Span Programs and Quantum Time Complexity

Arjan Cornelissen
QuSoft, Amsterdam, The Nehterlands
University of Amsterdam, The Netherlands

Stacey Jeffery
QuSoft, Amsterdam, The Netherlands
CWI, Amsterdam, The Netherlands

Maris Ozols
QuSoft, Amsterdam, The Netherlands
University of Amsterdam, The Netherlands

Alvaro Piedrafita
QuSoft, Amsterdam, The Netherlands
CWI, Amsterdam, The Netherlands

Abstract

Span programs are an important model of quantum computation due to their correspondence with quantum query and space complexity. While the query complexity of quantum algorithms obtained from span programs is well-understood, it is not generally clear how to implement certain query-independent operations in a time-efficient manner. In this work, we prove an analogous connection for quantum time complexity. In particular, we show how to convert a sufficiently-structured quantum algorithm for f with time complexity T into a span program for f such that it compiles back into a quantum algorithm for f with time complexity $\widetilde{\mathcal{O}}(T)$. This shows that for span programs derived from algorithms with a time-efficient implementation, we can preserve the time efficiency when implementing the span program, which means that span programs capture time, query and space complexities and are a complete model of quantum algorithms.

One practical advantage of being able to convert quantum algorithms to span programs in a way that preserves time complexity is that span programs compose very nicely. We demonstrate this by improving Ambainis's variable-time quantum search result using our construction through a span program composition for the OR function.

2012 ACM Subject Classification Theory of computation → Quantum query complexity; Theory of computation → Algorithm design techniques; Theory of computation → Quantum complexity theory

Keywords and phrases quantum query algorithms, span programs, variable-time quantum search

Digital Object Identifier 10.4230/LIPIcs.MFCS.2020.26

Related Version Full version available at arXiv:2005.01323.

Funding *Stacey Jeffery*: Supported by an NWO Veni Innovational Research Grant under project number 639.021.752, an NWO WISE Grant, and QuantERA project QuantAlgo 680-91-03. SJ is a CIFAR Fellow in the Quantum Information Science Program.
Maris Ozols: Supported by an NWO Vidi grant VI.Vidi.192.109.

Acknowledgements Stacey Jeffery thanks Tsuyoshi Ito for illuminating discussions on the topic of span programs and time complexity.

1 Introduction

Span programs are a model of computation first introduced in the context of classical complexity theory [15], and later introduced to the study of quantum algorithms by Reichardt and Špalek [18]. This connection to quantum algorithms proved to be particularly important when Reichardt showed that span programs are equivalent to dual solutions to the adversary bound, proving that the adversary bound is a tight lower bound on quantum query complexity, and span program complexity is a tight upper bound [17]. In particular, this means that for any decision problem, it is possible to design a query-optimal quantum algorithm using the span program framework, although finding such an algorithm is generally hard in practice. Later work connecting quantum space complexity to span programs enriched this connection further, showing that span program algorithms can also have optimal space complexity [13], although again, there is no prescriptive recipe for finding such algorithms.

While finding optimal span programs is difficult in general, span programs have been used to design quantum algorithms for a variety of problems, including *st*-connectivity [7], cycle detection and bipartiteness testing [3, 9], subgraph detection [6, 16], maximum bipartite matching [4], graph connectivity [12], k-distinctness [5], and formula evaluation [18, 17, 14], some of which have optimal query complexity. Given a span program P that decides a function f and has complexity $C(P)$, there is a generic transformation that compiles the span program into a quantum query algorithm with query complexity $\mathcal{O}(C(P))$.

The resulting algorithm does phase estimation to precision $C(P)^{-1}$ on a certain unitary U associated with the span program (for details, see Section 3). While it is clear that this unitary can be implemented using $\mathcal{O}(1)$ quantum queries to the input, it is not at all clear how to efficiently implement the query-independent parts of U. There has been some success in designing time-efficient span program algorithms for certain graph problems [7], but for other algorithms, perhaps most notably the span program algorithm for k-distinctness [5], no time-efficient implementation is known. This difficulty in analyzing the time complexity of span program algorithms represents a major drawback to the framework.

In this work, we make progress in understanding the relationship between span programs and quantum time complexity. We provide an explicit construction that turns any quantum algorithm that has sufficient structure (i.e., that allows for the construction of an efficient uniform access oracle) into a span program that compiles back to a quantum algorithm with essentially the same query, time and space complexity. This means that the span program framework is capable of producing time-efficient, well-structured quantum algorithms. The main result is stated in Theorem 9. We also exemplify how one might use our construction by improving Ambainis's variable time search result, in Theorem 13.

2 Preliminaries

2.1 Quantum query algorithms

Let $n \in \mathbb{N}$, $X \subseteq \{0,1\}^n$ and $f : X \to \{0,1\}$ be a (partial) Boolean function. We study quantum algorithms that compute (or decide) the value of $f(x)$ given quantum query access to individual bits of the input $x \in X$.

Let $\mathcal{A}$ be a quantum algorithm that acts on a state space $\mathbb{C}^{[n]\times\mathcal{W}}$, where $[n] := \{1,\dots,n\}$ and $\mathcal{W}$ is a finite set that labels the workspace states. We denote the initial state of $\mathcal{A}$ by $|\Psi_0\rangle \in \mathbb{C}^{[n]\times\mathcal{W}}$ and the unitary transformations that $\mathcal{A}$ applies at respective time steps by $U_1,\dots,U_T \in \mathcal{U}(\mathbb{C}^{[n]\times\mathcal{W}})$, where $T \in \mathbb{N}$ is the total number of time steps and the unitary tranformations are either oracle queries or efficiently implementable unitaries.

The algorithm $\mathcal{A}$ makes queries to an input string $x \in \{0,1\}^n$ by having a subset of the unitaries be (controlled) calls to an oracle $\mathcal{O}_x \in \mathcal{U}(\mathbb{C}^{[n]\times\mathcal{W}})$ defined as

$$\forall i \in [n], \forall j \in \mathcal{W}, \qquad \mathcal{O}_x : |i,j\rangle \mapsto (-1)^{x_i}|i,j\rangle, \tag{1}$$

where the two registers correspond to the input bit index and the workspace, respectively.

