes of heuristic techniques for establishing efficient variable orderings for decision diagram construction: *static* variable ordering and *dynamic* variable ordering. Static variable ordering techniques attempt to establish the variable ordering prior to constructing the actual decision diagram, while dynamic variable ordering techiques attempt to adjust the ordering online during the actual construction of the decision diagram.

Since static heuristics generate the final variable ordering before construction of the decision diagram even begins, there is no guarantee of good quality solutions from the resulting order. Alternatively, since dynamic variable ordering allows for adjusting the variable order during the actual construction of the decision diagram, they are generally considered more effective in providing efficient orderings; however, they are also typically much more time consuming in practice than the simpler, static heuristics, and thus, are often considered less practical.

The techniques presented in the remainder of this paper will focus solely on static variable ordering heuristics. Many of these techniques have been tested specifically on either BDD variable ordering problems or MDD variable ordering problems, but not necessarily both. However, due to the strong relationship of the two problem variants (as demonstrated in the definition of BDDs and MDDs above), most of these techniques are considered generically applicable to both BDD and MDD construction and will be treated as such within this paper.

The remainder of this paper is structured as follows. Section 2 presents a survey of some of the more common and efficient heuristic and meta-heuristic approaches used for establishing variable orders in decision diagram construction. The first part of Section 2 focuses on some of the simpler and more straightforward heuristics for variable ordering, while the remainder focuses more deeply on the details behind several of the more complex static heuristic techniques applied to this problem using *variable-span* and *event-span* metric optimization heuristic techniques. Section 3 presents the final conclusion of this paper.

2 Literature Review

Much research has been carried out on devising heuristic and meta-heuristic approaches for establishing near-optimal variable ordering for BDD/MDD construction. This section presents some of these techniques, grouped into subsections based on their general approach.

2.1 Topological Ordering Heuristics

Many of the originally researched heuristic approaches for variable ordering in decision diagram construction were focused on simple topological properties of the functional circuit being represented. For example, the research of [3], [11], and [14] establish variable ordering methods based on variants of depth-first and/or breadth-first search, which proceed from the primary outputs of the circuit to be represented and order the primary input variables based on the order they are visited in the depth-first or breadth-first traversal, respectively. The intuition behind these simple heuristics is that input variables that are *topologically* close together within the circuit should be relatively close together within the variable ordering for the resulting decision diagram.

Such simple topological ordering heuristics have been shown to work quite well for variable ordering on tree-like circuits [11]. However, the heuristics do not work well on all circuits, given that most circuits to be represented in practice are not very tree-like [13].

Due to the simplicity of these heuristics and their general applicability to the variable ordering problem, it is often the case that these topological ordering heuristics are presented as a baseline technique for comparison against newly-developed variable ordering heuristics in experimental analysis (e.g., [3]).

2.2 Influential Ordering Heuristics

One of the most intuitive approaches toward devising practical heuristics for variable ordering is to establish variable orders where the most influential of the primary inputs to the function/circuit are placed earlier on in the ordering. The general reasoning behind this approach is that variables placed later on in the ordering encode the functions of the variables preceding them in the ordering, based on the top-down evaluation procedure of the resulting decision diagram (as discussed briefly in the Introduction section). By placing the more influential variables earlier on in the ordering, all subsequent variables in the ordering will encode the functions represented by these more influential variables, hopefully leading to a reduced number of overall nodes in the resulting decision diagram.

2.2.1 Fanin Heuristic

One of the heuristic approaches based on influential ordering is the Fanin heuristic [15]. The Fanin heuristic is another variant of depth-first search heuristic (similar to the topological ordering heuristics mentioned above) that establishes a variable order by beginning a depth-first search from the primary output variable(s) and choosing the next node in the circuit to explore as the node which has the deepest sub-tree in the circuit. This heuristic depth-first search proceeds until all primary input variables have been placed in the order. The idea behind this approach is that primary inputs deeper within the circuit tend to be more influential than those primary inputs that are relatively shallow within the circuit, and should be placed relatively earlier on in the variable ordering.

