Quantum Algorithm for Finding the Optimal Variable Ordering for Binary Decision Diagrams*

Seiichiro Tani[1,a)]

Abstract: An ordered binary decision diagram (OBDD) is a directed acyclic graph that represents a Boolean function. OBDDs are also known as special cases of oblivious read-once branching programs in the field of complexity theory. Since OBDDs have many nice properties as data structures, they have been extensively studied for decades in both theoretical and practical fields, such as VLSI design, formal verification, machine learning, and combinatorial problems. Arguably, the most crucial problem in using OBDDs is that they may vary exponentially in size depending on their variable ordering (i.e., the order in which the variables are to read) when they represent the same function. Indeed, it is NP hard to find an optimal variable ordering that minimizes an OBDD for a given function. Hence, numerous studies have sought heuristics to find an optimal variable ordering. From practical as well as theoretical points of view, it is also important to seek algorithms that output optimal solutions with lower (exponential) time complexity than trivial brute-force algorithms do. Friedman and Supowit provided a clever deterministic algorithm with time/space complexity $O^*(3^n)$, where n is the number of variables of the function, which is much better than the trivial brute-force bound $O^*(n!2^n)$. This paper shows that a further speedup is possible with quantum computers by demonstrating the existence of a quantum algorithm that produces a minimum OBDD together with the corresponding variable ordering in $O^*(2.77286^n)$ time and space with an exponentially small error. Moreover, this algorithm can be adapted to constructing other minimum decision diagrams such as zero-suppressed BDDs, which provide compact representations of sparse sets and are often used in the field of discrete optimization and enumeration.
Full Version: arXiv:1909.12658

Keywords: quantum algorithm, ordered binary decision diagram.

1. Background

1.1 Ordered binary decision diagrams.

The ordered binary decision diagram (OBDD) is one of the data structures that have been most often used for decades to represent Boolean functions in practical situations, such as VLSI design, formal verification, optimization of combinatorial problems, and machine learning, and it has been extensively studied from both theoretical and practical standpoints (see standard textbooks and surveys, e.g., Refs. [2], [3], [4], [5], [6], [7]). Moreover, many variants of OBDDs have been invented to more efficiently represent data with properties observed frequently in specific applications (e.g., Refs. [8], [9], [10], [11], [12]). More technically speaking, OBDDs are directed acyclic graphs that represent Boolean functions and also known as special cases of oblivious read-once branching programs in the field of complexity theory. The reason for OBDDs' popularity lies in their nice properties — they can be uniquely determined up to isomorphism for each function once *variable ordering* (i.e., the order in which to read the variables) is fixed and, thanks to this property, the equivalence of functions can be checked by just testing the isomorphism between the OBDDs representing the functions (this can be done in linear time since each OBDD is a directed acyclic graphs with a single source node and labeled edges). In addition, binary operations such as AND and OR between two functions can be performed efficiently over the OBDDs representing those functions [13]. Since these properties are essential in many applications, OBDDs have gathered much attention from various research fields. To enjoy these nice properties, however, we actually need to address a crucial problem, which is that OBDDs may vary exponentially in size depending on their variable ordering. For instance, a Boolean function $f(x_1, \ldots, x_{2n}) = x_1x_2 + x_3x_4 + \cdots + x_{2n-1}x_{2n}$ has a $(2n + 2)$-sized OBDD for the ordering $(x_1, \ldots, x_{2n})$ and a 2^{n+1}-sized OBDD for the ordering $(x_1, x_3, \ldots, x_{2n-1}, x_2, x_4, \ldots, x_{2n})$ [3], Sec. 8.1 (see Fig. 1 for the case where $n = 3$). This is not a rare phenomenon; it could happen in many concrete functions that one encounters. Thus, since the early stages of OBDD research, one of the most central problems has been how to find an optimal variable ordering, i.e., one that minimizes OBDDs. Since there are $n!$ permutations over n variables, the brute-force search requires at least $n! = 2^{\Omega(n \log n)}$ time to find an optimal variable ordering. Indeed, finding an optimal variable ordering for a given function is an NP hard problem (see Sec. 6 for the studies on the hardness).

To tackle this high complexity, many heuristics have been proposed to find an optimal variable ordering or a relatively good one. These heuristics work well for Boolean functions appearing in specific applications since they are based on very insightful ob-

[1] NTT Communication Science Laboratories, NTT Corporation.
3-1 Morinosato-Wakamiya, Atsugi, Kanagawa 243-0198, Japan.
[a)] seiichiro.tani.cs@hco.ntt.co.jp
[*1] A conference version of this paper is Ref. [1].