We denote by $\mathcal{Q} \subset [T]$ the set that contains all $t \in \mathcal{Q}$, such that $U_t = \mathcal{O}_x$. Then $Q = |\mathcal{Q}|$ is the query complexity of $\mathcal{A}$. We assume that U_1 and U_T are not queries.

For every $x \in \{0,1\}^n$ we define the state of the algorithm at time $t \in [T]_0 := \{0, \dots, T\}$ on input x as

$$|\Psi_t(x)\rangle := U_t U_{t-1} \cdots U_1 |\Psi_0\rangle, \tag{2}$$

where the initial state $|\Psi_0\rangle \in \mathbb{C}^{[n]\times\mathcal{W}}$ is a computational basis state. Note that the right-hand side of Equation (2) has an implicit dependence on x since $U_t = \mathcal{O}_x$ for some t.

We can assume that there is a single-qubit answer register used to indicate the output of the computation. If Π_b denotes the orthogonal projector onto states with $|b\rangle$ in the answer register, for $b \in \{0,1\}$, then $p_b(x) := \left\| \Pi_b |\Psi_T(x)\rangle \right\|^2$ is the probability that the algorithm outputs b on input x. We say that $\mathcal{A}$ computes a function $f : X \to \{0,1\}$ with error probability $\varepsilon \in [0, 1/2)$ if $p_{1-f(x)}(x) \leq \varepsilon$ for all $x \in X$.

We require all quantum query algorithms to have additional structure, which can be assumed without loss of generality (see [10]). We call such algorithms *clean*.

▶ **Definition 1** (Clean quantum algorithm). *Let $\mathcal{A}$ be a quantum query algorithm acting on $\mathbb{C}^{[n]\times\mathcal{W}} = \mathbb{C}^{[n]\times\mathcal{W}'\times\{0,1\}}$ with the last register being the answer register. Suppose that the time complexity of $\mathcal{A}$ is T, the query complexity is Q, and the initial state has $|0\rangle$ in the answer register, so it can be expressed as $|\Psi_0\rangle = |\psi_0\rangle|0\rangle$ for some $|\psi_0\rangle \in \mathbb{C}^{[n]\times\mathcal{W}'}$. Define the* final accepting state *as $|\Psi_T\rangle := |\psi_0\rangle|1\rangle$. $\mathcal{A}$ is a* clean quantum algorithm *if it satisfies the following properties.*

1. Consistency: *For all inputs $x \in \{0,1\}^n$, $\langle\Psi_T|\Psi_T(x)\rangle = p_1(x)$ and $\langle\Psi_T|(I \otimes X)|\Psi_T(x)\rangle = p_0(x)$, where $p_b(x) = \left\|(I \otimes |b\rangle\langle b|)|\Psi_T(x)\rangle\right\|^2$ is the probability that $\mathcal{A}$ outputs b on input x, and X denotes the Pauli matrix implementing the logical* NOT.
2. Commutation: *$(I \otimes X)$ commutes with every unitary U_t of the algorithm, where X acts on the answer register.*
3. Query-uniformity: *Two consecutive queries are not more than $\lfloor 3T/Q \rfloor$ time steps apart, and the first and last queries are separated by at most $\lfloor 3T/Q \rfloor$ time steps from the start and the end of the algorithm, respectively.*

2.2 Accessing an algorithm as input

Throughout the rest of the paper, we will consider algorithms that, among other things, take other algorithms as input. This section concerns how we model this through several oracles. The model is essentially a generalization of the one used in [2].

Let $m \in \mathbb{N}$ and let $\mathcal{A} = \{\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(m)}\}$ be a set of quantum query algorithms. For every $k \in [m]$, let $T^{(k)}$ be the time complexity of $\mathcal{A}^{(k)}$, let $\mathcal{Q}^{(k)} \subseteq [T^{(k)}]$ be the set of time steps at which $\mathcal{A}^{(k)}$ performs queries to the input, let $U_1^{(k)}, \dots, U_{T^{(k)}}^{(k)}$ be the sequence of unitaries in $\mathcal{A}^{(k)}$, and suppose that $\mathcal{A}^{(k)}$ evaluates a function $f_k : X^{(k)} \subseteq \{0,1\}^{n^{(k)}} \to \{0,1\}$ with bounded error. For convenience we define $T_{\max} = \max_{k\in[m]} T^{(k)}$ and $n_{\max} = \max_{k\in[m]} n^{(k)}$, and we assume that all unitaries $U_t^{(k)}$ act on some common space $\mathbb{C}^{[n_{\max}]\times\mathcal{W}}$, where the first register is large enough to hold the input bit label for any of the Boolean functions f_k.

We define three different oracles associated with $\mathcal{A}$. First, the *algorithm oracle*, sometimes referred to as "select" [8], acts on $\mathbb{C}^{[m]\times[T_{\max}]\times[n_{\max}]\times\mathcal{W}}$ as

$$\mathcal{O}_{\mathcal{A}} : |k\rangle|t\rangle|\psi\rangle \mapsto |k\rangle|t\rangle U_t^{(k)}|\psi\rangle, \qquad \forall k \in [m], t \in [T^{(k)}] \setminus \mathcal{Q}^{(k)}, |\psi\rangle \in \mathbb{C}^{[n_{\max}]\times\mathcal{W}}.$$

Second, the *query time step oracle*, which allows us to determine whether a given algorithm $\mathcal{A}^{(k)}$ performs a query at a given time step t, acts on $\mathbb{C}^{[m]\times[T_{\max}]}$ as

$$\forall k \in [m], t \in [T^{(k)}], \qquad \mathcal{O}_{\mathcal{Q}} : |k\rangle|t\rangle \mapsto \begin{cases} -|k\rangle|t\rangle, & \text{if } t \in \mathcal{Q}^{(k)}, \\ |k\rangle|t\rangle, & \text{otherwise.} \end{cases}$$

Finally, given a list of inputs $x = (x^{(1)}, \dots, x^{(m)})$, where $x^{(k)} \in \{0,1\}^{n^{(k)}}$ is the input to function f_k, the *input oracle* to x acts on $\mathbb{C}^{[m]\times[n_{\max}]}$ as

$$\forall k \in [m], \qquad \mathcal{O}_x = \sum_{k=1}^{m} |k\rangle\langle k| \otimes \mathcal{O}_{x^{(k)}}, \qquad \text{where} \qquad \forall i \in [n_k], \qquad \mathcal{O}_{x^{(k)}} : |i\rangle \mapsto (-1)^{x_i^{(k)}}|i\rangle.$$

On other states, the behavior of the three oracles can be arbitrary.