2.2.2 Sub-Graph Complexity

Another novel influential ordering approach is developed in [18], in which the heuristic proceeds by iterating over all of the primary input variables from the original circuit and setting each primary input variable to a constant value (e.g., 0 or 1, in the case of Boolean circuits) to simplify the original circuit graph. By setting the (current) input variable to a constant value, the heuristic can simplify the original circuit graph by removing the inputs and gates which are no longer necessary in the circuit, given the current constant value assigned to the (current) primary input. This process is iteratively repeated for each primary input variable, and the variable whose constant-value assignment achieves the smallest sub-graph upon simplification is assigned to be the next variable in the order. This variable is removed from the circuit graph (using the constant-value assignment that resulted in the minimal sub-graph), the circuit graph is simplified, and the process repeats as before until all variables are ordered. The intuition behind this ordering heuristic is that the variables whose removal (or more correctly, constant-value assignment) results in the smallest remaining circuit sub-graphs must therefore be the most heavily-influential variables from the original circuit graph.

2.2.3 Dependent Count

The Dependent Count heuristic [12] is another influential ordering heuristic which seeks to order the primary input variables based on the number of primary output variables they ultimately influence within the circuit. For example, a primary input variable whose value affects two primary output variables is considered more influential within the function than a primary input variable whose value affects only one primary output variable. Using this concept of influence, the primary input variables are ordered in decreasing value based on the total number of primary output variables they affect within the circuit.

2.3 Priority Ordering Heuristics

Another simple variable ordering approach is to order the variables based on some externally-defined notion of priority amongst the variables (both primary input and primary output variables). The heuristics themselves do not care how or why the variable priority is established. Their goal is merely to attempt to preserve this established priority order as best as possible in the resulting variable ordering.

2.3.1 Variable Appending

The simplest and most straightforward heuristic of these techniques is called the Variable Appending heuristic [13]. In the Variable Appending heuristic, the algorithm is given the primary outputs in some pre-defined priority order, and, for each primary output in this order, it establishes the primary inputs that affect this primary output and orders them based on a pre-defined priority order for the primary inputs. If the primary input is already listed in the variable order built thus far in the algorithm, it is skipped; otherwise, it is simply appended to the current ordering.

2.3.2 Variable Interleaving

The research in [13] compares this simplistic appending heuristic to another heuristic approach they propose called Variable Interleaving. The Variable Interleaving heuristic is similar in nature to the appending heuristic, except that it attempts to interleave the primary input variables that occur in multiple primary output lists. The heuristic proceeds as follows.

The Variable Interleaving heuristic begins by constructing an empty, *global* variable order list. Just as in the Variable Appending heuristic, the Variable Interleaving algorithm is given the primary outputs in some pre-defined priority order. For each primary output in this order, it establishes the primary inputs that affect this primary output and orders them into a *local* list based on a pre-defined priority order for the primary inputs. For each input variable in the *local* list of the primary output, if the primary input is already listed in the *global* variable list built thus far in the algorithm, it is skipped; otherwise, it is inserted into to the current ordering of the *global* variable list immediately after the input variable that it appears after in the *local* variable list. This process proceeds until all primary inputs have been inserted into the *global* variable order list.

2.4 EVAL Meta-Heuristic

The EVAL meta-heuristic has been presented in [12] for utilizing some of the simpler heuristics presented in previous subsections. The idea behind the EVAL meta-heuristic is as follows. Given a list of static, black-box heuristics for variable ordering, the EVAL metaheuristic runs each static heuristic in parallel until the search space generated by the partial decision diagram constructed thus far using each heuristic's variable ordering reaches a given node limit or the construction algorithm reaches a given time limit. Once a limit is reached, the EVAL meta-heuristic chooses the most promising static heuristic approach based on the partial decision diagram created thus far using the ordering generated by that heuristic.

The criteria for the "most promising" heuristic is defined as the heuristic whose variable ordering has generated the partial decision diagram that encodes the most gate functions of the original circuit to be represented. Once the most promising heuristic is chosen, the entire decision diagram is then constructed using the chosen heuristic.

Experiments in [12] show that the EVAL metaheuristic is much better at reducing overall memory overhead during the construction of decision diagrams than is typically possible using only one of the static, black-box heuristics.