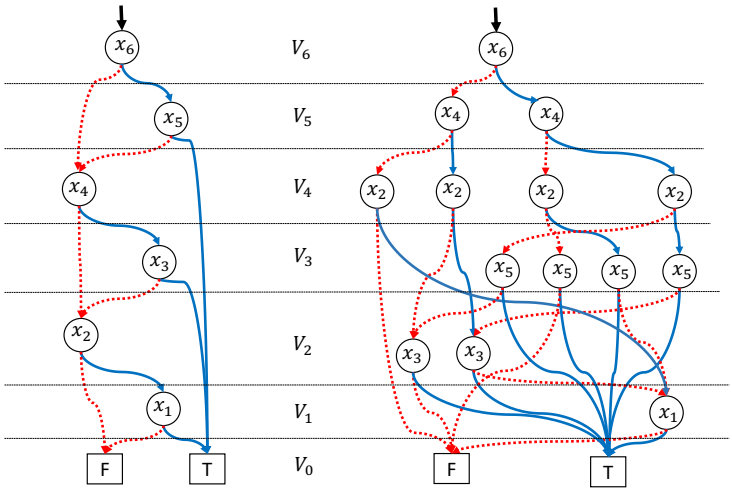

Fig. 1 The OBDDs represent the function $f(x_1, x_2, x_3, x_4, x_5, x_6) = x_1x_2 + x_3x_4 + x_5x_6$ under two variable orderings: $(x_1, x_2, x_3, x_4, x_5, x_6)$ (left) and $(x_1, x_3, x_5, x_2, x_4, x_6)$ (right), where the solid and dotted arcs express 1-edges and 0-edges, respectively, and the terminal nodes for true and false are labeled with T and F, respectively. In general, the function $f(x_1, \ldots, x_{2n}) = x_1x_2 + x_3x_4 + \cdots + x_{2n-1}x_{2n}$ has the a $(2n + 2)$-sized OBDD for the ordering $(x_1, \ldots, x_{2n})$ and a 2^{n+1}-sized OBDD for the ordering $(x_1, x_3, \ldots, x_{2n-1}, x_2, x_4, \ldots, x_{2n})$.

servations, but they do not guarantee a worst-case time complexity lower than that achievable with the brute-force search. The only algorithm with a much lower worst-case time complexity bound, $O^*(3^n)$ time ($O^*(\cdot)$ hides a polynomial factor), than the brute-force bound $O^*(n!2^n)$ for *all* Boolean functions with n variables was provided by Friedman and Supowit [14], and that was over thirty years ago!

In practice, it is often too costly to construct a minimum OBDD and the optimal variable ordering may change as the function changes during a procedure, say, by imposing additional constraints. Nevertheless, theoretically sound methods for finding an optimal variable ordering are worth studying for several reasons, such as to judge the optimization quality of heuristics and to be able to apply such methods at least to parts of the OBDDs within a heuristics procedure [3], Sec. 9.22.

1.2 Quantum Speedups of Dynamic Programming

Grover's quantum search algorithm [15] and its variants achieve quadratic speedups over any classical algorithm for *unstructured search*, a very fundamental problem (effectively, exhaustive search is the only classical strategy). Thus, one of the merits of the quantum search is its wide applicability. However, it does not immediately mean quantum speedups for all problems to which the quantum search is applicable, since there may exist better classical algorithms than simple exhaustive search. Indeed, quantum search for an optimal variable ordering of the OBDD from among $n!$ candidates takes approximately $\sqrt{n!} \approx 2^{\frac{1}{2}n\log n}$ time, while the best classical algorithm takes only $O^*(3^n) = O^*(2^{(\log_2 3)n})$. These classical algorithms often employ powerful algorithmic techniques such as dynamic programming, divide-and-conquer, and branch-and-bound. One typical strategy to gain quantum speedups would be to find parts of exhaustive search (often implicitly) performed within such classical algorithms and apply the quantum search to those parts. For instance, Dürr et al. [16] provided quantum algorithms for some graph problems, among which the quantum algorithm for the single-source shortest path problem achieves a quantum speedup by applying a variant of Grover's search algorithm to select the cheapest border edge in Dijkstra's algorithm. However, applying the quantum search in this way does not work when the number of states in a dynamic programming algorithm is much larger than the number of predecessors of each state. For instance, the Traveling Salesman Problem (TSP) can be solved in $O^*(2^n)$ time by a classical dynamic programming algorithm, but locally applying the quantum search can attain at most a polynomial-factor improvement. Recently, Ambainis et al. [17] has introduced break-through techniques to speed up dynamic programming approaches. They provide quantum algorithms that solve a variety of vertex ordering problems on graphs in $O^*(1.817^n)$ time, graph bandwidth in $O^*(2.946^n)$ time, and TSP and minimum set cover in $O^*(1, 728^n)$ time, where n is the number of vertices in the graphs.