We say that we have *uniform access* to a set of algorithms $\mathcal{A}$ if we have access to the three oracles $\mathcal{O}_{\mathcal{A}}$, $\mathcal{O}_{\mathcal{Q}}$ and $\mathcal{O}_x$ defined above. Moreover, if the time complexity of implementing the oracles $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{Q}}$ is polylogarithmic in $T_{\max} = \max_{k\in[m]} T^{(k)}$ and m, then we say that we have *efficient uniform access* to $\mathcal{A}$.

These oracles fully capture the set $\mathcal{A}$ and provide an interface for the higher-level algorithms to execute the algorithms in $\mathcal{A}$ as subroutines. For very unstructured algorithms, the implementation cost of the oracles would be large and one would need to have the algorithms stored in QRAM or have access to them as *quantum random access stored-program machines* [19]. However, quantum query algorithms that we encounter in practice can usually be described very succinctly, meaning that efficient uniform access to the oracles $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{Q}}$ is possible. To illustrate this, observe that if we measure the implementation cost of these oracles in terms of the number of gates, then the implementation of the uniform access oracles $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{Q}}$ cost only $\mathcal{O}(\text{poly}(n))$ gates, even though it costs $\mathcal{O}(n\sqrt{2^n})$ gates to implement Grover's algorithm that evaluates the function $OR : \{0,1\}^{2^n} \to \{0,1\}$. See [10] for more details.

2.3 Span programs

Following [11], we define a span program for evaluating a Boolean function as follows.

▶ **Definition 2** (Span program). *A span program $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ on $\{0,1\}^n$ consists of*

1. *a finite-dimensional Hilbert space $\mathcal{H}$ that decomposes as $\mathcal{H} = \mathcal{H}_1 \oplus \cdots \oplus \mathcal{H}_n \oplus \mathcal{H}_{\text{true}} \oplus \mathcal{H}_{\text{false}}$ where each $\mathcal{H}_i$, $i \in [n]$, decomposes further as $\mathcal{H}_i = \mathcal{H}_{i,0} \oplus \mathcal{H}_{i,1}$,*
2. *a finite-dimensional Hilbert space $\mathcal{V}$,*
3. *a linear operator $A : \mathcal{H} \to \mathcal{V}$, and*
4. *a target vector $|\tau\rangle \in \mathcal{V}$.*

With each string $x \in \{0,1\}^n$, we associate the subspace $\mathcal{H}(x) = \mathcal{H}_{1,x_1} \oplus \cdots \oplus \mathcal{H}_{n,x_n} \oplus \mathcal{H}_{\text{true}}$. For any subspace $\mathcal{H}' \subseteq \mathcal{H}$, we write $\Pi_{\mathcal{H}'}$ to denote the projector onto $\mathcal{H}' \subseteq \mathcal{H}$.

Intuitively, a span program encodes the decision problem "Is $|\tau\rangle \in A\mathcal{H}(x)$?". To answer this question in the affirmative, it is sufficient to provide a preimage of $|\tau\rangle$ under A in $\mathcal{H}(x)$, called a *positive witness*. In the negative case, one would like to find an object, called a *negative witness*, that precludes the existence of such a positive witness.

▶ **Definition 3** (Positive and negative witnesses). *Fix a span program $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ and an input $x \in \{0,1\}^n$. We call a vector $|w\rangle \in \mathcal{H}$* a positive witness *for x if $|w\rangle \in \mathcal{H}(x)$, and $A|w\rangle = |\tau\rangle$. The* positive witness size *of x is*

$$w_+(x, P) = w_+(x) := \min_{|w\rangle \in \mathcal{H}(x)} \{\||w\rangle\|^2 : A|w\rangle = |\tau\rangle\},$$

if there exists a positive witness for x, and $w_+(x) = \infty$ otherwise. We say that $|\omega\rangle \in \mathcal{V}$ is a negative witness *for x if $\langle\omega|A\Pi_{\mathcal{H}(x)} = 0$ and $\langle\omega|\tau\rangle = 1$. The* negative witness size *of x is*

$$w_-(x, P) = w_-(x) := \min_{|\omega\rangle\in\mathcal{V}} \{\|\langle\omega|A\|^2 : \langle\omega|A\Pi_{\mathcal{H}(x)} = 0, \langle\omega|\tau\rangle = 1\},$$

if there exists a negative witness, and $w_-(x) = \infty$ otherwise. We define the set of positive *and* negative inputs *of P, respectively, as*

$$P_1 := \{x \in X : w_+(x) < \infty\}, \qquad P_0 := \{x \in X : w_-(x) < \infty\}.$$

One can think of a span program as a puzzle in which several pieces are supplied (the space $\mathcal{H}(x)$) together with assembly instructions (the map A) and a contour of a shape to be constructed (the target vector $|\tau\rangle$). The larger the number of pieces required to construct the target, the harder it is to solve the puzzle. Alternatively, the larger the number of missing pieces required to declare the problem unsolvable, the harder it is to do so. This justifies the notion that larger witness sizes are indicative of harder span programs.

