Data structures for quantum circuit verification and how to compare them

Vinkhuijzen, L.T.

Chapter 6

The Power of Disjoint Support Decompositions in BDDs

The relative succinctness and ease of manipulation of different languages to express Boolean constraints is studied in knowledge compilation, and impacts areas including formal verification and circuit design. We give the first analysis of Disjoint Support Decomposition Binary Decision Diagrams (DSDBDD), introduced by Bertacco, which achieves a more succinct representation than Binary Decision Diagrams (BDDs) by exploiting Ashenhurst Decompositions. Our main result is that DSDBDDs can be exponentially smaller than BDDs.

This chapter contributes to Research Question 3:

> **Research question 3.** *Which classical decision diagrams might be effective for the analysis of quantum algorithms, if they were suitably adapted?*

Our answer is that we find the results of this chapter encouraging: the (classical) DSDBDD provides an example of a minor and conceptually simple modification of a DD which nevertheless leads to exponential improvements. Can such lessons inspire us to design a "quantum DSDBDD," which effects exponential improvement, too? Indeed, we formulate specific ways to draw inspiration from the DSDBDD to design such a "quantum DSDBDD" in Section 8.3. In the same chapter, we speculate about several

other opportunities to improve quantum decision diagrams.

6.1 Introduction

Decision Diagrams are data structures for the representation and manipulation of Boolean functions. They are used for probabilistic reasoning [28, 91], verification [162, 221, 339], circuit design [300, 301, 347, 372] and simulation of quantum computing [242, 333, 337]. Since Bryant [67] popularized the Binary Decision Diagram (BDD), there has been a proliferation of different decision diagrams which use different architectures, e.g., ZDD [229], TBDD [328], CBDD [68], SDD [96], uSDD [322], FBDD [129]. Darwiche and Marquis [97] analytically compare the succinctness and tractability of manipulation operations (e.g., computing the logical OR of two functions) of these different diagrams and other representations, such as CNF, resulting in a *knowledge compilation map*. In particular, they elucidate the inherent tradeoff between succinctness and tractability: Some diagrams can be exponentially more succinct, but do not admit efficient manipulation and/or query operations, or vice versa (e.g., d-DNNF [97] strictly contains DDs and allows model counting in polynomial time, but no efficient algorithm for computing the logical OR is known; and SDDs can be exponentially more succinct than BDDs [58]).

The Disjoint Support Decomposition BDD [46, 264] (DSDBDD*) augments a BDD with disjunctive decompositions (sometimes called Ashenhurst-Curtis decompositions [8, 22]). They are canonical like BDDs and support the same queries and operations as BDDs (model counting, conjunction, negation, etc.). DSDBDDs have so far been deployed in only few applications, mostly in circuit verification [265]. In order to know whether efforts to deploy them elsewhere are likely to be fruitful, we make an initial step towards placing DSDBDDs on the knowledge compilation map. Our main result is that DSDBDDs can be exponentially smaller than BDDs. To this end, we give a function that yields the separation, drawing on the theory of expander graphs to show that its BDD cannot be small. As corollaries, we also clarify the relation between other languages. Finally, we also point out some open questions.

*No name has been given to this diagram, so we use DSDBDD in accordance with conventions in the literature.

6.2 Background and related work

A decision diagram is a data structure used to represent and manipulate Boolean functions. For an accessible exposition of decision diagrams, the interested reader may consult Section 2.3. For the purposes of this chapter, it suffices to give the following definition of a Binary Decision Diagram (BDD):

Definition 6.1 (Binary Decision Diagram (BDD))**.** A BDD is a rooted, directed acyclic graph. It has two leaves, labeled TRUE (or 1) and FALSE (or 0). A non-leaf node is called a *Shannon node*; it is labeled with (the index of) a variable and has two outgoing edges, called the *low edge* and the *high edge*. Each node v represents a Boolean function $[\![v]\!]$, defined inductively as follows. In the base case, the TRUE and FALSE leaves represent the constant functions $[\![\text{TRUE}]\!] = 1$ and $[\![\text{FALSE}]\!] = 0$,

