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

Seiichiro Tani

NTT Communication Science Laboratories, NTT Corporation.

seiichiro.tani.cs@hco.ntt.co.jp

**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 variable 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.

# 1 Introduction

## 1.1 Background

**Ordered binary decision diagrams.** The ordered binary decision diagram (OBDD) [Lee59, Ake78] 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. [Bry92, MT98, DB98, Weg00, Knu09, Bry18]). Moreover, many variants of OBDDs have been invented to more efficiently represent data with properties observed frequently in specific areas (e.g., Refs. [Min93, BC95, BFG+97, CMZ+97, Min11]). More technically speaking, OBDDs are directed acyclic graphs that represents 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 the 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. In addition, binary operations such as AND and OR between two functions can be performed efficiently over the OBDDs representing those functions [Bry86]. 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, when they represent the same function. For instance, function $f(x_1, \ldots, x_{2n}) = x_1 x_2 + x_3 x_4 + \cdots + x_{2n-1} x_{2n}$ 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 $(x_1, x_3, \ldots, x_{2n-1}, x_2, x_4, \ldots, x_{2n})$ [MT98, Sec. 8.1] (see Fig. 1 for the case where $n = 6$). 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 of a function, 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.

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 in specific applications since they are based on very insightful observations, 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[1]] than the brute-force bound $O^*(n!2^n)$ for *all* Boolean functions with $n$ variables was provided by Friedman and Supowit [FS90], and that was almost thirty years ago!

In practical, 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 [MT98, Sec. 9.22].

**Quantum Speedups of Dynamic Programming** Grover's quantum search algorithm [Gro96] and its variants achieve quadratic speedups over any classical algorithm for the very fundamental problem of *exhaustive search*. 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 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

---

[1] $O^*(\cdot)$ hides a polynomial factor. This hidden polynomial factor can be improved [BW96, Knu09].

branch-and-bound. One typical strategy to gain quantum speedups would be to find exhaustive search (often implicitly) performed within such classical algorithms and apply the quantum search to that part. For instance, Dürr et al. [DHHM06] 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. [ABI$^+$19] 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. Recently, Shimizu and Mori [SM19] applied the idea of Ambainis et al. to graph coloring problems to achieve quantum speedups.

## 1.2 Our Results

In this paper, we show that quantum speedup is possible for the problem of finding an optimal variable ordering for a given function. Our algorithms assume the quantum random access memory (QRAM) model [GLM08], which is commonly used in the literature concerned with 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 (Informal)** *There exists a quantum algorithm that, for a function $f : \{0, 1\}^n \to \{0, 1\}$ 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 with respect to $n$, where the constant $\gamma$ is at most $2.77286$. Moreover, the OBDD produced by our algorithm is always a valid one for $f$, although it is not minimum with an exponentially small probability.*

This improves upon the classical best bound $O^*(3^n)$ [FS90] 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 less time complexity as far as we know.

It may seem somewhat restricted to assume that the function $f$ is given as the 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 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. This is because, in such cases, the truth table of $f$ can be prepared in $O^*(2^n)$ time 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 $f : \{0, 1\}^n \to \{0, 1\}$ 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 with respect to $n$, where the constant $\gamma$ 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 [Min93] have been shown to be very powerful in dealing with combinatorial problems (see Knuth's famous textbook [Knu09] for how to apply ZDDs to such problems). With
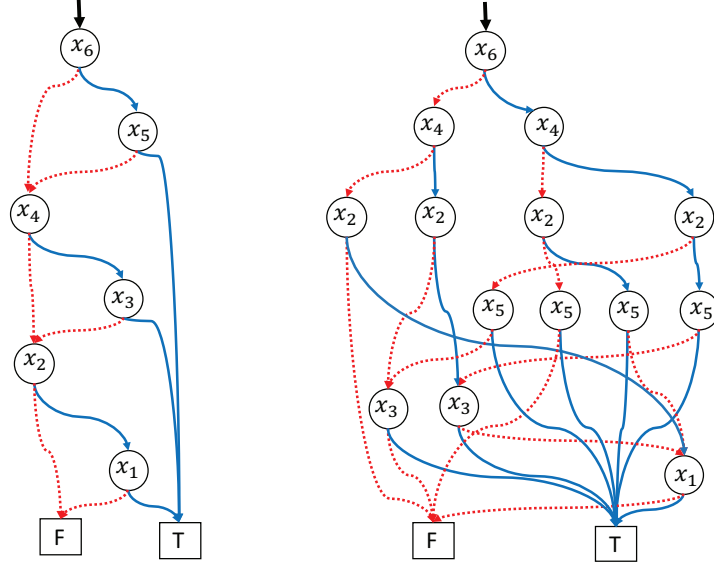
Figure 1: The OBDDs represent the function $f(x_1, x_2, x_3, x_4, x_5, x_6) = x_1x_2 + x_3x_4 + x_5x_6$ under two variable orderings: $(x_1, x_2, x_3, x_4, x_5, x_6)$ (left) and $(x_1, x_3, x_5, x_2, x_4, x_6)$ (right), where the solid and dotted lines express 1-edges and 0-edges, respectively, and the terminal nodes for false and true are labeled with F and T, 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 $(x_1, x_3, \ldots, x_{2n-1}, x_2, x_4, \ldots, x_{2n})$. A more detailed explanation of these OBDDs is provided in Example 1 in Sec. 2.2

just two-line 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 multiterminal BDDs (MTBDDs) [BFG⁺97, CMZ⁺97] is almost trivial).

## 1.3   Technical Outline

The first step to take is to somehow adapt the dynamic programming approach of the classical algorithm [FS90] (called, FS) to the framework provided by Ambainis et al. [ABI⁺19]. Consider a Boolean function $f$ over $n$ variables: $x_1, \ldots, x_n$. Let $(x_{\pi[1]}, \ldots, x_{\pi[n]})$ be the variable ordering from the one read last $(x_{\pi[1]})$ to the one read first $(x_{\pi[n]})$, where $\pi = (\pi[1], \ldots, \pi[n])$ is a permutation over $[n] := \{1, \ldots, n\}$. For $k = 1, \ldots, n$ in this order, and for every subset $K \subseteq [n]$ of cardinality $k$, the algorithm FS computes a lower bound on OBDD size when $\{\pi[1], \ldots, \pi[k]\} = K$ from the lower bounds on OBDD size when $\{\pi[1], \ldots, \pi[k-1]\} = K \setminus \{h\}$ 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 framework depends 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 cases of graph problems. In our case, however, optimizing the variable ordering from the opposite end, i.e., from the variable read *first* to the one read *last*, cannot be done with the algorithm FS and essentially requires the equivalence check of subfunctions of $f$, which is very costly. Furthermore, it is unclear whether the problem can be divided into subproblems.

In the following sections, we show that it is actually possible to divide our problem into *asymmetric* subproblems. 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 $(\pi[1], \ldots, \pi[k])$ that minimizes the size of the bottom $k$-layers of the corresponding OBDD, assuming that the set $\{\pi[1], \ldots, \pi[k]\}$ equals $K$,

3

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

Then, taking the minimum of the OBDD size over all $K$ and $k$ with quantum search 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 $(\pi[1], \ldots, \pi[k_1]), (\pi[k_1+1], \ldots, \pi[k_2]), \ldots, (\pi[k_m+1], \ldots, \pi[n])$. 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 quantum search 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.

## 1.4 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 [MS94] 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 [THY96] 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 Optimal Linear Arrangement [GJ79]. Bollig and Wegener [BW96] finally proved the NP hardness for a given *single*-rooted OBDD. This is still true if the input function is restricted to monotone functions [INY98]. Minimizing the width of an OBDD is also NP hard [Bol16]. As for approximation hardness, Sieling [Sie02a, Sie02b] 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 NP = P.

One may think that 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. However, this is not the case, which 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 [Lee59, HC92, HM94]. Examples of such functions include the multiplication function [Bry91], a threshold function [HTKY97], and the division function [HY97]. The sizes of OBDDs for several classes of Boolean functions are investigated [STY94, Hea93, HM00]. The OBDD size is also studied from the viewpoint of computational learning and knowledge-bases [TY00, HI02].

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 [TI94, SIT95]. A similar concept was discoved for ZDDs [Min93] by Knuth [Knu09]. This concept is now called the *frontier method*, and lots of work is based on it.

# 2 Preliminaries

## 2.1 Basic Terminology

Let $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}$ be the sets of natural numbers, integers, and real numbers, respectively. For each $n \in \mathbb{N}$, let $[n]$ be the set $\{1, \ldots, n\}$, and $\mathcal{S}_n$ be the permutation group over $[n]$. We may denote a singleton set $\{k\}$ by $k$ for notational simplicity if it is clear from the context; for instance, $I \setminus \{k\}$ may be denoted by $I \setminus k$, if we know $I$ is a set. For any subset $I \subseteq [n]$, let $\Pi_n(I)$ be the set of $\pi \in \mathcal{S}_n$ such that the first $|I|$ members $\{\pi[1], \ldots, \pi[|I|]\}$ constitutes $I$, i.e.,

$$\Pi_n(I) := \{\pi \in \mathcal{S}_n \colon \{\pi[1], \ldots, \pi[|I|]\} = I\} \subseteq \mathcal{S}_n.$$

For simplicity, we omit the subscript $n$ and write $\Pi(I)$. More generally, for any two disjoint subsets $I, J \subseteq [n]$, let

$$\Pi_n(\langle I, J\rangle) := \{\pi \in \mathcal{S}_n : \{\pi[1], \ldots, \pi[|I|]\} = I, \{\pi[|I|+1], \ldots, \pi[|I|+|J|]\} = J\} \subseteq \mathcal{S}_n.$$

For any disjoint subsets $I_1, \ldots, I_m \subseteq [n]$ for $m \in [n]$, $\Pi_n(\langle I_1, \ldots, I_m\rangle)$ is defined similarly. For notational consistency, we denote $\langle I\rangle$ by $I$.

We denote the union operation over *disjoint* sets by $\sqcup$ (instead of $\cup$) when we emphasize the disjointness of the sets. For example, the union of the disjoint sets $\{1, 2\}$ and $\{5, 6\}$ is denoted by $\{1, 2\} \sqcup \{5, 6\}$.

For $n$ Boolean variables $x_1, \ldots, x_n$, any set $I \subseteq [n]$, and any vector $b = (b_1, \ldots, b_{|I|}) \in \{0, 1\}^{|I|}$, $x_I$ denotes the ordered set $(x_{j_1}, \ldots, x_{j_{|I|}})$, where $\{j_i : i \in [|I|]\} = I$ and $j_1 < \cdots < j_{|I|}$, and $x_I = b$ denotes $x_{j_i} = b_i$ for each $i = [|I|]$. For any Boolean function $f : \{0, 1\}^n \to \{0, 1\}$ with variables $x_1, \ldots, x_n$, we denote by $f|_{x_I=b}$ the function obtained by restricting $f$ with $x_I = b$. If $I$ is a singleton set, say, $I = \{i\}$, we may write $x_i$ and $f|_{x_i=b}$ to mean $x_{\{i\}}$ and $f|_{x_{\{i\}}=b}$, respectively, for notational simplicity. We say that $g$ is a *subfunction* of $f$ if $g$ is equivalent to the function $f|_{x_I=b}$ for some $I \subseteq [n]$ and $b \in \{0, 1\}^{|I|}$.