▶ **Definition 4** (Span program complexity). *Let P be a span program, $f : X \subseteq \{0,1\}^n \to \{0,1\}$ a Boolean function, and $\lambda \in [0,1)$. The* positive *and* approximate negative complexity *of P (w.r.t. f) are, respectively,*

$$W_+(f, P) = W_+(P) := \max_{x\in f^{-1}(1)} w_+(x, P), \quad \widetilde{W}_-(f, P) = \widetilde{W}_-(P) := \max_{x\in f^{-1}(0)} \widetilde{w}_-(x, P),$$

where $\widetilde{w}_-(x, P)$ is the following minimization over all approximate negative witnesses *$|\widetilde{\omega}\rangle \in \mathcal{V}$:*

$$\widetilde{w}_-(x, P) := \min_{|\widetilde{\omega}\rangle\in\mathcal{V}} \left\{\|\langle\widetilde{\omega}|A\|^2 : \langle\widetilde{\omega}|\tau\rangle = 1, \left\|\langle\widetilde{\omega}|A\Pi_{\mathcal{H}(x)}\right\|^2 \leq \lambda/W_+(f, P)\right\}.$$

We say that P positively λ-approximates *f if $f^{-1}(1) \subseteq P_1$ and $\widetilde{w}_-(x, P) < \infty$ for all $x \in f^{-1}(0)$. The* complexity *of P with respect to f is*

$$C(P) := \sqrt{W_+(f, P)\widetilde{W}_-(f, P)},$$

We say that P decides *f* exactly *if it 0-approximates f.*

3 The time complexity of implementing a span program

Span programs by themselves are not quantum objects. Nevertheless, the elements that define a span program can be combined to form a quantum algorithm. In this section, we describe such an algorithm and consider its implementation and time complexity in a general setting. We express its time complexity in terms of the time complexities of several subroutines derived from the span program. We do not show how to implement these in general – such details will depend on the specific construction of the span program.

3.1 The span program algorithm

Let P be a span program that positively λ-approximates f. We call $|w_0\rangle = A^+|\tau\rangle$ the *minimal positive witness* of P, where A^+ is the Moore-Penrose pseudoinverse of A. Suppose that P is normalized, i.e., $\||w_0\rangle\| = 1$, and that $W_+(P) \leq 2$. The *span program unitary* of P on input $x \in \{0,1\}^n$ is

$$U = U(P, x) = (2\Pi_{\mathcal{H}(x)} - I)\Big(2\big(\Pi_{\ker A} + |w_0\rangle\langle w_0|\big) - I\Big). \tag{3}$$

The span program algorithm of [11] works by doing phase estimation of U on the initial state $|w_0\rangle$ to precision Θ and error probability at most ε, and then estimating the amplitude of this process on a 0 in the phase register using precision Θ' where

$$\Theta = \Omega\left(\sqrt{\frac{1-\lambda}{\widetilde{W}_-(P)}}\right), \qquad \varepsilon = \Omega\left(1-\lambda\right), \qquad \text{and} \qquad \Theta' = \Omega\left(1-\lambda\right).$$

This algorithm requires constructing the state $|w_0\rangle$, and then making $\mathcal{O}(\frac{1}{\Theta'}\frac{1}{\Theta}\log\frac{1}{\varepsilon})$ controlled calls to U. Note that

$$\frac{1}{\Theta'}\frac{1}{\Theta}\log\frac{1}{\varepsilon} = \mathcal{O}\left(\frac{\sqrt{\widetilde{W}_-(P)}}{(1-\lambda)^{3/2}}\log\frac{1}{1-\lambda}\right),$$

where the big-$\mathcal{O}$-notation corresponds to $\lambda \uparrow 1$ and $\widetilde{W}_-(P) \to \infty$. For a more detailed description of this algorithm, see [11].

In [10], we show that if P is not normalized or does not satisfy $W_+(P) \leq 2$, then we can turn it into a normalized span program P' that 2λ-approximates f with $W_+(P') \leq 2$ and $\widetilde{W}_-(P') \leq W_+(P)W_-(P)$. The resulting number of calls to the span program unitary $U(P', x)$ is then

$$\mathcal{O}\left(\frac{\sqrt{W_+(P)\widetilde{W}_-(P)}}{(1-2\lambda)^{3/2}}\log\frac{1}{1-2\lambda}\right), \tag{4}$$

where the big-$\mathcal{O}$-notation corresponds to $\lambda \uparrow 1/2$ and $W_+(P), \widetilde{W}_-(P) \to \infty$. We refer to this algorithm as the *algorithm compiled from* P, or the *span program algorithm for* P.

3.2 Time complexity of the span program algorithm

We now turn our attention to the time complexity of the span program algorithm for P. We express it in Theorem 6 in terms of the number of calls to certain black-box operations defined directly from the span program P. In principle, the time complexity of any span program algorithm can be analyzed using this theorem, and we expect that its relevance is not restricted to the application we present in subsequent sections.

In our analysis of span program time complexity we introduce a central concept of an *implementing subspace.* This subspace depends on the particular input $x \in \{0,1\}^n$, and it is often much smaller than the ambient Hilbert space $\mathcal{H}$. Throughout the execution of the span program algorithm the state vector remains in this subspace, so all operations in the span program algorithm need only be defined in this subspace to ensure successful computation of the span program.

▶ **Definition 5** (Implementing subspace). *Let $\lambda \in [0,1)$ and let $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ be a span program that positively λ-approximates a Boolean function $f : X \subseteq \{0,1\}^n \to \{0,1\}$. Let $x \in X$ and let $\mathcal{H}_x$ be a subspace of $\mathcal{H}$ such that:*

1. $\Pi_{\ker(A)}\mathcal{H}_x \subseteq \mathcal{H}_x$.
2. $\Pi_{\mathcal{H}(x)}\mathcal{H}_x \subseteq \mathcal{H}_x$.
3. $|0\rangle \in \mathcal{H}_x$*, where $|0\rangle$ is the all-zeros computational basis state.*
4. $|w_0\rangle \in \mathcal{H}_x$*, where $|w_0\rangle = A^+|\tau\rangle$ is the minimal witness for P.*

Then we refer to $\mathcal{H}_x$ as an implementing subspace of P for x.