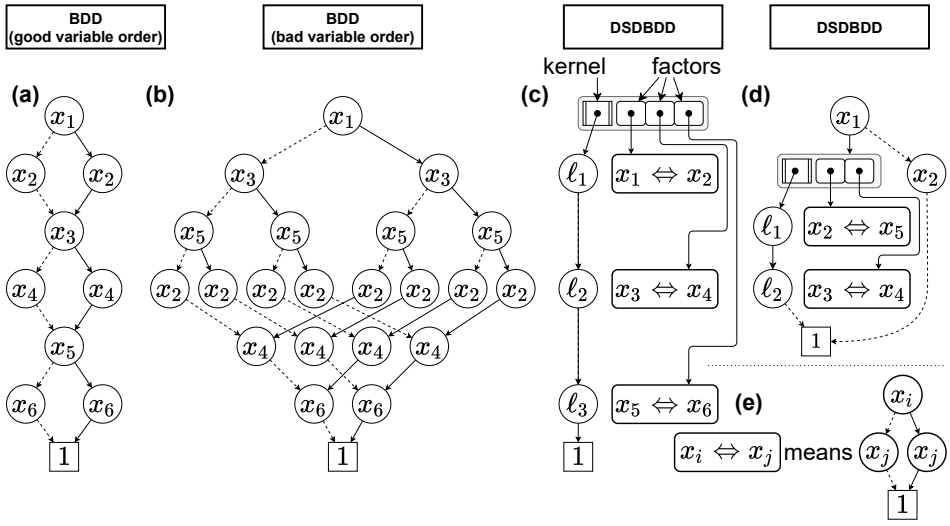

Figure 6.1: (a) and (b): BDDs for the function $f \triangleq (x_1 \Leftrightarrow x_2) \wedge (x_3 \Leftrightarrow x_4) \wedge (x_5 \Leftrightarrow x_6)$. Low (high) edges are drawn as dotted (solid) arcs. The FALSE Leaf and arcs which go to the FALSE Leaf are not drawn. These two BDDs use different variable orders: (a) uses $x_1 < x_2 < x_3 < x_4 < x_5 < x_6$, whereas (b) uses $x_1 < x_3 < x_5 < x_2 < x_4 < x_6$, which is why (b) is much bigger than (a). (c): a DSDBDD for the same function. The root node is a decomposition node, whose kernel and factors are indicated. For sake of clarity and compactness, in several parts of the figure we have "collapsed" the small BDD representing $x_i \Leftrightarrow x_j$ by drawing it as a single rectangle (e). (d): a DSDBDD which represents the function $f = \neg x_1 \wedge \neg x_2 \vee x_1 \wedge ((x_2 \Leftrightarrow x_5) \wedge (x_3 \oplus x_4))$ and whose root node is not a Decomposition node.

respectively. If a Shannon node v is labeled with variable x and has low edge to v_0 and high edge to v_1, then it represents the function $[\![v]\!] \triangleq \neg x \wedge [\![v_0]\!] \vee x \wedge [\![v_1]\!]$. A BDD is *ordered* if, on each path from the root to a leaf, each variable appears at most once and always in the same order. Two nodes u, v are called *equivalent* if they represent the same function, $[\![u]\!] = [\![v]\!]$. A BDD is *reduced* if there are no equivalent nodes. $\diamond$

Figures 6.1(a) and (b) show examples of BDDs. These two BDDs represent the same function, $f \triangleq (x_1 \Leftrightarrow x_2) \wedge (x_3 \Leftrightarrow x_4) \wedge (x_5 \Leftrightarrow x_6)$. They have a different shape, because they employ different variables orders, $x_1 < x_2 < x_3 < x_4 < x_5 < x_6$ and $x_1 < x_3 < x_5 < x_2 < x_4 < x_6$, respectively. In fact, if we generalized the functions and variable orders from $n = 6$ to $n > 6$, the corresponding BDD would stay linearly sized in the first case, but in the latter would become exponentially sized, in the number of variables. The effect of variable orders in BDDs is well known [56].