2.5 Metric Optimization Heuristics

The remainder of this paper will focus more deeply on another categorical subset of variable ordering heuristics: metric optimization heuristics. Metric optimization heuristics may be considered as any variable ordering heuristic technique that seeks to minimize (or maximize) some pre-defined metric on the structural or functional properties of a given variable ordering permutation.

While some of the heuristic approaches presented in earlier sections may also actually be classified as metric optimization heuristics (e.g., the Dependent count heuristic [12]), the metrics under consideration in this section are generally much more complex than some of the simpler techniques presented earlier, and are based on the concepts of *variable span* metrics and *event span* metrics (described further in the sub-sections below).

As will be shown below, mathematically determining/defining a metric function to appropriately summarize a Boolean or multi-valued circuit or function can help in determining an appropriate variable ordering for decision diagram construction. The chosen metric should be defined so as to capture enough relevant information about the circuit/function under consideration to prove applicable in establishing good variable orders.

So far in this paper, we have only considered the impact of the individual primary input variables on the ordering of the decision diagram. However, a Boolean or multi-valued circuit/function may be seen as consisting of both variables and *events.* A variable is generally seen as a primary input to the circuit or function (what we have considered thus far), while an *event* defines the point of functional interaction between two or more input values in determining some composite function (e.g, the equivalent of a Boolean or multi-valued gate within a circuit function or a transition within a Petri net). From a given circuit/function, depending on the presence of variables at different levels, one can generate a metric whose behavior can be correlated with the variable ordering that can be used to more efficiently construct a decision diagram. Similarly, the event information can also be utilized to generate a metric that can be studied for variable ordering.

In the following sub-sections, we are going to compare two slightly different, yet comparable, high-level techniques for determining variable ordering. The first approach considers metrics based on the concept of *variable span*, while the other approach considers metrics based on the concept of *event span.* The central tenet behind both of these generic approaches is to define a metric which serves to better cluster together variables and/or events, respectively, within the resulting ordering that are somehow closely correlated to one another within the original circuit or function.

We are briefly going to summarize the results of the papers based on minimum variable span that talk about the smallest communication graph [4] and normalized average lifetime [16] metrics. We will then discuss several event span metrics based on variable weighting by affected events [9][10], a hybrid combination of multiple event span metrics [17], weighted event locality [8], and the MINCE [1][3] and FORCE [2] heuristics, which also propose metrics based on event locality, but take a slightly different approach by utilizing the information from a *conjunctive normal form*

(CNF) representation of the circuit under consideration.

2.5.1 Smallest Communication Graph

The *variable span* metric from [4] attempts to find an optimal variable ordering to represent interacting finite state machines as BDDs. The transitions in the state machines are viewed as logic circuits and a decision diagram can be created to explore the state space. The metric optimization heuristic proposed uses the communication structure between state machines to determine an upper bound on the size of BDD for a given variable ordering.

Two specific upper-bounding metrics are defined in this research and used for optimization of variable ordering. The upper bound defined by either such metric can then be used to help decide if the variable ordering is efficient for BDD construction or not.

For a given ordering $x_{\sigma(1)}, ..., x_{\sigma(n)}$, the metric S^{σ} is defined as follows:

$$S^{\sigma} = \sum_{i=0}^{n} (2^{2(|x_{\sigma(i)}|)} \cdot 2^{\omega_f^{\sigma}(i+1)} \cdot 2^{2^{\omega_r^{\sigma}(i+1)}})$$

...such that $\omega_f^{\sigma}(k)$, and $\omega_r^{\sigma}(k)$ are the number of bits communicated from two partitioned sets of state machines (in the forward and backward communication structure, respectively, with the partition occuring at position $\sigma(k)$ in the ordering).

Another, simpler, yet looser, upper bound on the number of BDD nodes for the same ordering as above is the metric M^{σ}, defined as follows:

$$M^{\sigma} = n \cdot c \cdot 2^{\omega_f^{\sigma}} \cdot 2^{2^{\omega_r^{\sigma}}}$$

...such that ω_f^{σ}, and ω_r^{σ} are the *maximum* number of bits communicated from two partitioned sets of state machines (in the forward and backward communication structure, respectively) and c is the "maximum number of state bits in some machine" [4].