2. Our Results

In this paper, we show that quantum speedup is possible for the problem of finding an optimal variable ordering of the OBDD for a given function. This is the first quantum speedup for the OBDD-related problems. Our algorithms assume the quantum random access memory (QRAM) model [18], which is commonly used in the literature concerning quantum algorithms. In the model, one can read contents from or write them into quantum memory in a superposition. We provide our main result in the following theorem.

Theorem 1 There exists a quantum algorithm that, for a function $f\colon \{0,1\}^n \to \{0,1\}$ given as its truth table, produces a minimum OBDD representing f together with the corresponding variable ordering in $O^*(\gamma^n)$ time and space with an exponentially small error probability with respect to n, where the constant γ is at most 2.77286. Moreover, the OBDD produced by our algorithm

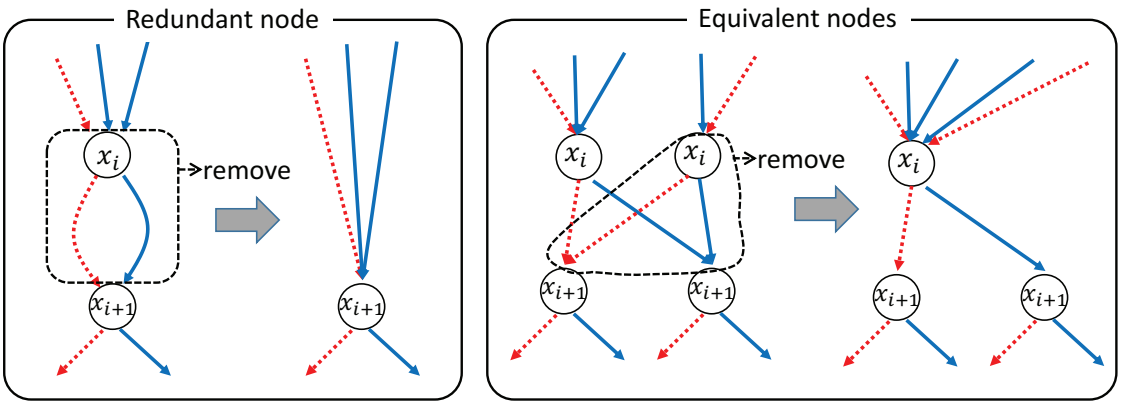

Fig. 2 Examples of a redundant node (left) and a pair of equivalent nodes (right), and their respective removal rules, where the solid and dotted arcs express 1-edges and 0-edges, respectively. In the example of a redundant node, u_0 and u_1 denote u_0 and u_1, respectively.

is always a valid one for f, although it is not minimum with an exponentially small probability.

This improves upon the classical best bound $O^*(3^n)$ [14] on time/space complexity. The classical algorithm achieving this bound is a deterministic one. However, there are no randomized algorithms that compute an optimal variable ordering in asymptotically less time complexity as far as we know.

It may seem somewhat restricted to assume that the function f is given as its truth table, since there are other common representations of Boolean functions such as DNFs, CNFs, Boolean circuits and OBDDs. However, this is not the case. Our algorithm actually works without increasing the order of complexity[*1] in more general settings where the input function f is given as any representation such that the value of f on any specified assignment can be computed over the representation in polynomial time in n, such as polynomial-size DNFs/CNFs/circuits and OBDDs of any size[*2] This is because, in such cases, the truth table of f can be prepared in $O^*(2^n)$ time/space, which is negligible compared with the total time/space complexity, and the minimum OBDD is computable from that truth table with our algorithm. We restate Theorem 1 in a more general form as follows.

Corollary 2 Let $R(f)$ be any representation of a Boolean function f with n variables such that the value of $f(x)$ on any given assignment $x \in \{0,1\}^n$ can be computed on $R(f)$ in polynomial time with respect to n. Then, there exists a quantum algorithm that, for a function $f\colon \{0,1\}^n \to \{0,1\}$ given as $R(f)$, produces a minimum OBDD representing f together with the corresponding variable ordering in $O^*(\gamma^n)$ time and space with an exponentially small error probability with respect to n, where the constant γ is at most 2.77286. Possible representations as $R(f)$ are polynomial-size DNFs/CNFs/circuits and OBDDs of any size for function f.