In the figure, the value $f(x)$ of an assignment x can be found by traversing the diagram as follows. One starts at the root node. A node is labeled with a variable x_i; if $x_i = 0$, we traverse the low (dotted) edge; otherwise, if $x_i = 1$, we traverse the high (solid) edge, until we arrive at a Leaf. To avoid cluttering the diagram, edges to the FALSE Leaf are not drawn in the figure.

A reduced and ordered BDD (ROBDD) is a canonical representation of its corresponding Boolean function [67]. From now on, we assume all BDDs are ROBDDs. Bryant [67] observed that such BDDs can be queried and manipulated in polynomial time in the size of the diagrams (number of nodes). For example, given BDDs f and g, with k and m nodes, respectively, a BDD representing the function $f \wedge g$ can be constructed and the number of models of f (i.e., $\vec{x}$ s.t. $f(x) = 1$) can be computed in $\mathcal{O}(km)$ and $\mathcal{O}(k)$ time, respectively. *Layer* i in an ordered BDD, is the set of nodes with variable label x_i (possibly empty).

A DSDBDD [45, 46, 92, 218, 264, 265, 282, 283] augments a BDD by considering the disjoint support decompositions of its nodes. A *disjoint support decomposition* of a function f decomposes it into its *kernel* k and its *factors* $j_1, \ldots, j_m$, as follows:

$$f(x_1, \ldots, x_n) = k(\ell_1, \ldots, \ell_m) \text{ with } \ell_i \triangleq j_i(x_{i,1}, \ldots, x_{i,n_i}) \tag{6.1}$$

Here factor j_i takes n_i variables as input; the variables ℓ_i are "dummy variables". The factors have no variables in common, so the numbers n_i sum to n. A decomposition is non-trivial if there are at least two factors, and one factor reads at least two variables.

Ashenhurst [22] was the first to develop a theory of disjoint support decompositions of Boolean functions and to give an algorithm which finds the decomposition given f's truth table, requiring time exponential in the number of variables. He showed that by repeatedly decomposing the functions k and j_i, the fixpoint reached is uniquely determined by f, up to complementation of the factors, and up to permutation of the order in which they appear as inputs to the kernel. This tree of functions is sometimes called the *Ashenhurst-Curtis decomposition* of f.

In [45], Bertacco and Damiani describe and implement an efficient algorithm to build a DSDBDD as follows. If a Shannon node in a BDD represents a function which allows a non-trivial decomposition, this node and its children are replaced by a dedicated *decomposition node* pointing to BDDs representing its kernel and its factors. These factors may themselves be decompositions, allowing 'nesting' of decompositions. This process is repeated until no Shannon node is eligible. Thus, a hybrid diagram is obtained, in which some nodes indicate decompositions (see Definition 6.2). Because of Ashenhurst's unique decomposition theorem [22], DSDBDDs are canonical like BDDs. The goal is that the new diagram is smaller than BDD, since this method may remove more nodes than it adds, but analytically little was known about this up to now.

Definition 6.2 (Disjoint Support Decomposition Diagram (DSDBDD))**.** A DSDBDD is a BDD whose internal nodes are either Shannon nodes or *decomposition nodes*. A decomposition node v has an outgoing edge to an internal node v_{ker} called its *kernel* and outgoing edges to its *factors* $v_1, \ldots, v_m$. It represents the function $[\![v]\!] = [\![v_{\text{ker}}]\!]([\![v_1]\!], \ldots, [\![v_m]\!])$, like in Equation 6.1. The diagram satisfies the following three rules:

1. If v is a factor of a decomposition node, then v satisfies $[\![v]\!](0, \ldots, 0) = 1$
2. Two factors $[\![v_i]\!]$ and $[\![v_j]\!]$ of a decomposition node have disjoint support, i.e., $vars([\![v_i]\!]) \cap vars([\![v_j]\!]) = \emptyset$, for $i \neq j$, where $vars(f)$ denotes the set of variables on which f depends.
3. The factors $v_1, \ldots, v_m$ of a decomposition node satisfy $\min vars([\![v_i]\!]) < \min vars([\![v_j]\!])$ for $i < j$, where min is relative to the diagram's variable order. ◇

Figure 6.1(c) shows a DSDBDD for the same function f as 6.1(a) and 6.1(b). Since this function can be expressed as a formula referencing each variable once, the DSDBDD

can easily decompose it, obtaining the kernel $k = \text{AND}$ on three variables. The factors are $x_i \Leftrightarrow x_{i+1}$ for $i = 1, 3, 5$. We remark that this succinct decomposition is available to a DSDBDD regardless of the variable order, whereas the BDD may have exponential size unless the right variable order is found. Figure 6.1(d) shows that the root of a DSDBDD is not necessarily a decomposition node.

Let us briefly motivate the three rules in Definition 6.2, which are similar to those formulated by Bertacco and Damiani [45]. The purpose of the rules is to keep the query and manipulation operations tractable, i.e., to prevent the diagram from becoming more expressive than intended. Notably, without rule 2, we no longer have efficient algorithms for querying and manipulating such a diagram; for example, model counting would be NP-hard, because, a 3-CNF formula may now be represented as a decomposition with kernel AND, and whose factors are disjunctions on three variables. Rule 1 compensates the fact that, according to Ashenhurst's Theorem, a decomposition is unique up to complementation of the factors. For example, if a function f has a decomposition $f = k(\ell_1, \ldots, \ell_m)$ with $\ell_i = j_i(x_{i,1}, \ldots, x_{i,n_i})$ as in Equation 6.1, then another decomposition is $f = k'(\neg\ell_1, \ldots, \neg\ell_m)$, where k' takes the values $k'(\ell_1, \ldots, \ell_m) \triangleq k(\neg\ell_1, \ldots, \neg\ell_m)$. More generally, for each factor, the complementation may be chosen independently, leading to exponentially many possible decompositions. Rule 1 uniquely determines the choice of complementation by enforcing that, for each factor, $j_i(0, \ldots, 0) = 1$. Similarly, rule 3 compensates for the fact that a decomposition is unique up to permutation of the kernel's input variables. For example, we may write the function f above as $f = k''(j_m, \ldots, j_1)$ where $k''(\ell_1, \ldots, \ell_m) \triangleq k(\ell_m, \ldots, \ell_1)$.

Technically, the kernel of a decomposition node takes as input variables that are not inputs to f. The question which variables of the kernel to identify with which variables of the DD can be an important design decision for DSDBDD package implementations, and for obtaining canonicity. The diagram can be made canonical by imposing additional rules. Since such a canonical diagram is included in the above definition, a separation between BDDs and Definition 6.2 implies a separation with the canonical version. Therefore, we omit the strengthening of Definition 6.2 to obtain canonicity for the purposes of this work.

DSDBDDs supports the same queries and manipulation operations as BDDs (i.e., conjunction, disjunction, negation, model counting, etc.). These algorithms greedily minimize the DD by checking, whenever a new node is constructed, whether the node allows a decomposition, and then building this decomposition before proceeding. The

worst-case running times of the algorithms are polynomial in the size of the BDDs (but not necessarily in the size of the DSDBDDs). In the best case, the running time is much better; in that case, the operands of, e.g., CONJOIN, are two decompositions whose kernels read exactly the same factors. In this case the operation can take advantage of the fact that, if $j_1, \ldots, j_m$ are functions such that f_1, f_2 decompose as $f = k_1(j_1, \ldots, j_m)$ and $f_2 = k_2(j_1, \ldots, j_m)$, then

$$f_1 \wedge f_2 = (k_1 \wedge k_2)(j_1, \ldots, j_m) \tag{6.2}$$

This allows the CONJOIN algorithm to work only on k_1 and k_2, whose diagrams may be exponentially smaller than the BDDs of f and g. In the worst case, however, the decompositions share no factors, so that CONJOIN must "unfold" these decomposition nodes into BDDs and the operation is done on the BDDs; hence, the running time is polynomial in the size of the BDDs. Bertacco and Plaza implemented these operations in the publicly available software package STACCATO [264,265]. They find that their package is competitive with CUDD both in terms of time and memory, on the task of compiling a Boolean circuit into a DD.