For example, for any $x \in X$, a valid implementing subspace $\mathcal{H}_x$ of P for input x is $\mathcal{H}$ itself. Now, we can state the main result of this section.

▶ **Theorem 6.** *Fix $\lambda \in [0, 1/2)$. Suppose $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ is a span program that positively λ-approximates a function $f : X \subseteq \{0,1\}^n \to \{0,1\}$. For all $x \in X$, let $\mathcal{H}_x$ be an implementing subspace for P. Suppose that we have access to the following subroutines and their controlled versions:*

1. *A subroutine $\mathcal{R}_{\ker(A)}$ that acts on $\mathcal{H}_x$ as $2\Pi_{\ker(A)} - I$.*
2. *A subroutine $\mathcal{C}_{|w_0\rangle}$ that leaves $\mathcal{H}_x$ invariant and maps $|0\rangle$ to $|w_0\rangle / \||w_0\rangle\|$.*
3. *A subroutine $\mathcal{R}_{\mathcal{H}(x)}$ that acts on $\mathcal{H}_x$ as $2\Pi_{\mathcal{H}(x)} - I$.*
4. *A subroutine $\mathcal{R}_{|0\rangle}$ that acts on $\mathcal{H}_x$ as $2|0\rangle\langle 0| - I$.*

Then we can implement the span program algorithm for P using a number of calls to the previous subroutines that satisfies

$$\mathcal{O}\left(\frac{\sqrt{W_+(P)\widetilde{W}_-(P)}}{(1-2\lambda)^{3/2}} \log \frac{1}{1-2\lambda}\right),$$

where the big-$\mathcal{O}$-notation refers to the limit where $\lambda \uparrow 1/2$ and $W_+(P), \widetilde{W}_-(P) \to \infty$. Moreover, the number of extra gates and auxiliary qubits used is $\mathcal{O}(\mathrm{polylog}(C(P), 1/(1-2\lambda)))$.[1] *Finally, it suffices to merely use upper bounds on $W_+(P)$, $W_-(P)$ and λ, if one substitutes these upper bounds in the relevant complexities.*

The benefit implementing subspaces becomes clear from Theorem 6. If we take $\mathcal{H}_x = \mathcal{H}$, then implementing $\mathcal{R}_{|0\rangle}$ takes $\Theta(\log \dim \mathcal{H})$ gates, as we must simply check that every qubit is in the state $|0\rangle$. However, for algorithms with large space complexity, such as the element distinctness algorithm [1], this is very costly, especially if we have to do it many times. In some cases, as in our main result Theorem 9 in Section 4, we can show that the span program has an implementing subspace with much smaller dimension, thus circumventing an undesired $\log(\dim \mathcal{H})$ overhead in the time complexity of the span program algorithm.

4 From algorithms to span programs

Let $\mathcal{A}$ be a clean quantum algorithm that evaluates a function $f : X \subseteq \{0,1\}^n \to \{0,1\}$ with error probability $0 \leq \varepsilon < 1/2$, as in Definition 1. Based on this algorithm, one can construct a span program that approximates the same function and whose complexity is equal to the query complexity of $\mathcal{A}$, up to a multiplicative constant. This construction was first introduced by Reichardt [17] in the case where the algorithm has one-sided error, and extended to the case of bounded (two-sided) error in [13].

Our contribution is to extend this construction so that it not only preserves the query complexity of $\mathcal{A}$ but also the time complexity. Starting with a quantum algorithm $\mathcal{A}$ whose query complexity is Q and time complexity is T, we construct a corresponding span program $P_{\mathcal{A}}$. If the span program is compiled back to a quantum algorithm, the resulting algorithm still solves the same problem, its query complexity is $\widetilde{\mathcal{O}}(Q)$ and its time complexity remains $\widetilde{\mathcal{O}}(T)$. This requires modifications to the span program construction, but more importantly, an additional highly non-trivial analysis of the time complexity of the span program algorithm.

[1] By $f(x,y) = \mathcal{O}(\mathrm{polylog}(x,y))$, we mean that there exist constants $C_1, C_2 > 0$ such that $f(x,y) = \mathcal{O}(\log^{C_1}(x) \log^{C_2}(y))$, in the limit where $x, y \to \infty$.

4.1 The span program of an algorithm

Recall from Section 2.2 that we can assume without loss of generality that there are no two consecutive queries in the algorithm $\mathcal{A}$, and that the first and last unitaries are not queries. We label the time steps where the algorithm queries the inputs by

$$\mathcal{Q} = \{q_1, \dots, q_Q\} \subseteq [T], \tag{5}$$

where T is the total time complexity and Q is the total number of queries. For convenience, we also define $q_0 = 0$, $q_{Q+1} = T + 1$. We denote the ℓ-th block of contiguous non-query time steps by $\mathcal{B}_\ell \subseteq [T]$, with $\ell \in [Q + 1]$. See Fig. 1 for an overview of this notation.

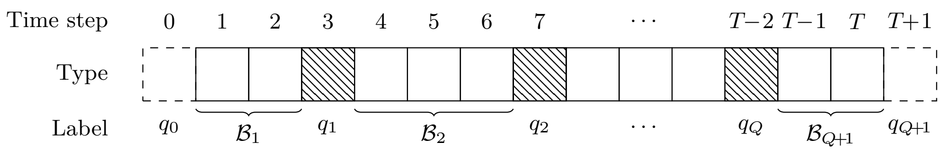

Figure 1 Synopsis of our notation. The cells indexed from 1 to T denote time steps of the algorithm $\mathcal{A}$ where time progresses to the right. The hatched cells denote time steps in which a query to the input x is performed. In all other time steps t a unitary U_t independent of x is applied.