Experimental results were carried out using a branch-and-bound method to calculate the optimum ordering for minimizing the metrics S^{σ} and M^{σ} (defined above) and were compared to a simple *k-lookahead* greedy heuristic, which chooses the next k variables in the order to be the k-length permutation of remaining variables that minimizes the partial metric cost thus far. Results showed that using the *k-lookahead* greedy heuristic, with $k = 2$, for minimizing the S^{σ} metric resulted in the most near-optimal solutions with "negligible" running time. It was also reported that minimizing the communication graph between finite state machines, while simultaneously allowing some "heuristic interleaving" (similar to the Variable Interleaving heuristic of [13]) of the variables seems to be the most practical approach altogether [4].

2.5.2 Normalized Average Lifetime

The *normalized average lifetime* metric [16] is a *variable span* metric proposed for image computation. Image computation is essentially the process of finding all of the next states from a given state based on a set of transitions. To determine the next set of states, this paper proposes a hybrid approach based on conjoining the partitions of the transition relation and recursive case splitting on the possible input or present state variable or the values of the next state variable. The transitions and states can be defined as Boolean formulas that can be represented as a binary decision diagram.

Given an ordered set of functions $(f_1, ..., f_n)$ (e.g., the transitions) and an ordered set of variables $x_1, ..., x_m$ upon which the functions may depend, the dependence matrix is defined as a matrix D with n rows and m columns such that $D_{i,j} = 1$ if function f_i depends on variable x_j, and $D_{i,j} = 0$ otherwise.

The normalized average lifetime metric is then defined as the following:

$$\lambda = \frac{\sum_{i \leq j \leq m}(h_j - l_j + 1)}{n \cdot m}$$

...such that n is the number of rows/transitions in the matrix and m is the number of columns/variables, h_j is the largest index i in column j such that $D_{i,j} = 1$, and l_j is the smallest index i in column j such that $D_{i,j} = 1$. The authors utilize this normalized average lifetime metric as a means of interpreting the current "clustering" produced by the structure of the dependence matrix associated with a given variable ordering when deciding whether to split or conjoin in their hybrid approach to image computation. The algorithm detailed in the paper is set to conjoin when $\lambda \leq 0.5 + \epsilon_1$ and to split when $\lambda > 0.5 + \epsilon_1$. Experiments show that setting $\epsilon_1 = 0.1$ provides good overall results for image computation. It should be noted that only the variables are being re-ordered during the process of image computation. The transitions themselves are clustered only once at the beginning of the computation and this ordering is maintained throughout the remainder of the variable ordering procedure.

2.5.3 Variable Weighting

The first *event span* method to be discussed in this paper is a straightforward approach based on variable weighting [9][10] called the *weighting heuristics*. This research utilizes the *weighting heuristics* algorithm for establishing a variable ordering for BDD construction for the purposes of performing *reachability analysis* on the state space of a transition relation; that is, computing the set of states of the system that are reachable from the given initial configuration. The goal is to be able to construct small enough BDDs in order to perform the reachability analysis within practical space bounds.

The idea behind the *weighting heuristics* algorithm is to establish a weight value for each of the primary input variables, based on the number of events they participate in within the transition functions, and order the variables based on this weighting. In particular, if a given primary input variable is an input parameter to a total of k transition functions (i.e., events), then it will be assigned a weight value of k. Once all input variables have been weighted, they are then ordered in decreasing order of this weighting.

Experimental analysis compared the *weighting heuristics* against several other variable ordering heuristics, including the Variable Appending and Variable Interleaving heuristics [13]. Results proved comparable against the other tested heuristics in the static ordering case, with the *weight-*

ing heuristics showing even more improvement over other heuristics when a dynamic, re-ordering of the variables was allowed between phases of the reachability analysis algorithm.