For any function $g(n)$ in $n$, we use the notation $O^*(g(n))$ to hide a polynomial factor in $n$. For instance, $n^3 2^n$ and $2^n$ are both in $O^*(2^n)$. We further denote $X = O^*(Y)$ by $X \lessgtr Y$.

We use the following upper bound many times in this paper. For $n \in \mathbb{N}$ and $k \in [n] \cup \{0\}$, it holds that $\binom{n}{k} \leq 2^{n\,\mathbf{H}(k/n)}$, where $\mathbf{H}(\cdot)$ represents the binary entropy function $\mathbf{H}(\delta) := -\delta \log_2 \delta - (1-\delta) \log_2(1-\delta)$.

## 2.2 Ordered Binary Decision Diagrams

This subsection briefly introduces ordered binary decision diagrams (OBDDs). For interested readers, see standard textbooks and survey papers (e.g., Refs. [Bry92, MT98, DB98, Weg00, Bry18]).

OBDDs are a special case of read-once oblivious branching programs in complexity-theoretic terms that is, branching programs that satisfy the following conditions: each variable is read at most once on each directed path from the root to a terminal node, and the orderings of variables to be read on all such paths are consistent with a certain fixed ordering.

In the following, we formally define OBDDs in graph-theoretic terms and provide several notations. For any Boolean function $f : \{0, 1\}^n \to \{0, 1\}$ 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 directed acyclic graph $G(V, E)$ defined as follows:

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: $T = \{\mathsf{t}, \mathsf{f}\}$. The set $N$ contains a unique source node $\mathsf{r} \in N$, 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 unique 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 $x_{\pi[i]}$[2]. For convenience, we define a map $\mathsf{var} : N \to [n]$ such that if $v \in V_i$ then $\mathsf{var}(v) = \pi[i]$.

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. There exists a bijective map $F$ from $N \sqcup T$ to the set of subfunctions of $f$ such that the following holds: (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., $F(u_b) = F(u)|_{x_{\mathsf{var}(u)}=b}$.

5. $\mathcal{B}(f, \pi)$ must be minimum 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:

---

[2]In the standard definition, $V_i$ is associated with the variable $x_{\pi[n-i]}$. Our definition follows the one given in [FS90] to avoid complicated subscripts of variables.

(a) if there exists a node $u \in N$ such that $u_0 \equiv u_1$, then remove $u$ and its outgoing edges, and redirect all its incoming edges of $u$ to $u_0$;

(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 $\mathsf{var}(u) = \mathsf{var}(v)$, $u_0 \equiv v_0$, and $u_1 \equiv v_1$.

Note that this is not necessarily an actual way of constructing $\mathcal{B}(f, \pi)$, but just a way of defining $\mathcal{B}(f, \pi)$.

**Example 1** *For ease of understanding the above notations, let us consider the OBDD on the right side in Fig. 1. The root node $\mathsf{r}$ is the uppermost node labeled with $x_6$. The ordering $\pi$ is $(1, 3, 5, 2, 4, 6)$. Every node in $V_i$ $(i = 1, \ldots, 6)$ is represented by a circle labeled with $x_{\pi[i]}$. 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 $\mathsf{r}$ to $u$ consists of three edges labeled with 0, 1, and 0 in this order from the root side, the subfunction $F(u)$ is defined as $F(r)|_{x_6=0,x_4=1,x_2=0} = f|_{x_6=0,x_4=1,x_2=0} = x_3$.*

For each $j \in [n]$, $\mathsf{Cost}_j(f, \pi)$ denotes the width at the level associated with the variable $x_j$, namely, the number of nodes in the level $\pi^{-1}[j]$. For $I \subseteq [n]$, let $\pi_I$ be a permutation $\pi$ in $\Pi(I)$ that minimizes the number of nodes in level 1 to level $|I|$:

$$\pi_I := \arg\min \left\{ \sum_{j=1}^{|I|} \mathsf{Cost}_{\pi[j]}(f, \pi) : \pi \in \Pi(I) \right\}. \tag{1}$$

Note that $\sum_{j=1}^{|I|} \mathsf{Cost}_{\pi[j]}(f, \pi) = \sum_{i \in I} \mathsf{Cost}_i(f, \pi)$ for $\pi \in \Pi(I)$. More generally, for disjoint subsets $I_1, \ldots, I_m \subseteq [n]$, $\pi_{\langle I_1, \ldots, I_m \rangle}$ is a permutation in $\Pi(\langle I_1, \ldots, I_m \rangle)$ that minimizes the number of the nodes in level 1 to level $|I_1| + \cdots + |I_m|$ over all $\pi \in \Pi(\langle I_1, \ldots, I_m \rangle)$:

$$\pi_{\langle I_1, \ldots, I_m \rangle} := \arg\min \left\{ \sum_{j=1}^{|I_1|+\cdots+|I_m|} \mathsf{Cost}_{\pi[j]}(f, \pi) : \pi \in \Pi(\langle I_1, \ldots, I_m \rangle) \right\}. \tag{2}$$

Note that $\min \sum_{j=1}^{|I_1|+\cdots+|I_m|} \mathsf{Cost}_{\pi[j]}(f, \pi) = \sum_{i \in I_1 \sqcup \cdots \sqcup I_m} \mathsf{Cost}_i(f, \pi)$ for any $\pi \in \Pi(\langle I_1, \ldots, I_m \rangle)$. The following well-known lemma captures the essential property of OBDDs. It states that the number of nodes at level $i \in [n]$ is constant over all $\pi$, provided that the two sets $\{\pi[1], \ldots, \pi[i-1]\}$ and $\{\pi[i+1], \ldots, \pi[n]\}$ are fixed.

**Lemma 3 ([FS90])** *For any non-empty subset $I \subseteq [n]$ and any $i \in I$, there exists a constant $c_f$ such that, for each $\pi \in \Pi(\langle I \setminus \{i\}, \{i\} \rangle)$, $\mathsf{Cost}_{\pi[|I|]}(f, \pi) \equiv \mathsf{Cost}_i(f, \pi) = c_f$.*

For convenience, we define shorthand for the minimums of the sums in Eqs. (1) and (2). For $I' \subseteq I \subseteq [n]$, $\mathrm{MINCOST}_I[I']$ is defined as the number of nodes in the levels associated with variables indexed by elements in $I'$ under permutation $\pi_I$, namely, $\mathrm{MINCOST}_I[I'] := \sum_{i \in I'} \mathsf{Cost}_i(f, \pi_I)$. More generally, for disjoint subsets $I_1, \ldots, I_m \subseteq [n]$ and $I' \subseteq I_1 \sqcup \cdots \sqcup I_m$,

$$\mathrm{MINCOST}_{\langle I_1, \ldots, I_m \rangle}[I'] := \sum_{i \in I'} \mathsf{Cost}_i(f, \pi_{\langle I_1, \ldots, I_m \rangle}).$$

As a special case, we denote $\mathrm{MINCOST}_{\langle I_1, \ldots, I_m \rangle}[I_1 \sqcup \cdots \sqcup I_m]$ by $\mathrm{MINCOST}_{\langle I_1, \ldots, I_m \rangle}$. We define $\mathrm{MINCOST}_\emptyset$ as 0.

## 2.3 The Algorithm by Freedman and Supowit

This subsection reviews the algorithm by Freedman and Supowit [FS90]. We will generalize their idea later and heavily use the generalized form in our quantum algorithm. Hereafter, we call their algorithm FS.

### 2.3.1 Key Lemma and Data Structures

The following lemma is the basis of the dynamic programming approach used in the algorithm.

**Lemma 4** *For any non-empty subset $I \subseteq [n]$ and any Boolean function $f : \{0, 1\}^n \to \{0, 1\}$, the following holds:* $\text{MINCOST}_I = \min_{k \in I} \left( \text{MINCOST}_{I \setminus k} + \text{Cost}_k(f, \pi_{\langle I \setminus k, k \rangle}) \right) = \min_{k \in I} \left( \text{MINCOST}_{\langle I \setminus k, k \rangle} \right).$

The proof is given in Appendix A.

Before sketching algorithm FS, we provide several definitions. For any $I \subseteq [n]$, $\text{TABLE}_I$ is an array with $2^{n-|I|}$ cells each of which stores a non-negative integer. Intuitively, for $b \in \{0, 1\}^{n-|I|}$, the cell $\text{TABLE}_I[b]$ stores (the pointer to) the unique node of $\mathcal{B}(f, \pi_I)$ associated via $F$ with function $f|_{x_{[n] \setminus I} = b}$. Hence, we may write $\text{TABLE}_I[x_{[n] \setminus I} = b]$ instead of $\text{TABLE}_I[b]$ to clearly indicate the value assigned to each variable $x_j$ for $j \in [n] \setminus I$. The purpose of $\text{TABLE}_I$ is to relate all subfunctions $f|_{x_{[n] \setminus I} = b}$ ($b \in \{0, 1\}^{n-|I|}$) to the corresponding nodes of $\mathcal{B}(f, \pi_I)$. We assume without loss of generality that the pointers to nodes of $\mathcal{B}(f, \pi_I)$ are non-negative integers and, in particular, those to the two terminal nodes corresponding to false and true are the integers 0 and 1, respectively. Thus, $\text{TABLE}_\emptyset$ is merely the truth table of $f$.

Algorithm FS computes $\text{TABLE}_I$ together with another data structure, $\text{NODE}_I$, and the value $\text{MINCOST}_I$ for all $I \subseteq [n]$ starting from $\text{TABLE}_\emptyset$ via dynamic programming. $\text{NODE}_I$ is the set of triples of (the pointers to) nodes, $(u, u_0, u_1) \in N \times (N \sqcup T) \times (N \sqcup T)$, in $\mathcal{B}(f, \pi_I)$, where $\text{var}(u) \in I$, and $(u, u_0)$ and $(u, u_1)$ are the 0-edge and 1-edge of $u$, respectively. Thus, $\text{NODE}_I$ contains the full structure of the subgraph of $\mathcal{B}(f, \pi_I)$ induced by $V_0 \sqcup \cdots \sqcup V_{|I|}$. The purpose of the $\text{NODE}_I$ is to prevent the algorithm from creating a redundant node, i.e., a node associated with the subfunction with which a certain existing node is associated (such a node (with its neighbors) should be stored in $\text{NODE}_I$). By the definition, $\text{NODE}_\emptyset$ is the empty set. We assume that $\text{NODE}_I$ is implemented with an appropriate data structure, such as a balanced tree, so that the time complexity required for membership testing and insertion is the order of logarithm in the number of triples stored in $\text{NODE}_I$.