There are many variants of OBDDs, among which the zero-suppressed BDDs (ZDDs or ZBDDs) introduced by Minato [8] have been shown to be very powerful in dealing with combinatorial problems (see Knuth's famous book [6] for how to apply ZDDs to such problems). With slight modifications, our algorithm can construct a minimum ZDD with the same time/space complexity. We believe that similar speedups are possible for many other variants of OBDDs (adapting our algorithm to multi-terminal BDDs (MTBDDs) [10], [11] is almost trivial).

3. Ordered Binary Decision Diagrams

We provide a quick review of OBDDs. For more details, consult standard textbooks (e.g., Refs. [3], [5]).

For any Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ over variables $x_1, \ldots, x_n$ and any permutation $\pi \in \mathcal{S}_n$ (called a *variable ordering*), an OBDD $\mathcal{B}(f,\pi)$ is a single-rooted directed acyclic graph $G(V,E)$ that is unique up to isomorphism, defined as follows (examples are shown in Fig. 1).

(1) The node set V is the union of two disjoint sets N and T of *non-terminal* nodes with out-degree two and *terminal* nodes with out-degree zero, respectively, where T contains exactly two nodes: $T = \{\mathsf{f}, \mathsf{t}\}$. The set N contains a unique source node r, called the *root*.

(2) $\mathcal{B}(f,\pi)$ is a leveled graph with $n+1$ levels. Namely, the node set can be partitioned into n subsets: $V := V_0 \sqcup V_1 \sqcup \cdots \sqcup V_n$, where $V_n = \{\mathsf{r}\}$ and $V_0 = T = \{\mathsf{t}, \mathsf{f}\}$, such that each directed edge $(u,v) \in E$ is in $V_i \times V_j$ for a pair $(i,j) \in [n] \times (\{0\} \sqcup [n-1])$ with $i > j$. For each $i \in [n]$, subset V_i (called the *level* i) is associated with the variable $x_{\pi[i]}$, or alternatively, each node in V_i is labeled with $x_{\pi[i]}$.[*3] For convenience, we define a map $\mathsf{var}\colon N \to [n]$ such that if $v \in V_i$ then $\mathsf{var} = \pi[i]$.

(3) The two edges emanating from every non-terminal node v are called the 0-*edge* and the 1-*edge*, which are labeled with 0 and 1, respectively. For every $u \in N$, let u_0 and u_1 be the destinations of the 0-edge and 1-edge of u, respectively.

(4) Let $\mathcal{F}(f)$ be the set of all subfunctions of f. Define a bijective map $F\colon V \to \mathcal{F}(f)$ as follows: (a) $F(\mathsf{r}) = f$ for $\mathsf{r} \in V_n$; (b) $F(\mathsf{t}) = \mathsf{true}$ and $F(\mathsf{f}) = \mathsf{false}$ for $\mathsf{t}, \mathsf{f} \in V_0$; (c) For every $u \in N$ and $b \in \{0,1\}$, $F(u_b)$ is the subfunction obtained from $F(u)$ by substituting $x_{\mathsf{var}(u)}$ with b, i.e., $F(u_b) = F(u)|_{x_{\mathsf{var}(u)}=b}$.

[*1] Here, we consider the complexity with respect to *the number of variables*. The complexity with respect to *input size* may vary drastically.

[*2] Regardless of the size of input OBDD, the value of f on any specified assignment can be computed in $O(n)$ time by traversing the path corresponding to the assignment from the root.

[*3] In the standard definition, V_i is associated with the variable $x_{\pi[n-i]}$. Our definition follows the one given in [14] to avoid complicated subscripts of variables.

(5) $\mathcal{B}(f,\pi)$ must be minimal in the sense that the following reduction rules cannot be applied. In other words, $\mathcal{B}(f,\pi)$ is obtained by maximally applying the following rules (Fig. 2):
  (a) if there exists a *redundant* node $u \in N$, then remove u and its outgoing edges, and redirect all the incoming edges of u to u_0, where a node u is redundant if u_0 is the same node as u_1.
  (b) if there exist *equivalent nodes* $\{u,v\} \subset N$, then remove v (i.e., any one of them) and its outgoing edges, and redirect all incoming edges of v to u, where u and v are equivalent if (1) $\mathsf{var}(u)$ is equal to $\mathsf{var}(v)$, and (2) u_0 and u_1 are the same nodes as v_0 and v_1, respectively.