2.5.4 Hybrid Metric Optimization

A complex hybrid *event span* ordering approach is presented in [17]. In this research, the authors present a heuristic algorithm that takes into account several similar "influence" metrics for each primary input of the function/circuit to be represented. The authors present several versions of this algorithm, each of which takes into account multiple different metrics, including the number of paths for a given primary input variable to all reachable primary outputs in the circuit graph (similar in nature to the Dependent Count heuristic [12], presented earlier), the total number of gate nodes encountered along all paths of the circuit for each primary input variable (similar in nature to the *weighting heuristics* [9][10], presented above), the sum of edge counts along the shortest paths for a given primary input variable to all other primary input variables (assuming the circuit graph is undirected), and the resulting all-pairs shortest path matrix for all primary input variables. These metrics are used together to establish and compare alternative orders for decision diagram construction.

It should be noted that, like the other metrics presented in this paper, these metrics by themselves are perfectly applicable for static variable ordering heuristics. However, the authors of this paper have utilized these metrics for a more dynamic variable ordering approach in their actual experimentation. The most complex of the algorithms in [17] is compared against three other dynamic variable ordering algorithms and is shown to produce a mininum number of nodes in its decision diagram constructions for almost 61% of the tested circuits.

2.5.5 MINCE

Another *event span* metric, called the MINCE (Min. Cut Etc.) heuristic, is presented in the work of [1][3]. In this research, the authors devise an algorithm for decision diagram variable ordering based on the following intuition. If the heuristic can partition the variables into groups with minimal functional correlation between variables in separate groups, then such a function is more likely to have a decision diagram graph representation with small edge cuts between the partitioned variables. Such decisions diagrams with small edge cuts must implicitly have relatively few overall edges in the graph and therefore must also have relatively few overall nodes (since each node in the directed acyclic graph representation of a decision diagram can only have a constant number, k, of outgoing edges).

The research in [1][3] shows that a given functional circuit can easily be represented in a *conjunctive normal form* (CNF) formula, where the CNF formula for the circuit represents the conjunction of the formulas for each of the gates (i.e., *events*) of the circuit. The authors interpret the CNF formula as a hypergraph, to be recursively partitioned using previously-known heuristic algorithms for solving the *balanced min-cut hypergraph partitioning* problem (applied recursively to define a min-cut bisection). The authors present the following metrics for optimization:

$$TotalSpan = \sum_{e \in E} span(e) = \sum_{i=0}^{|V|-1} cut(i)$$

$$AverageSpan = \frac{\sum_{e \in E} span(e)}{|E|}$$

The problem of finding a good variable ordering for decision diagram construction is now reduced to minimizing the average span of the recursive partitioning. MINCE uses a tool called CAPO [6] to minimize the average clause span of the CNF hypergraph. CAPO determines the cuts and partitions in the hypergraph and, for each partition, the average span is determined. The partitioning for which the metric is minimized is selected and the variable ordering is determined using this partitioning.

Applying such a metric optimization heuristic recursively results in a final linear ordering for the variables that compared well against many of the other, simpler topological ordering heuristics presented earlier (e.g., DFS and BFS orderings). The MINCE algorithm was able to construct more BDDs from large circuits in generally less time than some of these other heuristic techniques.

2.5.6 FORCE

The FORCE heuristic [2] is presented as an enhancement over the MINCE heuristic [1][3] (discussed above). Compared to MINCE, the FORCE algorithm is shown to be orders of magnitude faster and less memory intensive, while remaining competitive for BDD variable ordering. FORCE also doesn't rely on any external tools, such as the use of the CAPO tool [6] in MINCE, making it much simpler to integrate with other applications.

The idea behind the FORCE algorithm is simple: the algorithm computes the *forces* acting upon each variable and displaces the variables in the direction of the *forces* acting upon them. In FORCE, similar to MINCE, a CNF formula is viewed as a hypergraph, where the formula's variables correspond to vertices and clauses corresponds to hyperedges. The FORCE algorithm determines two values during execution and iteratively uses them to order the variables. In each iteration, the FORCE algorithms calculates the first of these two values, called the *center of gravity* (COG), for each hyperedge e:

$$COG(e) = \frac{\sum_{v \in e} l_v}{|e|}$$

...where l_v is the location of vertex v under the given ordering and $|e|$ is the number of vertices connected to hyperedge e.