More generally, for disjoint subset $I_1, \ldots, I_m \subseteq [n]$, $\text{TABLE}_{\langle I_1, \ldots, I_m \rangle}$ is an array with $2^{n-|I_1 \sqcup \cdots \sqcup I_m|}$ cells such that, for $b \in \{0, 1\}^{n-|I_1 \sqcup \cdots \sqcup I_m|}$, $\text{TABLE}_{\langle I_1, \ldots, I_m \rangle}[b]$ stores the nodes of $\mathcal{B}(f, \pi_{\langle I_1, \ldots, I_m \rangle})$ associated with the function $f|_{x_{[n] \setminus I_1 \sqcup \cdots \sqcup I_m} = b}$. $\text{NODE}_{\langle I_1, \ldots, I_m \rangle}$ is defined similarly for $\mathcal{B}(f, \pi_{\langle I_1, \ldots, I_m \rangle})$. For simplicity, we hereafter denote by $\mathcal{FS}(\langle I_1, \ldots, I_m \rangle)$ the quadruplet $(\pi_{\langle I_1, \ldots, I_m \rangle}, \text{MINCOST}_{\langle I_1, \ldots, I_m \rangle}, \text{TABLE}_{\langle I_1, \ldots, I_m \rangle}, \text{NODE}_{\langle I_1, \ldots, I_m \rangle})$.

### 2.3.2 Sketch of Algorithm FS [FS90]

Algorithm FS performs the following operations for $k = 1, \ldots, n$ in this order. For each $k$-element subset $I \subseteq [n]$, compute $\mathcal{FS}(\langle I \setminus i, i \rangle)$ from $\mathcal{FS}(\langle I \setminus i \rangle)$ for each $i \in I$ in the manner described later (note that, since the cardinality of the set $I \setminus i$ is $k - 1$, $\mathcal{FS}(\langle I \setminus i \rangle)$ has already been computed). Then set $\mathcal{FS}(I) \longleftarrow \mathcal{FS}(\langle I \setminus i^*, i^* \rangle)$, where $i^*$ is the index $i \in I$ that minimizes $\text{MINCOST}_{\langle I \setminus i, i \rangle}$, implying that $\pi_I$ is $\pi_{\langle I \setminus i^*, i^* \rangle}$. This is justified by Lemma 4.

To compute $\mathcal{FS}(\langle I \setminus i, i \rangle)$ from $\mathcal{FS}(\langle I \setminus i \rangle)$, do the following: First set $\text{NODE}_{\langle I \setminus i, i \rangle} \leftarrow \text{NODE}_{I \setminus i}$ and $\text{MINCOST}_{\langle I \setminus i, i \rangle} \leftarrow \text{MINCOST}_{I \setminus i}$ as their initial values. Then, for each $b \in \{0, 1\}^{n-|I|}$, set

$$u_0 \leftarrow \text{TABLE}_{I \setminus i}[x_{[n] \setminus I} = b, x_i = 0], \qquad u_1 \leftarrow \text{TABLE}_{I \setminus i}[x_{[n] \setminus I} = b, x_i = 1].$$

If $u_0 = u_1$, then store $u_0$ in $\text{TABLE}_{\langle I \setminus i, i \rangle}[b]$. Otherwise, test whether $(u, u_0, u_1)$ for some $u$ is a member of $\text{NODE}_{I \setminus i}$. If it is, store $u$ in the $\text{TABLE}_{\langle I \setminus i, i \rangle}[b]$; otherwise create a new triple $(u', u_0, u_1)$, insert it to $\text{NODE}_{\langle I \setminus i, i \rangle}$ and increment $\text{MINCOST}_{\langle I \setminus i, i \rangle}$. Here $u'$ must be different from any element in triples stored in $\text{NODE}_{\langle I \setminus i, i \rangle}$. Such $u'$ can be easily chosen by setting $u'$ to one plus the value of $\text{MINCOST}_{\langle I \setminus i, i \rangle}$ after the increment, since the $\text{MINCOST}_{\langle I \setminus i, i \rangle}$ is exactly the number of triples in $\text{NODE}_{\langle I \setminus i, i \rangle}$ and the numbers 0 and 1 are reserved for the terminal nodes. We call the above procedure *table compaction* with respect to $x_i$.

The complexity analysis is fairly simple. For each $k$, we need to compute $\mathcal{FS}(I)$ for $\binom{n}{k}$ possible $I$'s with $|I| = k$. For each $I$, it takes $O^*(2^{n-k})$ time since the the size of $\text{TABLE}_{I \setminus i}$ is $2^{n-k+1}$ and the operation to $\text{NODE}_{I \setminus i}$ takes a polynomial time in $n$. Thus, the total time is $\sum_{k=0}^{n} 2^{n-k} \binom{n}{k} = 3^n$, up to a polynomial factor. The point is that computing each $\mathcal{FS}(I)$ takes time linear to the size of $\text{TABLE}_{I \setminus i}$ up to a polynomial factor.

**Theorem 5 (Freedman and Supowit [FS90])** *Suppose that the truth table of $f\colon \{0,1\}^n \to \{0,1\}$ is given as input. Algorithm* FS *produces* $\mathcal{FS}([n])$ *in* $O^*(3^n)$ *time.*

## 2.4 Quantum Computation

We assume that readers have a basic knowledge of quantum computing (e.g., Refs. [NC00, KSV02, KLM07]). We provide only a lemma used to obtain our results.

The quantum search algorithm discovered by Grover [Gro96] has been generalized in many ways. For instance, Buhrman et al. [BCdWZ99] provided a small error version of quantum search, while Dürr and Høyer [DH96] devised a quantum algorithm that finds the minimum of a function. By combining these two algorithms, we obtain the following lemma (an adaptation of Corollary 2.3 in Ref. [LGM18]).

**Lemma 6 (Quantum Minimum Finding [DH96, BCdWZ99, LGM18])** *For every $\varepsilon > 0$ there exists a quantum algorithm that, for a function $f\colon [N] \to \mathbb{Z}$ given as an oracle, find an element $x \in [N]$ at which $f(x)$ achieves the minimum, with error probability at most $\varepsilon$ by making $O(\sqrt{N \log(1/\varepsilon)})$ queries.*

In this paper, the search space $N$ is exponentially large in $n$ and we are interested in exponential complexities, ignoring polynomial factors in them. We can thus safely assume $\varepsilon = 1/2^{p(n)}$ for a polynomial $p(n)$, so that the overhead is polynomially bounded. Since our algorithms use Lemma 6 constant times, their overall error probabilities are exponentially small. In the following proofs, we thus assume that $\varepsilon$ is exponentially small whenever we use Lemma 6, and do not explicitly analyze the error probability for simplicity.

Our algorithms assume that the quantum random access memory (QRAM) model [GLM08], which is commonly used in the literature when considering quantum algorithms. In the model, one can read contents from or write them into quantum memory in a superposition.

## 3 Quantum Algorithm with Divide-and-Conquer

We generalize Lemma 4 and Theorem 5 and use them in our quantum algorithm.

**Lemma 7** *For any disjoint subsets $I_1, \ldots, I_m, J \subseteq [n]$ with $J \neq \emptyset$ and any Boolean function $f\colon \{0,1\}^n \to \{0,1\}$, the following holds:*

$$\mathrm{MINCOST}_{\langle I_1,\ldots,I_m,J \rangle} = \min_{k \in J} \left( \mathrm{MINCOST}_{\langle I_1,\ldots,I_m,J\setminus\{k\} \rangle} + \mathsf{Cost}_k(f, \pi_{\langle I_1,\ldots,I_m,J\setminus\{k\},\{k\} \rangle}) \right)$$
$$= \min_{k \in J} \left( \mathrm{MINCOST}_{\langle I_1,\ldots,I_m,J\setminus\{k\},\{k\} \rangle} \right).$$

The proof of this lemma is very similar to that of Lemma 4 and deferred to Appendix A. Based on Lemma 7, we generalize Theorem 5 to obtain algorithm $\mathsf{FS}^*$ (its pseudo code is given in Appendix D).

**Lemma 8 (Classical Composition Lemma)** *For disjoint subsets $I_1, \ldots, I_m, J \subseteq [n]$ with $J \neq \emptyset$, there exists a deterministic algorithm $\mathsf{FS}^*$ that produces $\mathcal{FS}(\langle I_1, \ldots, I_m, J \rangle)$ from $\mathcal{FS}(\langle I_1, \ldots, I_m \rangle)$ for an underlying function $f\colon \{0,1\}^n \to \{0,1\}$ in $O^*\left(2^{n-|I_1 \sqcup \cdots \sqcup I_m \sqcup J|} \cdot 3^{|J|}\right)$ time and space. More generally, for each $k \in [|J|]$, the algorithm produces the set $\{\mathcal{FS}(\langle I_1, \ldots, I_m, K \rangle)\colon K \subseteq J, |K| = k\}$ from $\mathcal{FS}(\langle I_1, \ldots, I_m \rangle)$ in $O^*\left(2^{n-|I_1 \sqcup \cdots \sqcup I_m \sqcup J|} \sum_{j=0}^{k} 2^{|J|-j}\binom{|J|}{j}\right)$ time and space.*

Note that if $I_1 \sqcup \cdots \sqcup I_m = \emptyset$ and $J = [n]$, then we obtain Theorem 5.

*Proof.* We focuses on the simplest case of $m = 1$, for which our goal is to show an algorithm that produces $\mathcal{FS}(\langle I, J \rangle)$ from $\mathcal{FS}(I)$. It is straightforward to generalize the proof to the case of $m \geq 2$. The algorithm is based on Lemma 7. It performs the table compaction with respect to each variable in $\{x_i\colon i \in J\}$, starting from $\mathcal{FS}(I)$ [the difference is that the original algorithm performs the table compaction with respect to every variable, starting from $\mathcal{FS}(\emptyset)$]. The details of algorithm $\mathsf{FS}^*$ are shown in Appendix D. For each $j \in [|J|]$, $K \subseteq J$ with $|K| = j$, and $h \in K$, the time complexity of computing

8

$\mathcal{FS}(\langle I, K \rangle)$ from $\mathcal{FS}(\langle I, K-h \rangle)$ is linear to the size of $\text{TABLE}_{\langle I,K \rangle}$, i.e., $2^{n-|I|-j}$, up to a polynomial factor. The total time is thus, up to a polynomial factor,

$$\sum_{j=1}^{|J|} 2^{n-|I|-j} \binom{|J|}{j} < 2^{n-|I|-|J|} \sum_{j=0}^{|J|} 2^{|J|-j} \binom{|J|}{j} = 2^{n-|I \sqcup J|} \cdot 3^{|J|}.$$

If we stop the algorithm at $j = k$, then the algorithm produces the set $\{\mathcal{FS}(\langle I, K \rangle) \colon K \subseteq J, |K| = k\}$. The time complexity in this case is at most $2^{n-|I|-|J|} \sum_{j=0}^{k} 2^{|J|-j} \binom{|J|}{j}$, up to a polynomial factor.