Example 1 For ease of understanding the above notations, let us consider the OBDD on the right side in Fig. 1. The root r is the uppermost node labeled with x_6. The variable ordering π is $(1,3,5,2,4,6)$. Every node in V_ℓ ($\ell = 1,\dots,6$) is represented by a circle labeled with $x_{\pi[\ell]}$. For instance, V_3 consists of all the four nodes labeled with x_5. For each node $v \in V_3$, it holds that $\mathsf{var}(v) = 5$. Let u be the left node labeled with x_3. Since the path from r to u consists of three edges labeled with 0, 1, and 0 in this order from the root side, $F(u)$ is represented as $F(\mathsf{r})|_{x_6=0,x_4=1,x_2=0} = f|_{x_6=0,x_4=1,x_2=0} = x_3$.

4. Technical Outline

The first step to take is to somehow adapt the dynamic programming approach of the classical algorithm [14] (called, FS) to the framework provided by Ambainis et al. [17]. Consider a Boolean function f over n variables: $x_1,\dots,x_n$. Intuitively, FS determines the variable ordering of the minimum OBDD for f by performing dynamic programming from the variable to be read last toward that to be read first. More concretely, let $(x_{\pi[1]},\dots,x_{\pi[n]})$ be the variable ordering from the one read last ($x_{\pi[1]}$) to the one read first ($x_{\pi[n]}$), where $\pi = (\pi[1],\dots,\pi[n])$ is a permutation over $[n] := \{1,\dots,n\}$. For $k = 1,\dots,n$ in this order, and for every subset $K \subseteq [n]$ of cardinality k, the algorithm FS computes a lower bound on OBDD size when $\{\pi[1],\dots,\pi[k]\} = K$ from the lower bounds on OBDD size when $\{\pi[1],\dots,\pi[k-1]\} = K \setminus \{h\}$ for all $h \in K$. Thus, by thinking of each node $z \in \{0,1\}^n$ of weight k in a Boolean hypercube as the characteristic vector of K, the algorithm FS can be seen as solving a kind of shortest path problem on a Boolean hypercube. Hence, Ambainis et al.'s framework seems applicable. Their results depend on the property that a large problem can be divided into the same kind of subproblems or, in other words, *symmetric* subproblems in the sense that they can be solved with the same algorithm. This property naturally holds in many graph problems. In our case, firstly, it is unclear whether the problem can be divided into subproblems. Secondly, subproblems would be to optimize the ordering of variable starting from the middle variable or even from the opposite end, i.e., from the variable to be read *first*, toward the one to be read *last*. Such subproblems cannot be solved with the algorithm FS, and, in particular, optimizing in the latter case essentially requires the equivalence check of subfunctions of f, which is very costly.

Our technical contribution is to find, by carefully observing the unique properties of OBDDs, that it is actually possible to even recursively divide the original problem into *asymmetric* subproblems, to generalize the algorithm FS so that it can solve the subproblems, and to use the quantum minimum finding algorithm in order to efficiently select the subproblems that essentially contribute to the optimal variable ordering.

More concretely, we show that, for any $k \in [n]$, it is possible to divide the problem into two collections of subproblems as follows: for all $K \subseteq [n]$ of cardinality k,

- problems of finding the ordering $(\pi[1],\dots,\pi[k])$ that minimizes the size of the bottom k-layers of the corresponding OBDD, assuming that the set $\{\pi[1],\dots,\pi[k]\}$ equals K,
- problems of finding the ordering $(\pi[k+1],\dots,\pi[n])$ that minimizes the size of the upper $(n-k)$-layers of the corresponding OBDD, assuming that the set $\{\pi[k+1],\dots,\pi[n]\}$ equals $[n] \setminus K$.

Then, taking the minimum of the OBDD size over all K and k with the quantum minimum finding provides a minimum OBDD and the corresponding variable ordering. To obtain a better bound, a straightforward strategy is to consider m division points ($0 < k_1 < \cdots < k_m < n$) and optimize each of the $(m+1)$ suborderings $(\pi[1],\dots,\pi[k_1]), (\pi[k_1+1],\dots,\pi[k_2]),\dots,(\pi[k_m+1],\dots,\pi[n])$. However, this makes subproblems even more asymmetric. To deal with this asymmetry, we generalize the algorithm FS so that it can cover all the subproblems. Then, by applying it to each subproblem, we optimize the suborderings with the quantum minimum finding so that the OBDD size is minimized. To improve the complexity bound further, a simple idea would be to replace the generalized FS with the quantum algorithm we have just obtained. However, the latter algorithm works only for the original problem. Thus, we generalize the quantum algorithm so that it can be applied to the asymmetric subproblems. By repeating this composition and generalization, we obtain the final algorithm.