Recall that $\mathcal{W}$ is a finite set that labels the basis of the workspace of $\mathcal{A}$. Let $[T]_0 := \{0, \dots, T\}$. We define the following objects:

$$\begin{aligned} \forall i \in [n], b \in \{0,1\}, \qquad \mathcal{H}_{i,b} &= \mathrm{span}\{|t, b, i, j\rangle : t + 1 \in \mathcal{Q}, j \in \mathcal{W}\}, \\ \mathcal{H}_{\mathrm{false}} = \{0\}, \qquad \mathcal{H}_{\mathrm{true}} &= \mathrm{span}\{|t, 0, i, j\rangle : t + 1 \in [T + 1] \setminus \mathcal{Q}, i \in [n], j \in \mathcal{W}\}, \\ |\tau\rangle = |0\rangle|\Psi_0\rangle - |T\rangle|\Psi_T\rangle, \qquad \mathcal{V} &= \mathrm{span}\{|t, i, j\rangle : t \in [T]_0, i \in [n], j \in \mathcal{W}\}. \end{aligned} \tag{6}$$

where $|\Psi_0\rangle$ is the initial state of $\mathcal{A}$ (see Equation (2)) and $|\Psi_T\rangle$ is the final accepting state (see Definition 1). As usual, the spaces $\mathcal{H}(x)$ and $\mathcal{H}$ are defined from these as:

$$\forall x \in \{0,1\}^n, \quad \mathcal{H}(x) = \left(\bigoplus_{i=1}^{n} \mathcal{H}_{i,x_i}\right) \oplus \mathcal{H}_{\mathrm{true}}\,; \quad \mathcal{H} = \left(\bigoplus_{i \in [n], b \in \{0,1\}} \mathcal{H}_{i,b}\right) \oplus \mathcal{H}_{\mathrm{true}} \oplus \mathcal{H}_{\mathrm{false}}. \tag{7}$$

For better intuition, we provide a graphical depiction of $\mathcal{H}$, $\mathcal{H}_{\mathrm{true}}$, $\mathcal{H}(x)$ and $\mathcal{H}_{i,b}$ in Fig. 2.

Recall that Q denotes the total number of queries and ε is the error probability of $\mathcal{A}$. Let

$$a = \sqrt{\frac{\varepsilon}{2Q + 1}} \qquad \text{and} \qquad M = \max_{\ell \in [Q+1]} \sqrt{|\mathcal{B}_\ell|}, \tag{8}$$

where $\mathcal{B}_\ell \subseteq [T]$ is the ℓ-th contiguous block of non-query gates (see Fig. 1). Since any quantum algorithm can be made clean as in Definition 1, without loss of generality we can assume that $M \leq \sqrt{3T/Q}$. For all computational basis vectors $|t, b, i, j\rangle$ in $\mathcal{H}$, we define the action of the span program operator $A : \mathcal{H} \to \mathcal{V}$ as follows:

$$A|t, b, i, j\rangle = \begin{cases} a|T, i, j\rangle & \text{if } t = T, \\ M(|t, i, j\rangle - |t + 1\rangle U_{t+1}|i, j\rangle) & \text{if } \exists \ell \in [Q + 1] : t + 1 \in \mathcal{B}_\ell, \\ |t, i, j\rangle - (-1)^b |t + 1, i, j\rangle & \text{if } \exists \ell \in [Q] : t + 1 = q_\ell. \end{cases} \tag{9}$$

The unitary U_{t+1} is the $(t + 1)$-th unitary of algorithm $\mathcal{A}$ as defined in Section 2.1. The weights a and M are the main difference between our construction and that of [13], and will enable the time-efficient implementation of the span program.

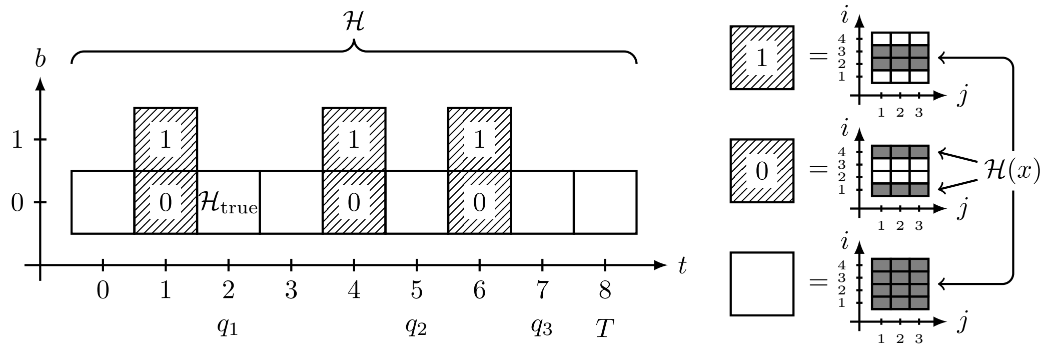

Figure 2 Graphical depiction of the relevant spaces when $T = 8$, $\mathcal{Q} = \{2, 5, 7\}$, $n = 4$, $|\mathcal{W}| = 3$ and $x = 0110$. The total space $\mathcal{H}$ is a direct sum of all blocks on the left, where the block at position $(t, b) \in [T]_0 \times \{0, 1\}$ denotes the subspace spanned by all computational basis states of the form $|t, b, \cdot, \cdot\rangle$. Every block is of one of three types, white, 0 or 1, shown on the right. The subspace $\mathcal{H}_{\text{true}}$ is the direct sum of all white blocks. Each block further decomposes as a direct sum over computational basis states $|i, j\rangle$, $i \in [n]$, $j \in \mathcal{W}$. The gray cells of all blocks together span the space $\mathcal{H}(x)$. Finally, for a given $i \in [n]$, the subspaces $\mathcal{H}_{i,0}$ and $\mathcal{H}_{i,1}$ consist of the i-th row within all 0 and 1 blocks, respectively.

▶ **Definition 7** (Span program of an algorithm). *The* span program of a quantum algorithm *$\mathcal{A}$ is $P_{\mathcal{A}} = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$, where $\mathcal{H}$, $\mathcal{V}$, $|\tau\rangle$, and A are defined in Equations* (6), (7) *and* (9).

This newly-defined span program evaluates the same function as the algorithm $\mathcal{A}$, as shown by the following lemma.