Since the space complexity is trivially upper-bounded by the time complexity, we complete the proof. □

**Remark 1** *One may think that the actual space complexity could be much less than the time complexity. However, this is not the case. The size of $\text{TABLE}_{\langle I,K \rangle}$ is also the dominant factor determining the space complexity. When computing $\text{TABLE}_{\langle I,K \rangle}$ with $|K| = j$, it suffices to keep $\text{TABLE}_{\langle I,K \rangle}$ and $\text{TABLE}_{\langle I,K' \rangle}$ in memory, where $|K| = j$ and $|K'| = j - 1$. The space complexity is thus, up to a polynomial factor, the maximum of $2^{n-|I|-j} \binom{|J|}{j} + 2^{n-|I|-(j-1)} \binom{|J|}{j-1}$ over all $j \in [|J|]$, which is the same order as the time complexity.*

**Remark 2** *It is not difficult to see that the algorithm $\mathsf{FS}^*$ works even when the function $f$ has a multi-valued function: $f \colon \{0, 1\}^n \to Z \subset \mathbb{N}$. The only difference from the Boolean case is that the truth table maps each Boolean assignment to a value in $Z$. In this case, the algorithm produces a variant of an OBDD (called an MTBDD) of minimum size. In addition, our algorithm with two-line modifications to the table compaction rule in $\mathsf{FS}^*$ can construct a minimum ZDD [Min93] for a given Boolean function. The details are described in Appendix D. These hold for the quantum algorithms described later, since they perform table compaction by running $\mathsf{FS}^*$ as a subroutine.*

The following theorem is the basis of our quantum algorithms.

**Lemma 9 (Divide-and-Conquer)** *For any disjoint subsets $I_1, \dots, I_m, J \subseteq [n]$ with $J \neq \emptyset$ and any $k \in [|J|]$, it holds that*

$$\text{MINCOST}_{\langle I_1,\dots,I_m,J \rangle}[J] = \min_{K \colon K \subseteq J, |K|=k} \left( \text{MINCOST}_{\langle I_1,\dots,I_m,K \rangle}[K] + \text{MINCOST}_{\langle I_1,\dots,I_m,K,J\setminus K \rangle}[J \setminus K] \right). \quad (3)$$

*In particular, when $I_1 \sqcup \cdots \sqcup I_m = \emptyset$ and $J = [n]$, it holds that*

$$\text{MINCOST}_{[n]} = \min_{K \subseteq [n], |K|=k} \left( \text{MINCOST}_K + \text{MINCOST}_{\langle K,[n]\setminus K \rangle}[[n] \setminus K] \right). \quad (4)$$

*Proof.* We first prove the special case of $I_1 \sqcup \cdots \sqcup I_m = \emptyset$ and $J = [n]$. By the definition, we have

$$\text{MINCOST}_{[n]} = \sum_{j=1}^{n} \mathsf{Cost}_{\pi[j]}(f, \pi) = \sum_{j=1}^{k} \mathsf{Cost}_{\pi[j]}(f, \pi) + \sum_{j=k+1}^{n} \mathsf{Cost}_{\pi[j]}(f, \pi),$$

for the optimal permutation $\pi = \pi_{[n]}$. Let $K = \{\pi[1], \dots, \pi[k]\}$. By Lemma 3, the first sum is independent of how $\pi$ maps $\{k+1, \dots, n\}$ to $[n] \setminus K$. Thus, it is equal to the minimum of $\sum_{j=1}^{k} \mathsf{Cost}_{\pi_1[j]}(f, \pi)$ over all $\pi_1 \in \Pi(K)$, i.e., $\text{MINCOST}_K$. Similarly, the second sum is independent of how $\pi$ maps $[k]$ to $K$. Thus, it is equal to the minimum of $\sum_{j=k+1}^{n} \mathsf{Cost}_{\pi_2[j]}(f, \pi)$ over all $\pi_2 \in \Pi(\langle K, [n] \setminus K \rangle)$, i.e., $\text{MINCOST}_{\langle K,[n]\setminus K \rangle}[[n] \setminus K]$. This completes the proof of Eq. (4).

We can generalize this in a straightforward manner. Let $\pi = \pi_{\langle I_1,\dots,I_m,J \rangle}$ and $\ell = |I_1 \sqcup \cdots \sqcup I_m|$. Then, we have

$$\text{MINCOST}_{\langle I_1,\dots,I_m,J \rangle}[J] = \sum_{j=1}^{k} \mathsf{Cost}_{\pi[\ell+j]}(f, \pi) + \sum_{j=k+1}^{|J|} \mathsf{Cost}_{\pi[\ell+j]}(f, \pi).$$

By defining $K := \{\pi[\ell+1], \dots, \pi[\ell+k]\}$, the same argument as the special case of $\ell = 0$ implies that the first and second sums are $\text{MINCOST}_{\langle I_1,\dots,I_m,K \rangle}[K]$ and $\text{MINCOST}_{\langle I_1,\dots,I_m,K,J\setminus K \rangle}[J \setminus K]$, respectively. This completes the proof of Eq. (3). □

## 3.1 Simple Cases

We provide simple quantum algorithms on the basis of Lemma 9. The lemma states that, for each $k$, $\text{MINCOST}_{[n]}$ is the minimum of $\text{MINCOST}_K + \text{MINCOST}_{\langle K, [n] \setminus K \rangle}[[n] \setminus K]$ over all $K \subseteq [n]$ with $|K| = k$. To find $K$ from among $\binom{n}{k}$ possibilities that minimizes this amount, we use Lemma 6. To compute $\text{MINCOST}_K + \text{MINCOST}_{\langle K, [n] \setminus K \rangle}[[n] \setminus K] = \text{MINCOST}_{\langle K, [n] \setminus K \rangle}$, it suffices to first compute $\mathcal{FS}(K)$ (including $\text{MINCOST}_K$), and then $\mathcal{FS}(\langle K, [n] \setminus K \rangle)$ (including $\text{MINCOST}_{\langle K, [n] \setminus K \rangle}$) from $\mathcal{FS}(K)$. The time complexity of this computation is $O^*(2^{n-k} 3^k + 3^{n-k})$ by Lemma 8. Thus, for $k = \alpha n$ with $\alpha \in (0, 1)$ fixed later, the total time complexity up to a polynomial factor is

$$T(n) = \sqrt{\binom{n}{\alpha n}} \left( 2^{(1-\alpha)n} 3^{\alpha n} + 3^{(1-\alpha)n} \right) \lessgtr 2^{\frac{1}{2} \mathbf{H}(\alpha)n} \left\{ 2^{[(1-\alpha)+\alpha \log_2 3]n} + 2^{[(1-\alpha) \log_2 3]n} \right\}.$$

To balance the both terms, we set $(1 - \alpha) + \alpha \log_2 3 = (1 - \alpha) \log_2 3$ and obtain $\alpha = \frac{\log_2 3 - 1}{2 \log_2 3 - 1} \approx 0.269577$. Let $\alpha^* := \frac{\log_2 3 - 1}{2 \log_2 3 - 1}$. We have $\min_{\alpha \in [0,1]} T(n) = O\left(2^{\frac{1}{2} \mathbf{H}(\alpha^*)n + (1-\alpha^*)n + \alpha^*(\log_2 3)n}\right) = O(\gamma_0^n)$, where $\gamma_0 = 2.98581 \ldots$. This slightly improves the classical best bound $O^*(3^n)$ on the time complexity. To improve the bound further, we introduce a preprocess that classically computes $\mathcal{FS}(K)$ for every $K$ with $|K| = \alpha n$ ($\alpha \in (0, 1)$) by algorithm FS*. By Lemma 7, the preprocessing time is then

$$\sum_{j=1}^{\alpha n} 2^{n-j} \cdot \binom{n}{j} \leq \alpha n \cdot \max_{j \in [\alpha n]} 2^{n-j} \binom{n}{j} \lessgtr \begin{cases} 2^{(1-\alpha)n + \mathbf{H}(\alpha)n} & (\alpha < 1/3) \\ 2^{\frac{2}{3}n + \mathbf{H}(1/3)n} & (\alpha \geq 1/3), \end{cases}$$

since $2^{n-j} \binom{n}{j}$ increases when $j < n/3$ and decreases otherwise. Note that once this preprocess is completed, we can use $\mathcal{FS}(K)$ for free and assume that the cost for accessing $\mathcal{FS}(K)$ is polynomially bounded for all $K \subseteq [n]$ with $|K| = \alpha n$.

Then, assuming that $\alpha < 1/3$, the total time complexity up to a polynomial factor is

$$T(n) = \sum_{j=1}^{\alpha n} 2^{n-j} \cdot \binom{n}{j} + \sqrt{\binom{n}{\alpha n}} \left( n^{O(1)} + 3^{(1-\alpha)n} \right) \lessgtr 2^{[(1-\alpha) + \mathbf{H}(\alpha)]n} + 2^{[\frac{1}{2} \mathbf{H}(\alpha) + (1-\alpha) \log_2 3]n}.$$

To balance the both terms, we set $(1 - \alpha) + \mathbf{H}(\alpha) = \frac{1}{2}\mathbf{H}(\alpha) + (1 - \alpha) \log_2 3$ and obtain the solution $\alpha^* := 0.274863 \ldots$, which is less than $1/3$ as we assumed. At $\alpha = \alpha^*$, we have $T(n) \lessgtr 2^{[(1-\alpha^*) + \mathbf{H}(\alpha^*)]n} = O^*(\gamma_1^n)$, where $\gamma_1$ is at most $2.97625$ ($< \gamma_0$). Thus, introducing the preprocess improves the complexity bound.

Appendix B shows that, by using Lemma 9 twice, it is possible to improve the complexity bound further.

## 3.2 General Case

We can improve this bound further by applying Lemma 9 $k$ times. We denote the resulting algorithm with parameters $k$ and $\alpha := (\alpha_1, \ldots, \alpha_k)$ by OptOBDD$(k, \alpha)$ where $0 < \alpha_1 < \cdots < \alpha_k < 1$. Its pseudo code is given in Appendix D. In addition, we assume $\alpha_{k+1} = 1$ in the following complexity analysis.

To simplify notations, define two function as follows: for $x, y \in \mathbb{R}$ such that $0 < x < y < 1$,

$$f(x, y) := \frac{1}{2} y \cdot \mathbf{H}\left(\frac{x}{y}\right) + g(x, y), \qquad g(x, y) := (1 - y) + (y - x) \log_2(3).$$

The time required for the preprocess is $\sum_{\ell=1}^{\alpha_1 n} 2^{n-\ell} \cdot \binom{n}{\ell}$ up to a polynomial factor. Thus, the total time

complexity can be described as the following recurrence:

$$T(n) = \sum_{\ell=1}^{\alpha_1 n} 2^{n-\ell} \cdot \binom{n}{\ell} + L_{k+1}(n), \tag{5}$$

$$L_{j+1}(n) = \sqrt{\binom{\alpha_{j+1}n}{\alpha_j n}} \left(L_j(n) + 2^{(1-\alpha_{j+1})n} 3^{(\alpha_{j+1}-\alpha_j)n}\right) = \sqrt{\binom{\alpha_{j+1}n}{\alpha_j n}} \left(L_j(n) + 2^{g(\alpha_j, \alpha_{j+1})n}\right), \tag{6}$$

$$L_1(n) = O^*(1), \tag{7}$$

for each $j \in [k]$. Since $L_1(n) = O^*(1)$, we have $L_2(n) \lessapprox \sqrt{\binom{\alpha_2 n}{\alpha_1 n}} \cdot 2^{g(\alpha_1, \alpha_2)n} \lessapprox 2^{f(\alpha_1, \alpha_2)n}$. By setting $f(\alpha_1, \alpha_2) = g(\alpha_2, \alpha_3)$, we have

$$L_3(n) = \sqrt{\binom{\alpha_3 n}{\alpha_2 n}} \cdot (L_2(n) + 2^{g(\alpha_2, \alpha_3)n}) \lessapprox \sqrt{\binom{\alpha_3 n}{\alpha_2 n}} \cdot 2^{g(\alpha_2, \alpha_3)n} \lessapprox 2^{f(\alpha_2, \alpha_3)n}.$$

In general, for $j = 2, \ldots, k$, setting $f(\alpha_{j-1}, \alpha_j) = g(\alpha_j, \alpha_{j+1})$ yields $L_{j+1}(n) \lessapprox 2^{f(\alpha_j, \alpha_{j+1})n}$. Therefore, the total complexity [Eq. (5)] is

$$T(n) \lessapprox \sum_{\ell=1}^{\alpha_1 n} 2^{n-\ell} \cdot \binom{n}{\ell} + 2^{f(\alpha_k, \alpha_{k+1})n} \lessapprox 2^{(1-\alpha_1)n + \mathbf{H}(\alpha_1)n} + 2^{f(\alpha_k, 1)n}.$$

To optimize the right-hand side, we set parameters so that $1 - \alpha_1 + \mathbf{H}(\alpha_1) = f(\alpha_k, 1)$.