5. Conclusion

We have provided a quantum algorithm that solves the optimal variable ordering problem, one of the central problems concerning OBDDs, in $O^*(\gamma^n)$ time and space in the quantum random access memory (QRAM) model, where constant γ is at most 2.77286 and n is the number of variables on which the input Boolean function depends. This implies an exponential speedup over the best known classical algorithm, which runs in $O^*(3^n)$ time and space.

There are several questions that we have left open. First, we do not believe that our complexity bound is (nearly) tight. Thus, the first question is whether it is possible to improve the time and/or space complexity. Our algorithm has an exponentially small error probability, with which its output, i.e., a variable ordering π and the OBDD according to π, is not optimal. However, the output OBDD is still a valid OBDD for input function f, that is, neither an OBDD not representing f nor a bit string not representing any OBDD. It would be interesting to consider the case of allowing the output that is not a valid OBDD with a small probability.

Second, a trivial classical lower bound is $\Omega(2^n)$, since the input is the truth table of a Boolean function over n variables. This also holds even in the quantum setting, since the quantum query

complexity of identifying all N bits hidden in an input oracle is $\Omega(N)$ in the worst case [19], [20]. Is it possible to provide a better lower bound (under some conjectures)?

Third, we have considered time-efficient algorithms. Their space complexity is equal to their time-complexity, up to polynomial factors. However, space complexity would be more critical than time complexity in some cases. Hence, another interesting direction appears to be the quest for space-efficient algorithms. Actually, the $O^*(2^n)$ space complexity can be achieved by the brute-force algorithm, which, for every permutation $\pi \in \Pi([n])$, constructs $\mathcal{B}(f, \pi)$ in $O^*(2^n)$ time by restricting the FS-algorithm to the fixed variable ordering π. However, this incurs the huge time complexity of $O^*(n!2^n)$. In general, there is a trade-off between time and space. Thus, a reasonable direction would be to make the product TS of time complexity T and space complexity S as small as possible. For instance, the product TS of the brute-force algorithm is $O^*(n!4^n)$, while the best known classical algorithm FS has $TS = O^*(3^n \cdot 3^n) = O^*(9^n)$. Our algorithm has a much smaller TS value of $O^*(2.77286^{2n}) = O^*(7.68875^n)$. Is it possible to provide an algorithm that achieves a lower TS value?

Finally, the divide-and-conquer lemma (given in the full paper) does not depend on quantum computing. It would be interesting to find classical applications of this lemma.

6. Related Work

The studies related to minimizing OBDDs are so numerous that we cannot cover all of them. We thus pick up some of purely theoretical work.

Meinel and Slobodová [21] proved that it is NP hard to construct an optimal OBDD for a Boolean function given by a logical circuit, a DNF, a CNF, or an OBDD, even if the optimal OBDD is of constant size. Tani, Hamaguchi and Yajima [22] proved that it is NP hard to improve the variable ordering (and thus, to find an optimal variable ordering) for a given *multi*-rooted OBDD, where the NP hardness is proved by a reduction from an NP complete problem, Optimal Linear Arrangement [23]. Bollig and Wegener [24] finally proved the NP hardness for a given *single*-rooted OBDD by providing a sophisticated reduction from the same problem. This is still true if the input function is restricted to monotone functions [25]. Minimizing the width of an OBDD is also NP hard [26]. As for approximation hardness, Sieling [27], [28] proved that if there exists a polynomial-time approximation scheme for computing the size of the minimum OBDD for a given OBDD, it then holds that P = NP.

It would be nice if, for every function, there exists at least one variable ordering under which the OBDD for the function is of a size bounded by a polynomial. As one may expect, this is not the case. It can be proved by a counting argument that there exists a function for which the OBDD size grows exponentially in the number of variables under *any* variable ordering [29], [30], [31]. Moreover, concrete examples of such functions are known: the multiplication function [32], a threshold function [33], and the division function [34] (for other classes of Boolean functions, see Ref. [35], [36], [37]). The OBDD size is also studied from the viewpoint of computational learning and knowledge-bases [38], [39].

In applying OBDDs to graph problems, it is possible to find variable orderings for which OBDD size is nontrivially upper-bounded in terms of certain measures characterizing graph structures [40], [41]. A similar concept was discoved for ZDDs [8] by Knuth [6]. This concept is now called the *frontier method*, and lots of work is based on it.