The next step of the FORCE algorithm is to calculate the new tentative location, l'_v, of each variable along the real line based on the average of their *center of gravity*:

$$l'_v = \left(\sum_{e \in E_v} COG(e)\right) / |E_v|$$

Once all variables have been assigned a new tentative location, l'_v, then these variables are ordered linearly by this value and assigned integer indices based on their relative ordering (i.e., a bijective mapping from relative positions along the real line to the values in $1, ..., n$ is carried out). This procedure continues to iterate until either there is either no longer any improvement in some specified ordering evaluation metric, such as the *TotalSpan* metric presented earlier for the MINCE algorithm, or the number of iterations reaches a limit of $clog|V|$ for some constant c.

The running time of FORCE is shown to be: $O((|C| + |V|log|V|)log|V|)$

The use of the FORCE heuristic is mainly intended for SAT solvers and an initial variable ordering can prove to be very efficient for further SAT operations. However, experiments show that FORCE is generally expected to produce slightly inferior, yet generally competitive, variable orderings when compared to MINCE, while the time spent on finding a variable ordering is generally *much* less due to its great speed improvements over MINCE.

2.5.7 Weighted Event Span

Another *event span* metric, parameterized by moment, was proposed in [8] for variable ordering in terms of events and state variables.

Similar to the dependence matrix presented in [16] used for optimizing the normalized average lifetime metric, the research in [8] defines a boolean dependence matrix $A \in \{0,1\}^{|\{K,...,1\}| \times |\varepsilon|}$, where K is number of state variables $(i_K, ..., i_1)$ and $|\varepsilon|$ is the total number of events, such that $A_{k,e} = 1$ if the variable at level k depends on event e (and vice-versa), and $A_{k,e} = 0$ otherwise.

Given a permutation, π, of variable ordering for establishing the dependence matrix A^π on this permutation, the following metrics are proposed:

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

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

...such that e is an event,...

$$Top_\pi(e) = max_\pi\{k : A^\pi_{k,e} = 1\}$$

...is the highest level that depends on event e for permutation π, and...

$$Bot_\pi(e) = min_\pi\{k : A^\pi_{k,e} = 1\}$$

...is the lowest level that depends on event e for permutation π.

The *Normalized Event Span (NES)* metric represents the normalized average (dependence) span of all events (this is similar in nature to the normalized average lifetime metric). The *Weighted Event Span (WES)* metric is a further generalization of the NES metric, parameterized by a moment i, such that larger values of moment i place higher importance on the actual *location* of variables in the ordering (relative to the average ordering position $K/2$), with less emphasis on clustering. Using this metric, variables ordered closer to the top of the decision diagram variable ordering are given a higher weighting to denote the more costly operations on MDD nodes at higher levels in the ordering.

Experiments were carried out to find the smallest MDD sizes for all permutations of a small set of variables. The WES metric for the first three moments (i.e., $WES^{(0)}$, $WES^{(1)}$, and $WES^{(2)}$) of each of these smallest MDDs was then calculated and compared to an experimental run focused on minimizing the WES metric for these particular moments. The experiments seemed to suggest an obvious correlation between minimal WES metric values for a given ordering and minimal MDD sizes for the same ordering. The tests indicate that minimizing $WES^{(0)}$

results in multiple potential minimums, making it difficult for the minimization algorithm to make a distinct and obvious choice amongst the possible orders which might also lead to minimal MDD size, whereas the other moment metrics seem to have fewer overall minimums to choose from, resulting in potentially better direction for finding the minimal MDD ordering. The metric $WES^{(1)}$ is suggested as the best overall moment for this metric.

Additional experiments were carried out to compare the $WES^{(1)}$ metric against the NES metric (which is essentially equivalent to the $WES^{(0)}$ metric) in generating the state space using both a BFS and saturation algorithm. Overall results indicate that the $WES^{(1)}$ metric is generally a more reliable metric for minimizing MDD size in this process.

3 Conclusion

This paper has presented a survey on many varied static heuristics applied to the NP-complete problem of variable ordering for the construction of minimal decision diagrams (both BDD and MDD variants). The general impression of this survey seems to suggest an initial research focus on the basic topological properties of the circuits/functions under consideration for representation as a decision diagram, whereas more recent research has taken this concept further into the domain of clustering based on both the primary input variables as well the underlying events of the circuit/function to be modeled.