In summary, we need to find the values of parameters $\alpha_1, \ldots, \alpha_k$ that satisfy the following system of equations:

$$1 - \alpha_1 + \mathbf{H}(\alpha_1) = f(\alpha_k, 1), \tag{8}$$

$$f(\alpha_{j-1}, \alpha_j) = g(\alpha_j, \alpha_{j+1}) \quad (j = 2, \ldots, k). \tag{9}$$

By numerically solving this system of equations, we obtain $T(n) = O(\gamma_k^n)$, where $\gamma_k$ and the corresponding $\alpha_i$'s for $k = 1, \ldots, 6$ are shown in Tab. 1 in Appendix C ($\alpha_1 < 1/3$ is satisfied as we assumed). The value of $\gamma_k$ becomes smaller as $k$ increases. However, incrementing $k$ beyond 6 provides only negligible improvement of $\gamma_k$. The value of $\gamma_6$ is at most 2.83728. Since the space complexity is trivially upper-bounded by the time complexity, we have the following theorem.

**Theorem 10** *For the truth table of $f : \{0, 1\}^n \to \{0, 1\}$ given as input, the quantum algorithm* $\mathsf{OptOBDD}(k, \alpha)$ *produces $\mathcal{FS}([n])$ with probability $1 - \exp(-\Omega(n))$ in $O^*(\gamma^n)$ time and space, where the constant $\gamma$ is at most 2.83728, which is achieved by setting $k = 6$ and*

$$\alpha = (0.183791, 0.183802, 0.183974, 0.186131, 0.206480, 0.343573).$$

## 4 Quantum Algorithm with Composition

### 4.1 Quantum Composition Lemma

Recall that the classical composition lemma (Lemma 8) states that algorithm $\mathsf{FS}$ can be generalized to compute $\mathcal{FS}(\langle I_1, \ldots, I_m, J \rangle)$ from $\mathcal{FS}(\langle I_1, \ldots, I_m \rangle)$. By generalizing the quantum algorithm given in Theorem 10, we now provide a quantum version of Lemma 8, called the *quantum composition lemma*.

**Lemma 11 (Quantum Composition: Base Part)** *For any disjoint subsets $I_1, \ldots, I_m, J \subseteq [n]$, there exists a quantum algorithm $\mathsf{OptOBDD}_\Gamma^*(k, \alpha)$ that, produces $\mathcal{FS}(\langle I_1, \ldots, I_m, J \rangle)$ with probability $1 - \exp(-\Omega(n))$ from $\mathcal{FS}(\langle I_1, \ldots, I_m \rangle)$ for an underlying function $f : \{0, 1\}^n \to \{0, 1\}$ in $O^*\left(2^{n - |I_1 \sqcup \cdots \sqcup I_m \sqcup J|} \cdot \gamma^{|J|}\right)$ time and space, where $\gamma$ is the constant defined in Theorem 10 (the subscript $\Gamma$ of $\mathsf{OptOBDD}_\Gamma^*(k, \alpha)$ indicates the subroutine used in the algorithm, and $\Gamma$ is set to the* composable *deterministic algorithm $\mathsf{FS}^*$ in this theorem).*

We provide a pseudo code of algorithm $\mathsf{OptOBDD}^*_\Gamma(k, \alpha)$ in Appendix D, where $\Gamma$ appears in line 17.

*Proof.* Since the space complexity is trivially upper-bounded by the time complexity, we analyze the time complexity in the following. For simplicity, we assume $m = 1$ and write just $I$ instead of $I_1$. It is straightforward to generalize to the case of $m \geq 2$. We now provide the algorithm $\mathsf{OptOBDD}^*_\Gamma(k, \alpha)$ that produces $\mathcal{FS}(\langle I, J \rangle)$ from $\mathcal{FS}(\langle I \rangle)$, where the subroutine $\Gamma$ used in line 17 is set to algorithm $\mathsf{FS}^*$.

As parameters, the algorithm has an integer $k \in \mathbb{N}$ and a vector $\alpha := (\alpha_1, \ldots, \alpha_k) \in \mathbb{R}^k$ such that $0 < \alpha_1 < \cdots < \alpha_k < 1$. Set $k$ and $\alpha$ to the same values assumed in the algorithm in Theorem 10. In addition, we assume $\alpha_{k+1} = 1$ in the following. Let $n' = |J|$. In the preprocessing phase, the algorithm computes the set $\{\mathcal{FS}(\langle I, K \rangle) \colon K \subseteq J, |K| = \alpha_1 n'\}$ from $\mathcal{FS}(I)$ based on Lemma 8. The time complexity of the preprocess is, up to a polynomial factor, $2^{n-|I|-n'} \sum_{\ell=1}^{\alpha_1 n'} 2^{n'-\ell} \binom{n'}{\ell}$. Thus, the total time complexity is expressed as

$$T'(n, n') = 2^{n-|I|-n'} \sum_{\ell=1}^{\alpha_1 n'} 2^{n'-\ell} \binom{n'}{\ell} + L'_{k+1}(n, n'), \tag{10}$$

where $L'_{k+1}(n, n')$ is the time taken to perform the rest of the algorithm.

Based on Lemma 9, the algorithm proceeds as the one given in Theorem 10 (the theorem corresponds to the special case of $I = \emptyset$ and $J = [n]$). The complexity $L'_{k+1}(n, n')$ is then expressed by the following recurrence:

$$L'_{j+1}(n, n') = \sqrt{\binom{\alpha_{j+1} n'}{\alpha_j n'}} \left( L'_j(n, n') + 2^{n-|I|-\alpha_{j+1} n'} 3^{(\alpha_{j+1}-\alpha_j)n'} \right) \quad [j \in [k]],$$
$$L'_1(n, n') = O^*(1).$$

In the following, we prove by induction that $T'(n, n') = 2^{n-|I|-n'} T(n')$, where the function $T(\cdot)$ is defined in Eq. (5). Since $n' = |J|$ and $T(n') = O^*(\gamma^{n'})$, this completes the proof for $m = 1$.

Since $L'_1(n, n') = O^*(1)$, we have

$$L'_2(n, n') = 2^{n-|I|-n'} \sqrt{\binom{\alpha_2 n'}{\alpha_1 n'}} 2^{(1-\alpha_2)n'} 3^{(\alpha_2-\alpha_1)n'} = 2^{n-|I|-n'} L_2(n'),$$

where the function $L_2(\cdot)$ is defined in Eq. (6) for $j = 1$. This is the base case of the induction. Then, assuming that $L'_j(n, n') = 2^{n-|I|-n'} L_j(n')$, we have

$$L'_{j+1}(n, n') = \sqrt{\binom{\alpha_{j+1} n'}{\alpha_j n'}} \left( 2^{n-|I|-n'} L_j(n') + 2^{n-|I|-\alpha_{j+1} n'} 3^{(\alpha_{j+1}-\alpha_j)n'} \right)$$
$$= 2^{n-|I|-n'} \sqrt{\binom{\alpha_{j+1} n'}{\alpha_j n'}} \left( L_j(n') + 2^{n'-\alpha_{j+1} n'} 3^{(\alpha_{j+1}-\alpha_j)n'} \right) = 2^{n-|I|-n'} L'_{j+1}(n').$$

Therefore, it holds that $L'_{k+1}(n, n') = 2^{n-|I|-n'} L_{k+1}(n')$ by induction. Then, it follows from Eq. (10) that

$$T'(n, n') = 2^{n-|I|-n'} \sum_{\ell=1}^{\alpha_1 n'} 2^{n'-\ell} \binom{n'}{\ell} + 2^{n-|I|-n'} L_{k+1}(n') = 2^{n-|I|-n'} T(n').$$

Since $T(n') = T(|J|) = O^*(\gamma^{|J|})$ by Theorem 10, the lemma follows. $\qquad\square$

**Lemma 12 (Quantum Composition: Induction Part)** *Suppose that $\Gamma$ is a quantum algorithm that, for any disjoint subsets $I_1, \ldots, I_m, J \subseteq [n]$ with $J \neq \emptyset$, produces $\mathcal{FS}(\langle I_1, \ldots, I_m, J \rangle)$ from $\mathcal{FS}(\langle I_1, \ldots, I_m \rangle)$ with probability $1 - \exp(-\Omega(n))$ in $O^* \left( 2^{n-|I_1 \sqcup \cdots \sqcup I_m \sqcup J|} \cdot \gamma^{|J|} \right)$ time and space for an underlying function*

12

$f\colon \{0,1\}^n \to \{0,1\}$. *Then, for any possible $k \in \mathbb{N}$ and $\boldsymbol{\alpha} \in \mathbb{R}^k$ and for any disjoint subsets $I_1, \ldots, I_m, J \subseteq [n]$ with $J \neq \emptyset$, $\mathsf{OptOBDD}^*_\Gamma(k, \boldsymbol{\alpha})$ produces $\mathcal{FS}(\langle I_1, \ldots, I_m, J \rangle)$ from $\mathcal{FS}(\langle I_1, \ldots, I_m \rangle)$ with probability $1 - \exp(-\Omega(n))$ in $O^*\left(2^{n - |I_1 \sqcup \cdots \sqcup I_m \sqcup J|} \cdot \beta_k^{|J|}\right)$ time and space for the function $f$, where $\beta_k^n$ upper-bounds, up to a polynomial factor, the time complexity of $\mathsf{OptOBDD}^*_\Gamma(k, \boldsymbol{\alpha})$ for input $\{I = \emptyset, J = [n], \mathcal{FS}([n])\}$, that is, $T(n) = O^*(\beta_k^n)$ for $T(n)$ that satisfies the following recurrence:*

$$T(n) = \sum_{\ell=1}^{\alpha_1 n} 2^{n-\ell} \binom{n}{\ell} + L_{k+1}, \tag{11}$$

$$L_{j+1} = \sqrt{\binom{\alpha_{j+1} n}{\alpha_j n}} \left(L_j + 2^{(1-\alpha_{j+1})n} \gamma^{(\alpha_{j+1} - \alpha_j)n}\right) = \sqrt{\binom{\alpha_{j+1} n}{\alpha_j n}} \left(L_j + 2^{g_\gamma(\alpha_j, \alpha_{j+1})}\right) \quad [j \in [k]], \tag{12}$$

$$L_1 = O^*(1), \tag{13}$$

*where $f(x, y) := \frac{1}{2} y \cdot \mathbf{H}\left(\frac{x}{y}\right) + g_\gamma(x, y)$ and $g_\gamma(x, y) := (1 - y) + (y - x) \log_2 \gamma$.*

*Proof.* Recall that algorithm $\mathsf{FS}^*$ is used as a subroutine in $\mathsf{OptOBDD}(k, \boldsymbol{\alpha})$ provided in Theorem 10. Since the input and output of $\Gamma$ assumed in the statement are the same as those of algorithm $\mathsf{FS}^*$, one can use $\Gamma$ instead of algorithm $\mathsf{FS}^*$ in $\mathsf{OptOBDD}(k, \boldsymbol{\alpha})$ (compromising on an exponentially small error probability). Let $\mathsf{OptOBDD}_\Gamma(k, \boldsymbol{\alpha})$ be the resulting algorithm. Then, one can see that the time complexity $T(n)$ of $\mathsf{OptOBDD}_\Gamma(k, \boldsymbol{\alpha})$ satisfies the recurrence: Eqs. (11)-(13), which are obtained by just replacing $g(x, y)$ with $g_\gamma(x, y)$ in Eqs. (5)-(7). Suppose that $T(n) = O^*(\beta_k^n)$ follows from the recurrence.