Acknowledgments This work is partially supported by JSPS KAKENHI Grant No. JP20H05966 and JST [Moonshot R&D – MILLENNIA Program] Grant No. JPMJMS2061.

References

[1] Tani, S.: Quantum Algorithm for Finding the Optimal Variable Ordering for Binary Decision Diagrams, *Proceedings of the 17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020)* (Albers, S., ed.), LIPIcs, Vol. 162, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 36:1–36:19 (2020).

[2] Bryant, R. E.: Symbolic Boolean Manipulation with Ordered Binary-decision Diagrams, *ACM Comput. Surv.*, Vol. 24, No. 3, pp. 293–318 (1992).

[3] Meinel, C. and Theobald, T.: *Algorithms and Data Structures in VLSI Design: OBDD - Foundations and Applications*, Springer (1998).

[4] Drechsler, R. and Becker, B.: *Binary Decision Diagrams: Theory and Implementation*, Springer (1998).

[5] Wegener, I.: *Branching Programs and Binary Decision Diagrams*, SIAM Monographs on Discrete Mathematics and Applications, SIAM (2000).

[6] Knuth, D. E.: *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams*, Addison-Wesley Professional, 1 edition (2009).

[7] Bryant, R. E.: Binary Decision Diagrams, *Handbook of Model Checking*, pp. 191–217 (2018).

[8] Minato, S.: Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems, *Proceedins of the 30th ACM/IEEE Design Automation Conference*, pp. 272–277 (1993).

[9] Bryant, R. E. and Chen, Y.: Verification of Arithmetic Circuits with Binary Moment Diagrams, *Proceedings of the 32st Conference on Design Automation, San Francisco, California, USA, Moscone Center, June 12-16, 1995.*, pp. 535–541 (1995).

[10] Bahar, R. I., Frohm, E. A., Gaona, C. M., Hachtel, G. D., Macii, E., Pardo, A. and Somenzi, F.: Algebric Decision Diagrams and Their Applications, *Formal Methods in System Design*, Vol. 10, No. 2–3, pp. 171–206 (1997).

[11] Clarke, E., Mcmillan, K. L., Zhao, X., Fujita, M. and Yang, J.: Spectral Transforms for Large Boolean Functions with Applications to Technology Mapping, *Formal Methods in System Design*, Vol. 10, No. 2–3, pp. 137–148 (1997).

[12] Minato, S.: πDD: A New Decision Diagram for Efficient Problem Solving in Permutation Space, *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT 2011)*, Lecture Notes in Computer Science, Vol. 6695, pp. 90–104 (2011).

[13] Bryant, R. E.: Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Trans. Comput.*, Vol. 35, No. 8, pp. 677–691 (1986).

[14] Friedman, S. J. and Supowit, K. J.: Finding the optimal variable ordering for binary decision diagrams, *IEEE Transactions on Computers*, Vol. 39, No. 5, pp. 710–713 (1990).

[15] Grover, L. K.: A fast quantum mechanical algorithm for database search, *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pp. 212–219 (1996).

[16] Dürr, C., Heiligman, M., Høyer, P. and Mhalla, M.: Quantum Query Complexity of Some Graph Problems, *SIAM Journal on Computing*, Vol. 35, No. 6, pp. 1310–1328 (2006).

[17] Ambainis, A., Balodis, K., Iraids, J., Kokainis, M., Prusis, K. and Vihrovs, J.: Quantum Speedups for Exponential-Time Dynamic Programming Algorithms, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pp. 1783–1793 (2019).

[18] Giovannetti, V., Lloyd, S. and Maccone, L.: Quantum Random Access Memory, *Phys. Rev. Lett.*, Vol. 100, p. 160501 (2008).

[19] Ambainis, A., Iwama, K., Nakanishi, M., Nishimura, H., Raymond, R., Tani, S. and Yamashita, S.: Average/Worst-Case Gap of Quantum Query Complexities by On-Set Size, *arXiv*, Vol. 0908.2468 (2009).

[20] Kothari, R.: An optimal quantum algorithm for the oracle identification problem, *Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, pp. 482–493 (2014).