▶ **Lemma 8.** *Let $\mathcal{A}$ be a clean quantum algorithm for f with error probability $0 \leq \varepsilon < 1/5$, making Q queries, and let $P_{\mathcal{A}}$ the span program for $\mathcal{A}$ from Definition 7. Then $P_{\mathcal{A}}$ positively 5ε-approximates f with complexities $W_+(P_{\mathcal{A}}) \leq 3(2Q + 1) = \mathcal{O}(Q)$ and $\widetilde{W}_-(P_{\mathcal{A}}) = \mathcal{O}(Q)$.*

We prove this lemma by finding explicit positive and negative witnesses for different inputs $x \in X$ (see [10]).

4.2 Time complexity of algorithms compiled from span programs of algorithms

We can now analyze the time complexity of the algorithm compiled from $P_{\mathcal{A}}$. Our main result is summarized below.

▶ **Theorem 9.** *Let $\mathcal{A}$ be a clean quantum query algorithm that acts on S qubits, has query complexity Q, time complexity T, and evaluates a function $f : X \subseteq \{0, 1\}^n \to \{0, 1\}$ with bounded error. Let $P_{\mathcal{A}}$ be the span program for this algorithm, as in Definition 7. Then we can implement the algorithm compiled from $P_{\mathcal{A}}$ with:*
1. *$\mathcal{O}(Q \log(Q))$ calls to $\mathcal{O}_x$.*
2. *$\mathcal{O}(T \log(Q))$ calls to $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{Q}}$, as defined in Section 2.2.*
3. *$\mathcal{O}(T\text{polylog}(T))$ additional gates.*
4. *$\mathcal{O}(\text{polylog}(T) + S^{o(1)})$ auxiliary qubits.*

If we additionally require that the error probability of $\mathcal{A}$ is $o(1/Q^2)$, then the $\log(Q)$ factors and the $S^{o(1)}$ term can be removed. We can also drop $S^{o(1)}$ if we assume that $T = S^{1+\Omega(1)}$.

The proof leans on the structure of Theorem 6. This theorem says that every quantum query algorithm can be converted into a span program and back into a quantum algorithm while keeping the query and time complexity unaffected up to polylogarithmic factors. This implies that span programs fully capture both query, time and space complexity up to polylogarithmic factors, which establishes an equivalence between quantum algorithms and span programs, and strengthens the motivation for considering the latter as an important formalism from which to derive quantum algorithms.

We define a suitable implementing subspace that allows for efficient implementations of the four subroutines that are required.

▶ **Definition 10** (Implemeting subspace for $\mathcal{A}$)**.** *Let $x \in X$ and for all $t \in [T-1]_0$, let $|\widetilde{\Psi}_t(x)\rangle = U_{t+1}^\dagger \cdots U_T^\dagger |\Psi_T\rangle$. Furthermore, let $|\widetilde{\Psi}_T(x)\rangle = |\Psi_T\rangle$. We define two subspaces of $\mathcal{H}$ as follows, where we let t range over $\{0,\dots,T\}$:*

$$\overline{\mathcal{H}}_x = \mathrm{span}\{|t\rangle|0\rangle|\Psi_t(x)\rangle : t+1 \notin \mathcal{Q}\} \oplus \mathrm{span}\{|t\rangle|+\rangle|\Psi_t(x)\rangle, |t\rangle|-\rangle|\Psi_{t+1}(x)\rangle : t+1 \in \mathcal{Q}\},$$
$$\widetilde{\mathcal{H}}_x = \mathrm{span}\{|t\rangle|0\rangle|\widetilde{\Psi}_t(x)\rangle : t+1 \notin \mathcal{Q}\} \oplus \mathrm{span}\{|t\rangle|+\rangle|\widetilde{\Psi}_t(x)\rangle, |t\rangle|-\rangle|\widetilde{\Psi}_{t+1}(x)\rangle : t+1 \in \mathcal{Q}\}.$$

Next, we let $\mathcal{H}_x = \overline{\mathcal{H}}_x$ if $f(x) = 1$ and $\mathcal{H}_x = \overline{\mathcal{H}}_x + \widetilde{\mathcal{H}}_x$ if $f(x) = 0$.

The intuition behind our use of implementing subspaces is that, for a given input $x \in X$, the state of the algorithm moves through the Hilbert space in a simple, one-dimensional path. So, given a time step $t \in [T]_0$ and an input $x \in X$, we can deduce what the corresponding state in algorithm $\mathcal{A}$ must be at that time step, and hence we can deduce the state in the last register of $\mathcal{H}$. The only difficulty arises at the query time steps, where we use the state before the query when the first two registers are in state $|t\rangle|+\rangle$, and the state after the query when the first two registers are in state $|t\rangle|-\rangle$.

Note that $\overline{\mathcal{H}}_x$ and $\widetilde{\mathcal{H}}_x$ are very close to one another when x is a positive instance. This is because $|\Psi_T(x)\rangle$ is very close to $|\Psi_T\rangle$ when $f(x) = 1$. Hence, in this case it makes sense to only take one of the spaces as the implementing subspace. On the other hand, if $f(x) = 0$, the two spaces $\overline{\mathcal{H}}_x$ and $\widetilde{\mathcal{H}}_x$ are almost orthogonal, and hence we take both.

The main benefit of this implementing subspace is that we reduce the cost of reflecting around the state $|0\rangle|0\rangle|\Psi_0\rangle$ significantly. Such a reflection is in principle very simple because we can assume that $|\Psi_0\rangle = |0\rangle^{\otimes \log(n)+\log(|\mathcal{W}|)}$. However, in general, the time complexity of this reflection is $\Theta(\log(T) + \log(n) + \log(|\mathcal{W}|))$, since it is necessary to check that every qubit is in the $|0\rangle$ state.[2] As $|\mathcal{W}|$ can be exponentially large in T, this incurs significant overhead on the resulting time complexity of the algorithm. Using our implementing subspace, we get the number of gates down to $\Theta(\log(T))$.