Next, we generalize $\mathsf{OptOBDD}_\Gamma(k, \boldsymbol{\alpha})$ so that it produces $\mathcal{FS}(\langle I_1, \ldots, I_m, J \rangle)$ from $\mathcal{FS}(\langle I_1, \ldots, I_m \rangle)$ for any disjoint subsets $I_1, \ldots, I_m, J \subseteq [n]$ with $J \neq \emptyset$. The proof is very similar to that of Lemma 11. The only difference is that the time complexity of $\Gamma$ is $O^*\left(2^{n - |I_1 \sqcup \cdots \sqcup I_m \sqcup J|} \cdot \gamma^{|J|}\right)$, instead of $O^*\left(2^{n - |I_1 \sqcup \cdots \sqcup I_m \sqcup J|} \cdot 3^{|J|}\right)$. Namely, when $m = 1$ and $n' = |J|$, the time complexity of $\mathsf{OptOBDD}^*_\Gamma(k, \boldsymbol{\alpha})$ satisfies the following recurrence: for each $j \in [n]$,

$$T'(n, n') = 2^{n - |I| - n'} \sum_{\ell=1}^{\alpha_1 n'} 2^{n' - \ell} \binom{n'}{\ell} + L'_{k+1}(n, n'),$$

$$L'_{j+1}(n, n') = \sqrt{\binom{\alpha_{j+1} n'}{\alpha_j n'}} \left(L'_j(n, n') + 2^{n - |I| - \alpha_{j+1} n'} \gamma^{(\alpha_{j+1} - \alpha_j) n'}\right) \quad [j \in [n]],$$

$$L'_1(n, n') = O^*(1),$$

from which it follows that $T'(n, n') = 2^{n - |I| - n'} T(n') = O^*\left(2^{n - |I \sqcup J|} \cdot \beta_k^{|J|}\right)$. It is straightforward to generalize to the case of $m \geq 2$. $\qquad\square$

## 4.2 The Final Algorithm

Lemmas 11 and 12 naturally lead to the following algorithm. We first define $\Gamma_1$ as $\mathsf{OptOBDD}^*_{\mathsf{FS}^*}(k^{(0)}, \boldsymbol{\alpha}^{(0)})$ for some $k^{(0)} \in \mathbb{N}$ and $\boldsymbol{\alpha}^{(0)} \in \mathbb{R}^{k^{(0)}}$. Then, we define $\Gamma_2$ as $\mathsf{OptOBDD}^*_{\Gamma_1}(k^{(1)}, \boldsymbol{\alpha}^{(1)})$ for some $k^{(1)} \in \mathbb{N}$ and $\boldsymbol{\alpha}^{(1)} \in \mathbb{R}^{k^{(1)}}$. In this way, we can define $\Gamma_{i+1}$ as $\mathsf{OptOBDD}^*_{\Gamma_i}(k^{(i)}, \boldsymbol{\alpha}^{(i)})$ for some $k^{(i)} \in \mathbb{N}$ and $\boldsymbol{\alpha}^{(i)} \in \mathbb{R}^{k^{(i)}}$.

Fix $k^{(i)} = 6$ for every $i$. To optimize parameter $\boldsymbol{\alpha}^{(i)}$ for each $i$, we need to set parameter $\boldsymbol{\alpha}^{(i)} = (\alpha_1^{(i)}, \ldots, \alpha_6^{(i)}) \in [0, 1]^6$ so that it satisfies the system of equations

$$1 - \alpha_1^{(i)} + \mathbf{H}(\alpha_1^{(i)}) = f(\alpha_6^{(i)}, 1) \tag{14}$$

$$f(\alpha_{j-1}^{(i)}, \alpha_j^{(i)}) = g_\gamma(\alpha_j^{(i)}, \alpha_{j+1}^{(i)}) \qquad (j = 2, \ldots, 6). \tag{15}$$

By solving this system of equations numerically for $\gamma = 3$, we have $\beta_6 < 2.83728$ as shown in Theorem 10. Then, solving the system of equations numerically with $\gamma = 2.83728$, we have $\beta_6 <$

2.79364. In this way, we obtain a certain $\gamma$ less than 2.77286 at the tenth composition (see Tab. 2 in Appendix C). We therefore obtain the following theorem.

**Theorem 13** *For the truth table of $f\colon \{0,1\}^n \to \{0,1\}$ given as input, the above quantum algorithm produces $\mathcal{F}\mathcal{S}([n])$ in $O^*(\gamma^n)$ time and space with probability $1 - \exp(-\Omega(n))$, where the constant $\gamma$ is at most 2.77286.*

# Appendix

## A  Proofs of Lemmas

*Proof of Lemma 4.* The last equality holds since, by the definition, $\mathrm{MINCOST}_{\langle I\backslash k, k\rangle}$ equals the sum of $\mathrm{MINCOST}_{I\backslash k}$ and $\mathsf{Cost}_k(f, \pi_{\langle I\backslash k, k\rangle})$.

To show the first equality, assume $k = \pi_I[|I|]$. By the definition, we have

$$\mathrm{MINCOST}_I = \sum_{i\in I}\mathsf{Cost}_i(f, \pi_I) = \sum_{i\in I\backslash\{k\}}\mathsf{Cost}_i(f, \pi_I) + \mathsf{Cost}_k(f, \pi_I).$$

The first term is equal to $\mathrm{MINCOST}_{I\backslash k}$, since otherwise there exists $\pi' \in \Pi(I)$ with $\pi'[|I|] = k$ and $\pi'[j] \neq \pi_I[j]$ for some $j \in [|I| - 1]$ such that

$$\sum_{i\in I}\mathsf{Cost}_i(f, \pi') = \sum_{i\in I\backslash\{k\}}\mathsf{Cost}_i(f, \pi') + \mathsf{Cost}_k(f, \pi')$$
$$< \sum_{i\in I\backslash\{k\}}\mathsf{Cost}_i(f, \pi_I) + \mathsf{Cost}_k(f, \pi_I) = \mathrm{MINCOST}_I,$$

which contradicts the definition of $\mathrm{MINCOST}_I$, where we use $\mathsf{Cost}_k(f, \pi') = \mathsf{Cost}_k(f, \pi_I)$ by Lemma 3.

The remaining term $\mathsf{Cost}_k(f, \pi_I)$ is equal to $\mathsf{Cost}_k(f, \pi_{\langle I\backslash k, k\rangle})$ by Lemma 3. This completes the proof. $\square$

*Proof of Lemma 7.* We focus on the simplest case of $m = 1$ and write just $I$ instead of $I_1$, since it is straightforward to generalize the proof to the case of $m \geq 2$.

The last equality holds since, by the definition, $\mathrm{MINCOST}_{\langle I, J\backslash\{k\}, \{k\}\rangle}$ equals the sum of $\mathrm{MINCOST}_{I, J\backslash\{k\}}$ and $\mathsf{Cost}_k(f, \pi_{\langle I, J\backslash\{k\}, \{k\}\rangle})$.

To show the first equality, assume $k = \pi_{\langle I, J\rangle}[|I \sqcup J|]$. By the definition, we have

$$\mathrm{MINCOST}_{\langle I, J\rangle} = \sum_{i\in I\sqcup J}\mathsf{Cost}_i(f, \pi_{\langle I, J\rangle}) = \sum_{i\in I\sqcup J\backslash\{k\}}\mathsf{Cost}_i(f, \pi_{\langle I, J\rangle}) + \mathsf{Cost}_k(f, \pi_{\langle I, J\rangle}).$$

The first term is equal to $\mathrm{MINCOST}_{\langle I, J\backslash\{k\}\rangle}$, since otherwise there exists $\pi' \in \Pi(\langle I, J\rangle)$ with $\pi'[|I \sqcup J|] = k$ and $\pi'[j] \neq \pi_{\langle I, J\rangle}[j]$ for some $j \in [|I \sqcup J| - 1]$ such that

$$\sum_{i\in I\sqcup J}\mathsf{Cost}_i(f, \pi') = \sum_{i\in I\sqcup J\backslash\{k\}}\mathsf{Cost}_i(f, \pi') + \mathsf{Cost}_k(f, \pi')$$
$$< \sum_{i\in I\sqcup J\backslash\{k\}}\mathsf{Cost}_i(f, \pi_{\langle I, J\rangle}) + \mathsf{Cost}_k(f, \pi_{\langle I, J\rangle}) = \mathrm{MINCOST}_{\langle I, J\rangle},$$

which contradicts the definition of $\mathrm{MINCOST}_{\langle I, J\rangle}$, where we use $\mathsf{Cost}_k(f, \pi') = \mathsf{Cost}_k(f, \pi_{\langle I, J\rangle})$ by Lemma 3. The remaining term $\mathsf{Cost}_k(f, \pi_{\langle I, J\rangle})$ is equal to $\mathsf{Cost}_k(f, \pi_{\langle I, J-k, k\rangle})$ by Lemma 3. This completes the proof. $\square$

# B The Case of Two Parameters

To improve the complexity bound further, we use Lemma 9 recursively. Let $k_1$ and $k_2$ be parameters fixed later such that $0 < k_1 < k_2 < n$. By applying the lemma once, we have

$$\text{MINCOST}_{[n]} = \min_{K_2 \subseteq [n], |K_2| = k_2} \left( \text{MINCOST}_{K_2} + \text{MINCOST}_{\langle K_2, [n] \setminus K_2 \rangle}[[n] \setminus K_2] \right).$$

Then, we apply the lemma to $\text{MINCOST}_{K_2}$ to obtain

$$\text{MINCOST}_{K_2} = \min_{K_1 \subseteq K_2, |K_1| = k_1} \left( \text{MINCOST}_{K_1} + \text{MINCOST}_{\langle K_1, K_2 \setminus K_1 \rangle}[K_2 \setminus K_1] \right).$$

To find the optimal $K_1$ from among $\binom{k_2}{k_1}$ possibilities and the optimal $K_2$ from among $\binom{n}{k_2}$, we use Lemma 6. As in the previous case, we perform the classical preprocess that computes $\mathcal{FS}(K)$ for all $K \subseteq [n]$ with $|K| = k_1$.