[21] Meinel, C. and Slobodová, A.: On the complexity of constructing optimal ordered binary decision diagrams, *Proceedings of 19th Mathematical Foundations of Computer Science 1994*, Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 515–524 (1994).
[22] Tani, S., Hamaguchi, K. and Yajima, S.: The Complexity of the Optimal Variable Ordering Problems of a Shared Binary Decision Diagram, *IEICE Transactions on Information and Systems*, Vol. E79-D, No. 4, pp. 271–281 (1996). (A conference version is in *Proceedings of the Fourth International Symposium on Algorithms and Computation (ISAAC'93)*, vol. 2906, pp.389–398, Lecture Notes in Computer Science, Springer, 1993).
[23] Garey, M. R. and Johnson, D. S.: *COMPUTERS AND INTRACTABILITY — A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 2 edition (1979).
[24] Bollig, B. and Wegener, I.: Improving the variable ordering of OBDDs is NP-complete, *IEEE Transactions on Computers*, Vol. 45, No. 9, pp. 993–1002 (1996).
[25] Iwama, K., Nouzoe, M. and Yajima, S.: Optimizing OBDDs Is Still Intractable for Monotone Functions, *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS'98)*, Lecture Notes in Computer Science, Vol. 1450, Springer, pp. 625–635 (1998).
[26] Bollig, B.: On the Minimization of (Complete) Ordered Binary Decision Diagrams, *Theory of Computing Systems*, Vol. 59, No. 3, pp. 532–559 (2016).
[27] Sieling, D.: The complexity of minimizing and learning OBDDs and FBDDs, *Discrete Applied Mathematics*, Vol. 122, No. 1, pp. 263 – 282 (2002).
[28] Sieling, D.: The Nonapproximability of OBDD Minimization, *Information and Computation*, Vol. 172, No. 2, pp. 103 – 138 (2002).
[29] Lee, C. Y.: Representation of switching circuits by binary-decision programs, *The Bell System Technical Journal*, Vol. 38, No. 4, pp. 985–999 (1959).
[30] Heh-Tyan Liaw and Chen-Shang Lin: On the OBDD-representation of general Boolean functions, *IEEE Transactions on Computers*, Vol. 41, No. 6, pp. 661–664 (1992).
[31] Heap, M. A. and Mercer, M. R.: Least upper bounds on OBDD sizes, *IEEE Transactions on Computers*, Vol. 43, No. 6, pp. 764–767 (1994).
[32] Bryant, R. E.: On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication, *IEEE Transactions on Computers*, Vol. 40, No. 2, pp. 205–213 (1991).
[33] Hosaka, K., Takenaga, Y., Kaneda, T. and Yajima, S.: Size of ordered binary decision diagrams representing threshold functions, *Theoretical Computer Science*, Vol. 180, No. 1, pp. 47 – 60 (1997).
[34] Horiyama, T. and Yajima, S.: Exponential Lower Bounds on the Size of OBDDs Representing Integer Divistion, *Proceedings of the 8th International Symposium on Algorithms and Computation, (ISAAC '97), Singapore, December 17-19, 1997, Proceedings*, Lecture Notes in Computer Science, Springer, pp. 163–172 (1997).
[35] Sawada, H., Takenaga, Y. and Yajima, S.: On the Computational Power of Binary Decision Diagrams, *IEICE Trans. Info. & Syst., D*, Vol. 77, No. 6, pp. 611–618 (1994).
[36] Heap, M.: On the exact ordered binary decision diagram size of totally symmetric functions, *Journal of Electronic Testing*, Vol. 4, No. 2, pp. 191–195 (1993).
[37] Heinrich-Litan, L. and Molitor, P.: Least upper bounds for the size of OBDDs using symmetry properties, *IEEE Transactions on Computers*, Vol. 49, No. 4, pp. 360–368 (2000).
[38] Takenaga, Y. and Yajima, S.: Hardness of identifying the minimum ordered binary decision diagram, *Discrete Applied Mathematics*, Vol. 107, No. 1-3, pp. 191–201 (2000).
[39] Horiyama, T. and Ibaraki, T.: Ordered binary decision diagrams as knowledge-bases, *Artif. Intell.*, Vol. 136, No. 2, pp. 189–213 (2002).
[40] Tani, S. and Imai, H.: A Reordering Operation for an Ordered Binary Decision Diagram and an Extended Framework for Combinatorics of Graphs, *Proceedings of the Fifth International Symposium on Algorithms and Computation (ISAAC'94)*, Lecture Notes in Computer Science, Vol. 834, Springer-Verlag, pp. 575–583 (1994).
[41] Sekine, K., Imai, H. and Tani, S.: Computing the Tutte Polynomial of a Graph of Moderate Size, *Proceedings of the Sixth International Symposium on Algorithms and Computation (ISAAC '95)*, Lecture Notes in Computer Science, Vol. 1004, Springer, pp. 224–233 (1995).