Similar ideas appear in AND/OR multi-valued DDs (AOMDDs) [215], which are canonial, and in BDS-Maj diagrams [11]. In BDS-Maj, the kernel is always chosen to be the Majority function on three inputs, and the factors may share variables, unlike in a DSDBDD.

6.3 Succinctness separation between DSDBDD and BDD

Theorem 6.1 shows an example of a *separating function* g (Equation 6.4) which has a small DSDBDD but exponential-sized BDD, *for every variable order*. It is based on three multiplexed copies of the order-parameterized function f, with variable orders π_0, π_1, π_2. By abuse of notation, we use z both as a bit-string, and as the integer $z \in \{0, 1, 2\}$ which the bit-string represents in base 2. The function f is well known to yield exponential BDDs for non-interleaved variable orders, as our generalized Lemma

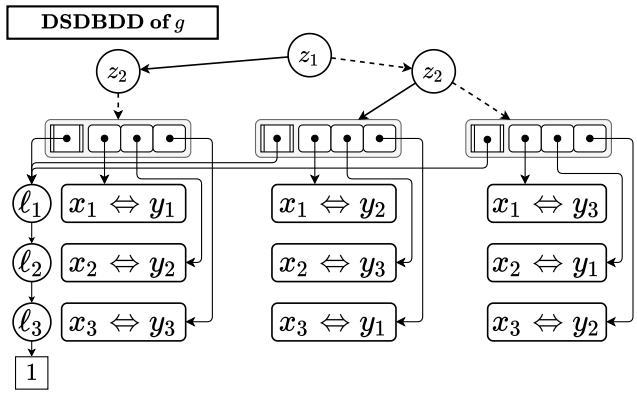

Figure 6.2: The DSDBDD of the function g, in Equation 6.4 when $n = 3$. The permutations used are $\pi_0 = (1)(2)(3), \pi_1 = (1,2,3), \pi_2 = (1,3,2)$. The rectangle containing $x_i \Leftrightarrow x_j$ represents the BDD of the function $x_i \Leftrightarrow x_j$, as shown in Figure 2.3(e).

6.1 shows. We state it without proof.

$$f[\pi](x_1, \ldots, x_n, y_1, \ldots, y_n) \triangleq (x_1 \Leftrightarrow y_{\pi(1)}) \wedge \cdots \wedge (x_n \Leftrightarrow y_{\pi(n)}) \tag{6.3}$$

$$g(z, x_1, \ldots, x_n, y_1, \ldots, y_n) \triangleq f[\pi_z](x_1, \ldots, x_n, y_1, \ldots, y_n) \text{ for } z \in \{0,1,2\}. \tag{6.4}$$

Lemma 6.1. Let $\pi \in S_n$ and σ be an order over $\{x_1, \ldots, x_n, y_1, \ldots, y_n\}$ (the variables $f[\pi]$). For $1 \leq i \leq n$, say that x_i and $y_{\pi(i)}$ are *partners*. Let L be the first n variables according to σ. If k elements in L have their partner outside of L, then a BDD of $f[\pi]$ with variable order σ has at least 2^k nodes on layer n.

By choosing distinct permutations π, π', the functions $f[\pi], f[\pi']$ will disagree on which variables are partners. Theorem 6.1 shows that there exist many irreconcilable choices for permutations $\pi_0 - \pi_2$ in g, because the corresponding "partner graph", connecting two partner variables according to either permutation, is an *expander*, i.e., has high connectivity.