If we set $k_1 = \alpha_1 n$ and $k_2 = \alpha_2 n$, assuming that $\alpha_1 < 1/3$, the total time complexity (up to a polynomial factor) is

$$T(n) = \sum_{\ell=1}^{\alpha_1 n} 2^{n-\ell} \cdot \binom{n}{\ell} + L_3(n) \lessapprox 2^{(1-\alpha_1)n + \mathbf{H}(\alpha_1)n} + L_3(n), \tag{16}$$

where

$$L_3(n) = \sqrt{\binom{n}{\alpha_2 n}} \left( L_2(n) + 3^{(1-\alpha_2)n} \right) \lessapprox 2^{\frac{1}{2}\mathbf{H}(\alpha_2)n} (L_2(n) + 2^{(1-\alpha_2)(\log_2 3)n}), \tag{17}$$

$$L_2(n) = \sqrt{\binom{\alpha_2 n}{\alpha_1 n}} \left( L_1(n) + 2^{(1-\alpha_2)n} 3^{(\alpha_2-\alpha_1)n} \right) \lessapprox 2^{\frac{1}{2}\mathbf{H}(\alpha_1/\alpha_2)\alpha_2 n} (L_1(n) + 2^{(1-\alpha_2)n + (\alpha_2-\alpha_1)(\log_2 3)n}), \tag{18}$$

$$L_1(n) = O^*(1). \tag{19}$$

Since we are not interested in polynomial factors in $T(n)$, we ignore them in providing an analysis of this system below.

From Eqs. (18) and (19), $L_2(n)$ is at most $2^{\frac{1}{2}\mathbf{H}(\alpha_1/\alpha_2)\alpha_2 n} \cdot 2^{(1-\alpha_2)n + (\alpha_2-\alpha_1)(\log_2 3)n}$. Suppose that $L_2(n)$ equals this upper bound, since our goal is to upper-bound the complexity. To balance the two terms on the right-hand side of Eq. (17), we set $L_2(n) = 2^{(1-\alpha_2)(\log_2 3)n}$, which implies

$$\frac{1}{2}\alpha_2 \mathbf{H}(\alpha_1/\alpha_2) + (1 - \alpha_2) + (\alpha_2 - \alpha_1)\log_2 3 = (1 - \alpha_2)\log_2 3. \tag{20}$$

On the condition that this holds, $L_3(n)$ is at most $2^{\frac{1}{2}\mathbf{H}(\alpha_2)n} 2^{(1-\alpha_2)(\log_2 3)n}$. Suppose again that $L_3(n)$ equals this upper bound. To balance the two terms on the right-hand side in Eq. (16), we have $L_3(n) = 2^{(1-\alpha_1)n + \mathbf{H}(\alpha_1)n}$, implying that

$$(1 - \alpha_1) + \mathbf{H}(\alpha_1) = \frac{1}{2}\mathbf{H}(\alpha_2) + (1 - \alpha_2)\log_2 3. \tag{21}$$

By numerically solving Eqs. (20) and (21), we have $\alpha_1^* = 0.192755$ and $\alpha_2^* = 0.334571$. Note that $\alpha_1^*$ is less than $1/3$, as we assumed. For $\alpha_1 = \alpha_1^*$ and $\alpha_2 = \alpha_2^*$,

$$T(n) \lessapprox 2^{[(1-\alpha_1^*) + \mathbf{H}(\alpha_1^*)]n} = O^*(\gamma_2),$$

where $\gamma_2 = 2.8569 < \gamma_1$.

# C  Numerically Optimization Data

Table 1: Values of $\gamma_k$ and the corresponding $\alpha_i$'s of algorithm $\mathsf{OptOBDD}(k,\alpha)$: Each value is written with 6 digits, but the actual calculation is done with 20-digit precision.

| $k$ | $\gamma_k$ | $\alpha_1$ | $\alpha_2$ | $\alpha_3$ | $\alpha_4$ | $\alpha_5$ | $\alpha_6$ |
|---|---|---|---|---|---|---|---|
| 1 | 2.97625 | 0.274862 | — | — | — | — | — |
| 2 | 2.85690 | 0.192754 | 0.334571 | — | — | — | — |
| 3 | 2.83925 | 0.184664 | 0.205128 | 0.342677 | — | — | — |
| 4 | 2.83744 | 0.183859 | 0.186017 | 0.206375 | 0.343503 | — | — |
| 5 | 2.83729 | 0.183795 | 0.183967 | 0.186125 | 0.206474 | 0.343569 | — |
| 6 | 2.83728 | 0.183791 | 0.183802 | 0.183974 | 0.186131 | 0.206480 | 0.343573 |

Table 2: Values of $\gamma$ and the corresponding $\alpha_i$'s of algorithm $\mathsf{OptOBDD}^*_\Gamma(k,\alpha)$: Each value is written with 6 digits, but the actual calculation is done with 20-digit precision.

| $\gamma$ | $\beta_6$ | $\alpha_1$ | $\alpha_2$ | $\alpha_3$ | $\alpha_4$ | $\alpha_5$ | $\alpha_6$ |
|---|---|---|---|---|---|---|---|
| 3 | 2.83728 | 0.183792 | 0.183802 | 0.183974 | 0.186132 | 0.206480 | 0.343573 |
| 2.83728 | 2.79364 | 0.165753 | 0.165759 | 0.165857 | 0.167339 | 0.183883 | 0.312741 |
| 2.79364 | 2.77981 | 0.160487 | 0.160491 | 0.160574 | 0.16189 | 0.177376 | 0.303603 |
| 2.77981 | 2.77521 | 0.158777 | 0.15878 | 0.158859 | 0.160124 | 0.175273 | 0.300622 |
| 2.77521 | 2.77366 | 0.158203 | 0.158207 | 0.158284 | 0.159532 | 0.174568 | 0.299621 |
| 2.77366 | 2.77313 | 0.158009 | 0.158013 | 0.158089 | 0.159332 | 0.174330 | 0.299282 |
| 2.77313 | 2.77295 | 0.157943 | 0.157947 | 0.158023 | 0.159264 | 0.174249 | 0.299166 |
| 2.77295 | 2.77289 | 0.15792 | 0.157924 | 0.158000 | 0.159241 | 0.174221 | 0.299127 |
| 2.77289 | 2.77287 | 0.157913 | 0.157916 | 0.157992 | 0.159233 | 0.174212 | 0.299114 |
| 2.77287 | 2.77286 | 0.157910 | 0.157914 | 0.157990 | 0.159230 | 0.174208 | 0.299109 |

# D  Pseudo Codes of Algorithms

---

**Algorithm FS***: Composable variant of algorithm FS. "$A \leftarrow B$" means that $B$ is substituted for $A$.

---

**Input:** disjoint subsets $I, J \in [n]$ and $\mathcal{FS}(I)$
**Output:** $\mathcal{FS}(\langle I, J \rangle)$

```
 1  Function Main()
 2      for k := 1 to |J| do
 3          for each k-element subset K ⊆ J do
 4              MINCOST⟨I,K⟩ ← +∞;;                              // initialization
 5              for each k ∈ K do
 6                  FS(⟨I, K − k, k⟩) ← COMPACT(I, K, k, FS(⟨I, K − k⟩));
 7                  if MINCOST⟨I,K⟩ > MINCOST⟨I,K−k,k⟩ then
 8                      FS(⟨I, K⟩) ← FS(⟨I, K − k, k⟩);
 9                  end
10              end
11          end
12      end
13      return FS(⟨I, J⟩)
14  end
15  Function COMPACT(I, K, k, FS(⟨I, K − k⟩))        // produce FS(⟨I,K−k,k⟩) from FS(⟨I,K−k⟩)
16      π⟨I,K−k,k⟩ ← (π⟨I,K−k⟩[1], . . . , π⟨I,K−k⟩[|I ⊔ K| − 1], k); ;    // initialization
17      MINCOST⟨I,K−k,k⟩ ← MINCOST⟨I,K−k⟩;;                      // initialization
18      NODE⟨I, K − k, k⟩ ← NODE⟨I, K − k⟩;;                    // initialization
19      for b ∈ {0, 1}^(n−|I|−|K|) do
20          u0 ← TABLE⟨I,K−k⟩[x[n]\(I⊔K) = b, xk = 0];
21          u1 ← TABLE⟨I,K−k⟩[x[n]\(I⊔K) = b, xk = 1];
22          if u0 = u1 then
23              TABLE⟨I,K−k,k⟩[x[n]\(I⊔K) = b] ← u0
24          else if ∃u (u, u0, u1) ∈ NODE⟨I, K − k, k⟩ then
25              TABLE⟨I,K−k,k⟩[x[n]\(I⊔K) = b] ← u
26          else                                                // create a new node
27              MINCOST⟨I,K−k,k⟩ ← MINCOST⟨I,K−k,k⟩ + 1;
28              TABLE⟨I,K−k,k⟩[x[n]\(I⊔K) = b] ← MINCOST⟨I,K−k,k⟩ + 1;
29              Insert (u, u0, u1) into NODE⟨I, K − k, k⟩
30          end
31      end
32      return FS(⟨I, K − k, k⟩)
33  end
```

---

### Adaptation to ZDD

To adapt FS* to ZDD, it suffices to modify lines 22 and 23 in algorithm FS* follows:

**if** $u_1 = 0$ **then**
  $\text{TABLE}_{\langle I, K-k, k \rangle}[x_{[n] \backslash (I \sqcup K)} = b] \leftarrow u_0$

**Algorithm OptOBDD**$(k, \alpha)$**:** Quantum OBDD-minimization algorithm with parameters $k \in \mathbb{N}$ and $\alpha := (\alpha_1, \ldots, \alpha_k) \in \mathbb{R}^k$ satisfying $0 < \alpha_1 < \cdots < \alpha_k < 1$, where the quantum minimum finding algorithm is used in line 9, and FS* is used in lines 3 and 16. "$A \leftarrow B$" means that $B$ is substituted for $A$.