Clustering approaches which take into account the events of the function in correlation with the actual variables seem to be generally more effective in reducing overall sizes of resulting decision diagrams, as the *event span* metrics seem to promote a more holistic summarization of the properties of the circuits/functions. Additional future metrics that take into consideration not only the clustering of events and variables, but also the positioning of those variables in the resulting ordering (e.g., whether at the top of the ordering, towards the middle, or at the bottom), similar to [8], may also result in a more practical overall approach to variable ordering. Further research in defining such new metrics for variable ordering seems to be the most promising area of related research for static variable ordering heuristics.

References

[1] F.A. Aloul, I.L. Markov, and K.A. Sakallah, *Faster SAT and Smaller BDDs via Common Function Structure*. Proc. of the International Conference on Computer Aided Design, 2001.

[2] F.A. Aloul, I.L. Markov, and K.A. Sakallah, *FORCE: A Fast and Easy-To-Implement Variable-Ordering Heuristic*. Great Lakes Symposium on VLSI: pp.116-119, 2003.

[3] F.A. Aloul, I.L. Markov, and K.A. Sakallah, *MINCE: A Static Global Variable-Ordering Heuristic for SAT Search and BDD Manipulation*. J. UCS, 10(12): pp.1562-1596, 2004.

[4] A. Aziz, S. Tasiran, and R.K. Brayton, *BDD Variable Ordering for Interacting Finite State Machines*. In 31st ACM/IEEE Design Automation Conference (DAC), 1994.

[5] B. Bollig, et al., *Improving the Variable Ordering of OBDDs is NP-Complete.* IEEE Transactions on Computers, 1996.

[6] A. Caldwell, A. Kahng, and I. L. Markov, *Can Recursive Bisection Produce Routable Placements?* In Proceedings of the Design Automation Conference (DAC), pp.477-482, 2000.

[7] G. Ciardo, *CS246 - Notes on Decision Diagrams.* `http://www.cs.ucr.edu/~ciardo/teaching/CS246/CS246dds.pdf`.

[8] G. Ciardo, R. Siminiceanu, *New Metrics for Static Variable Ordering in Decision Diagrams.* In TACAS 2006, LNCS 3920, pp. 90-104, 2006.

[9] D. Deharbe and J. Vidal, *Optimizing bdd-based verification analysing variable dependencies.* In XIV Symposium on Integrated Circuits and System Design (SBCCI'01), pp. 64-69. Computer Society Press, 2001.

[10] D. Deharbe, J. Vidal, and D. Borrione, *Improving static ordering of bdds for reachability analysis.* In IEEE/ACM 11th International Workshop on Logic Synthesis, pp. 1-5, 2002.

[11] R. Drechsler, *Verification of Multi-Valued Logic Networks.* Multiple-Valued Logic - An International Journal, 3(1): pp.77-88, 1998.

[12] R. Drechsler, *Evaluation of Static Variable Ordering Heuristics for MDD Construction.* ISMVL 2002: pp.254-260, 2002.

[13] H. Fujii, G. Ootomo, and C. Hori, *Interleaving Based Variable Ordering Methods for Ordered Binary Decision Diagrams.* International Conference on CAD: pp.38-41, 1993.

[14] M. Fujita, H. Fujisawa, and Y. Matsunaga, *Variable Ordering Algorithms for Ordered Binary Decision Diagrams and their Evaluation.* IEEE Trans. Computer Aided Design, vol. 12: pp.6-12, 1993.

[15] S. Malik, et al., *Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment.* International Conference on CAD: pp.6-9, 1988.

[16] I.-H. Moon, et al. *To split or to conjoin: the question in image computation.* In Proceedings of the 37th Conference on Design Automation (DAC-00), pp. 23-28, 2000.

[17] P.W.C. Prasad, et al., *Binary Decision Diagrams: An Improved Variable Ordering Using Graph Representation of Boolean Functions.* International Journal of Computational Science, 1(1): pp.1-7, 2006.

[18] M. Raseen, K. Thanuduki, *ROBDD Optimization Using Sub Graph Complexity.* International Journal of Computer Science and Network Security (IJCSNS), VOL.8 No.8, 2008.