Theorem 6.1. Let π_0, π_1, π_2 be permutations chosen uniformly and independently at random from S_n. Then it holds that, with high probability, for every variable order σ over $\{x_1, \ldots, x_n, y_1, \ldots, y_n\}$, at least one of the BDDs for $f[\pi_0], f[\pi_1], f[\pi_2]$ has size $2^{\Omega(n)}$ and hence the BDD for g is also large.

Proof. Let G be the undirected bipartite graph with nodes $V = \{x_1, \ldots, x_n, y_1, \ldots, y_n\}$

and edges $E = E_0 \cup E_1 \cup E_2$ with $E_j = \{(x_i, y_{\pi_j(i)}) \mid 1 \leq i \leq n\}$. Then G is an expander with high probability by Theorem 4.16 in [164]. That is, there is a constant $\varepsilon > 0$ (independent of n) such that, with high probability, for all sets of vertices $L \subset V$, if $|L| \leq n$, then

$$\frac{|N(L) \setminus L|}{|L|} \geq \varepsilon \qquad \text{where } N(L) = \{w \mid \exists v \in L : (v, w) \in E\} \tag{6.5}$$

Let σ be a variable order of V (the variables of the functions $f[\pi_j]$), and let L be the first n variables according σ. Then there are at least $\varepsilon \cdot n$ vertices in $\overline{L}$ connected to L. Since each vertex is connected to at most 3 edges, it holds that one of the edge sets E_j is responsible for at least $\varepsilon \cdot n/3$ edges crossing over from L to $\overline{L}$. Let $K = E_j \cap (L \times \overline{L})$ be a set of pairs $(x_i, y_{\pi_j(i)})$ such that x_i is in L, but its partner $y_{\pi_j(i)}$ is $\overline{L}$. It follows from Lemma 6.1 that the corresponding function $f[\pi_j]$ has a BDD of size at least $2^{|K|} = 2^{\Omega(n)}$. Since $g_{|z:=j} = f[\pi_j]$, and since a BDD is at least as large as the BDDs of its subfunctions, g also has at least $2^{\Omega(n)}$ nodes. This holds w.h.p. over the choice of permutations. □

The DSDBDD of g is shown in Figure 6.2, for $n = 3$. For larger n, the DSDBDD simply has more "rows", i.e., there are still three decomposition nodes, and they have n factors. The DSDBDD of g therefore has only $\mathcal{O}(n)$ nodes for larger n.

An immediate corollary is that the same relation holds between DSDBDDs versus ZDDs [229], Tagged BDDs [328] and CBDDs [68], since these decision diagrams are all at most a factor n smaller than BDDs on any function.

6.4 Conclusion and future work

We have analyzed the Disjoint Support Decomposition Binary Decision Diagram and found that it strictly dominates BDD and ZDD in terms of memory, up to polynomial overhead. That is, DSDBDDs can be exponentially smaller than BDDs. It remains an open question how DSDBDDs relates to other very expressive DDs; notably, it would be good to know its relation to SDDs, FBDDs, non-deterministic BDDs ($\vee$-BDD [54,55]) and d-DNNF. In addition, it would be interesting to map the complexity of DSDBDDs of the different operations considered by Darwiche & Marquis [97].

To the best of our knowledge, DSDBDDs have not been deployed on large, real-world

problems as encountered, e.g., in model checking and synthesis. Given that we showed that DSDBDDs can be exponentially more succinct, and they retain canonicity of BDDs, it could be worthwhile to test the scalability of DSDBDD in practice. In a similar vein, the integration of disjoint support decompositions into other decision diagrams could be considered. Minato [230] shows how to find the DSDs of the nodes in a ZDD; a next step would be to integrate this into the Boolean operations of ZDDs, as was done in [264,265], so that the diagram remains small during compilation. Other promising candidates for integration with DSDs are FDDs and SDDs; we are not aware of any work in this direction.