**Input:** TABLE$_\emptyset$ (accessible from all Functions)
**Output:** $\mathcal{FS}([n])$

```
 1 Function Main()
 2 │   π∅ ← ∅, MINCOST∅ ← 0, NODE∅ ← ∅;                              // initialization
 3 │   compute the set {FS(K): K ⊆ [n], |K| = α₁n} by algorithm FS (or FS*);
 4 │   make the set of these FS(K) global (i.e., accessible from all Functions);
 5 │   return DivideAndConquer([n], k + 1)
 6 end
 7 Function DivideAndConquer(L, t)                  // Compute FS(L) with α₁,...,αₜ(= |L|/n)
 8 │   if t=1 then return FS(L);                              // FS(L) has been precomputed.
 9 │   Find K(⊂ L) of cardinality αₜ₋₁n, with Lemma 6, that minimizes
10 │        MINCOST⟨K,L\K⟩ ∈ FS(⟨K, L \ K⟩) ← ComputeFS(K, L\, t);
11 │   let K* be the set that achieves the minimum;
12 │   return FS(⟨K*, L \ K*⟩)
13 end
14 Function ComputeFS(K, M, t)                   // Compute FS(⟨K, M⟩) with α₁,...,αₜ(= |K ⊔ M|/n)
15 │   FS(K) ← DivideAndConquer(K, t − 1);
16 │   FS(⟨K, M⟩) ← FS*(K, M, FS(K));
17 │   return FS(⟨K, M⟩)
18 end
```

---

**Algorithm OptOBDD**$_\Gamma^*(k, \alpha)$**:** Composable Quantum OBDD-minimization algorithm with subroutine $\Gamma$ and parameters $k \in \mathbb{N}$ and $\alpha = (\alpha_1, \ldots, \alpha_k) \in \mathbb{R}^k$ satisfying $0 < \alpha_1 < \cdots < \alpha_k < 1$, where the quantum minimum finding algorithm is used in line 10, and subroutine $\Gamma$ is used in line 17. "$A \leftarrow B$" means that $B$ is substituted for $A$. $\Gamma(I_1, I_2, J, \mathcal{FS}(I_1, I_2))$ produces $\mathcal{FS}(I_1, I_2, J)$ from $\mathcal{FS}(I_1, I_2)$.

**Input:** TABLE$_\emptyset$, $I \subseteq [n], J \subseteq [n], \mathcal{FS}(\langle I, J \rangle)$. (accessible from all Functions)
**Output:** $\mathcal{FS}(\langle I, J \rangle)$

```
 1 Function Main()
 2 │   if I is empty then π∅ ← ∅, MINCOST∅ ← 0, NODE∅ ← ∅;             // initialization
 3 │   n′ ← |J|;;                                                     // initialization
 4 │   compute the set {FS(⟨I, K⟩): K ⊆ J, |K| = α₁n′} by algorithm FS*;
 5 │   make n′ and the above set of FS(⟨I, K⟩) global (i.e., accessible from all Functions);
 6 │   return DivideAndConquer(J, k + 1)
 7 end
 8 Function DivideAndConquer(L, t)                     // Compute FS(⟨I, L⟩) with α₁,...,αₜ
 9 │   if t=1 then return FS(I, L);                           // FS(I, L) has been precomputed.
10 │   Find K(⊂ L) of cardinality αₜ₋₁n′, with Lemma 6, that minimizes
11 │        MINCOST⟨I,K,L\K⟩ ∈ FS(⟨I, K, L \ K⟩) ← ComputeFS(K, L \ K, t);
12 │   let K* be the set that achieves the minimum;
13 │   return FS(⟨I, K*, L \ K*⟩)
14 end
15 Function ComputeFS(K, M, t)            // Compute FS(⟨K, M⟩) with α₁,...,αₜ(= |I ⊔ K ⊔ M|/n)
16 │   FS(I, K) ← DivideAndConquer(K, t − 1);
17 │   FS(⟨I, K, M⟩) ← Γ(I, K, M, FS(I, K));
18 │   return FS(⟨I, K, M⟩)
19 end
```

# References

[ABI+19]     Andris Ambainis, Kaspars Balodis, Janis Iraids, Martins Kokainis, Krisjanis Prusis, and Jevgenijs Vihrovs. Quantum speedups for exponential-time dynamic programming algorithms. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 1783–1793, 2019.

[Ake78]      S. B. Akers. Binary decision diagrams. *IEEE Trans. Comput.*, 27(6):509–516, June 1978.

[BC95]       Randal E. Bryant and Yirng-An Chen. Verification of arithmetic circuits with binary moment diagrams. In *Proceedings of the 32st Conference on Design Automation, San Francisco, California, USA, Moscone Center, June 12-16, 1995.*, pages 535–541, 1995.

[BCdWZ99]    Harry M. Buhrman, Richard E. Cleve, Ronald de Wolf, and Christof Zalka. Bounds for small-error and zero-error quantum algorithms. In *Proceedings of 40th Annual Symposium on Foundations of Computer Science, FOCS '99*, pages 358–368, 1999.

[BFG+97]     R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebric decision diagrams and their applications. *Formal Methods in System Design*, 10(2–3):171–206, 1997.

[Bol16]      Beate Bollig. On the minimization of (complete) ordered binary decision diagrams. *Theory of Computing Systems*, 59(3):532–559, Oct 2016.

[Bry86]      Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, August 1986.

[Bry91]      Randal E. Bryant. On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication. *IEEE Transactions on Computers*, 40(2):205–213, Feb 1991.

[Bry92]      Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3):293–318, September 1992.

[Bry18]      Randal E. Bryant. Binary decision diagrams. *Handbook of Model Checking*, pages 191–217, 2018.

[BW96]       B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45(9):993–1002, Sep. 1996.

[CMZ+97]     E.M. Clarke, K. L. Mcmillan, X. Zhao, M. Fujita, and J. Yang. Spectral transforms for large boolean functions with applications to technology mapping. *Formal Methods in System Design*, 10(2–3):137–148, 1997.

[DB98]       Rolf Drechsler and Bernd Becker. *Binary Decision Diagrams: Theory and Implementation*. Springer, 1998.

[DH96]       Christoph Dürr and Peter Høyer. A quantum algorithm for finding the minimum. Technical Report quant-ph/9607014, arXiv, 1996.

[DHHM06]     Christoph Dürr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla. Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6):1310–1328, 2006.

[FS90]       S. J. Friedman and K. J. Supowit. Finding the optimal variable ordering for binary decision diagrams. *IEEE Transactions on Computers*, 39(5):710–713, May 1990.

[GJ79]     Michael R. Garey and David S. Johnson. *COMPUTERS AND INTRACTABILITY — A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 2 edition, 1979.

[GLM08]    Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Phys. Rev. Lett.*, 100:160501, Apr 2008.

[Gro96]    Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 212–219, 1996.

[HC92]     Heh-Tyan Liaw and Chen-Shang Lin. On the obdd-representation of general boolean functions. *IEEE Transactions on Computers*, 41(6):661–664, June 1992.

[Hea93]    Mark Heap. On the exact ordered binary decision diagram size of totally symmetric functions. *Journal of Electronic Testing*, 4(2):191–195, 1993.

[HI02]     Takashi Horiyama and Toshihide Ibaraki. Ordered binary decision diagrams as knowledge-bases. *Artif. Intell.*, 136(2):189–213, 2002.

[HM94]     M. A. Heap and M. R. Mercer. Least upper bounds on obdd sizes. *IEEE Transactions on Computers*, 43(6):764–767, June 1994.

[HM00]     L. Heinrich-Litan and P. Molitor. Least upper bounds for the size of obdds using symmetry properties. *IEEE Transactions on Computers*, 49(4):360–368, April 2000.

[HTKY97]   K. Hosaka, Y. Takenaga, T. Kaneda, and S. Yajima. Size of ordered binary decision diagrams representing threshold functions. *Theoretical Computer Science*, 180(1):47 – 60, 1997.

[HY97]     Takashi Horiyama and Shuzo Yajima. Exponential lower bounds on the size of obdds representing integer division. In *Proceedings of the 8th International Symposium on Algorithms and Computation, (ISAAC '97), Singapore, December 17-19, 1997, Proceedings*, Lecture Notes in Computer Science, pages 163–172. Springer, 1997.

[INY98]    Kazuo Iwama, Mitsushi Nouzoe, and Shuzo Yajima. Optimizing obdds is still intractable for monotone functions. In *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS'98)*, volume 1450 of *Lecture Notes in Computer Science*, pages 625–635. Springer, 1998.

[KLM07]    Phillip Kaye, Raymond Laflamme, and Michele Mosca. *An Introduction to Quantum Computing*. Oxford University Press, 2007.

[Knu09]    Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams*. Addison-Wesley Professional, 1 edition, March 2009.

[KSV02]    Alexei Yu. Kitaev, Alexander H. Shen, and Mikhail N. Vyalyi. *Classical and Quantum Computation*, volume 47 of *Graduate Studies in Mathematics*. AMS, 2002.

[Lee59]    C. Y. Lee. Representation of switching circuits by binary-decision programs. *The Bell System Technical Journal*, 38(4):985–999, July 1959.

[LGM18]    François Le Gall and Frédéric Magniez. Sublinear-time quantum computation of the diameter in congest networks. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, pages 337–346, New York, NY, USA, 2018. ACM.

[Min93] Shin-ichi Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proceedins of the 30th ACM/IEEE Design Automation Conference*, pages 272–277, June 1993.

[Min11] Shin-ichi Minato. πDD: A new decision diagram for efficient problem solving in permutation space. In *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT 2011)*, volume 6695 of *Lecture Notes in Computer Science*, pages 90–104, 2011.

[MS94] Christoph Meinel and Anna Slobodová. On the complexity of constructing optimal ordered binary decision diagrams. In *Proceedings of 19th Mathematical Foundations of Computer Science 1994*, pages 515–524, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[MT98] Christoph Meinel and Thorsten Theobald. *Algorithms and Data Structures in VLSI Design: OBDD - Foundations and Applications*. Springer, 1998.

[NC00] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[Sie02a] Detlef Sieling. The complexity of minimizing and learning OBDDs and FBDDs. *Discrete Applied Mathematics*, 122(1):263 – 282, 2002.

[Sie02b] Detlef Sieling. The nonapproximability of OBDD minimization. *Information and Computation*, 172(2):103 – 138, 2002.

[SIT95] Kyoko Sekine, Hiroshi Imai, and Seiichiro Tani. Computing the Tutte polynomial of a graph of moderate size. In *Proceedings of the Sixth International Symposium on Algorithms and Computation (ISAAC '95)*, volume 1004 of *Lecture Notes in Computer Science*, pages 224–233. Springer, 1995.

[SM19] Kazuya Shimizu and Ryuhei Mori. Exponential-time quantum algorithms for graph coloring problems. *CoRR*, abs/1907.00529, 2019.

[STY94] Hiroshi Sawada, Yasuhiko Takenaga, and Shuzo Yajima. On the computational power of binary decision diagrams. *IEICE Trans. Info. & Syst., D*, 77(6):611–618, 1994.

[THY96] Seiichiro Tani, Kiyoharu Hamaguchi, and Shuzo Yajima. The complexity of the optimal variable ordering problems of a shared binary decision diagram. *IEICE Transactions on Information and Systems*, E79-D(4):271–281, 1996. (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).

[TI94] Seiichiro Tani and Hiroshi Imai. A reordering operation for an ordered binary decision diagram and an extended framework for combinatorics of graphs. In *Proceedings of the Fifth International Symposium on Algorithms and Computation (ISAAC'94)*, volume 834 of *Lecture Notes in Computer Science*, pages 575–583. Springer-Verlag, 1994.

[TY00] Yasuhiko Takenaga and Shuzo Yajima. Hardness of identifying the minimum ordered binary decision diagram. *Discrete Applied Mathematics*, 107(1-3):191–201, 2000.

[Weg00] Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, 2000.