5 Application to variable time search

In the previous section, we described a construction that turns a quantum algorithm into a span program. One advantage of the latter is that span programs compose very nicely (see [17] for a number of examples). We illustrate this by describing a construction that, given span programs for n functions $\{f_k : \{0,1\}^{m_k} \to \{0,1\}\}_{k=1}^n$, outputs a span program for the logical OR of their output: $f(x^{(1)},\dots,x^{(n)}) = \bigvee_{k=1}^n f_k(x^{(k)})$. In short, we show that given query-, time- and space-efficient quantum implementations for each f_k, the resulting algorithm compiled from the OR span program can also be implemented query-, time- and

[2] This detail has been neglected in previous work. For example, efficient implementation of such a reflection is not discussed in [2]. While this is not inconsistent with the stated results, since the result only claims to count oracle calls to $\mathcal{O}_x$ and $\mathcal{O}_{\mathcal{A}}$, this is not true of subsequent work that uses the results of [2]. We suspect that an argument like ours could also be made in previous work, but feel it is sufficiently non-trivial that it should not be taken for granted.

space-efficiently. Throughout this section we write f_k as functions on $\{0,1\}^{m_k}$ for the sake of simplicity, even though the results also hold for partial Boolean functions with arbitrary domains $X_k \subseteq \{0,1\}^{m_k}$.

5.1 The OR of span programs

Fix $\lambda \in (0,1/n)$. For $k \in [n]$, let $P^{(k)} = (\mathcal{H}^{(k)}, \mathcal{V}^{(k)}, A^{(k)}, |\tau^{(k)}\rangle)$ be a span program on $\{0,1\}^{m_k}$ that positively λ-approximates $f_k : \{0,1\}^{m_k} \to \{0,1\}$.[3] Let $W_+^{(k)}$ and $W_-^{(k)}$ be some upper bounds on $W_+(P^{(k)})$ and $\widetilde{W}_-(P^{(k)})$ respectively, and assume that every $x \in f_k^{-1}(0)$ has an approximate negative witness $|\tilde{\omega}^{(k)}\rangle \in \mathcal{V}^{(k)}$ with $\left\|\langle\tilde{\omega}^{(k)}|A^{(k)}\Pi_{\mathcal{H}^{(k)}(x)}\right\|^2 \leq \lambda/W_+^{(k)}$ and $\left\|\langle\tilde{\omega}^{(k)}|A^{(k)}\right\|^2 \leq W_-^{(k)}$. Let $C_k = \sqrt{W_+^{(k)}W_-^{(k)}}$.

Assume, by applying an appropriate basis change, that $|\tau^{(k)}\rangle = |0\rangle$ for every $k \in [n]$. For each k, extend $|\tau^{(k)}\rangle = |0\rangle$ to an orthonormal basis $\{|0\rangle, |k,1\rangle, \dots, |k, \dim(\mathcal{V}^{(k)})-1\rangle\}$ for $\mathcal{V}^{(k)}$ so that, aside from the single overlapping dimension $|0\rangle$, the subspaces $\mathcal{V}^{(k)}$ are orthogonal to one another. Let $\overline{\mathcal{V}}^{(k)} = \mathrm{span}\{|k,1\rangle, \dots, |k, \dim(\mathcal{V}^{(k)})-1\rangle\}$, so that $\mathcal{V}^{(k)} = \mathrm{span}\{|0\rangle\} \oplus \overline{\mathcal{V}}^{(k)}$.

Let $f : \{0,1\}^{m_1+\dots+m_n} \to \{0,1\}$ be defined by $f(x^{(1)}, \dots, x^{(n)}) = \bigvee_{k=1}^n f_k(x^{(k)})$. We can define a span program P on $\{0,1\}^{m_1+\dots+m_n}$ that decides f as follows:

$$\forall k \in [n], \ell \in [m_k], b \in \{0,1\}, \quad \mathcal{H}_{k,\ell,b} = \mathrm{span}\{|k\rangle\} \otimes \mathcal{H}_{\ell,b}^{(k)}, \quad \mathcal{H}_{\mathrm{true}} = \bigoplus_{k=1}^n \mathcal{H}_{\mathrm{true}}^{(k)},$$

$$\mathcal{H}_{\mathrm{false}} = \mathrm{span}\{|0,0\rangle\}, \quad \mathcal{V} = \mathrm{span}\{|0\rangle\} \oplus \bigoplus_{k=1}^n \overline{\mathcal{V}}^{(k)}, \quad A = \sum_{k=1}^n \sqrt{W_+^{(k)}}\langle k| \otimes A^{(k)}. \tag{10}$$

Above, we are indexing into an input $x \in \{0,1\}^{m_1+\dots+m_n}$ by using a pair of indices, $k \in [n]$ and $\ell \in [m_k]$, in the obvious way. From this definition of P, we get:

$$\mathcal{H}(x) = \bigoplus_{k\in[n]} \mathrm{span}\{|k\rangle\} \otimes \mathcal{H}^{(k)}(x^{(k)}), \quad \text{where} \quad \forall k \in [n], \mathcal{H}^{(k)}(x^{(k)}) = \bigoplus_{\ell\in[m_k]} \mathcal{H}_{\ell, x_\ell^{(k)}}^{(k)}. \tag{11}$$

▶ **Definition 11.** *Let $\{P^{(k)}\}_{k=1}^n$ be a set of span programs, where $P^{(k)} = (\mathcal{H}^{(k)}, \mathcal{V}^{(k)}, A^{(k)}, |\tau^{(k)}\rangle)$. Then we let $P = (\mathcal{H}, \mathcal{V}, A, |0\rangle)$ be the* OR span program *of these span programs, where $\mathcal{H}$, $\mathcal{V}$ and A are defined in Equation* (10)*.*

We prove that the OR span program P indeed evaluates f (see [10] for proof).

▶ **Lemma 12.** *P positively $n\lambda$-approximates f with complexity $C(P) \leq \sqrt{\sum_{k=1}^n C_k^2}$.*

5.2 Implementation of the OR span program

We are now in position to state the main theorem of this section.

▶ **Theorem 13** (Variable-time quantum search)**.** *Let $\mathcal{A} = \{\mathcal{A}^{(k)}\}_{k=1}^n$ be a finite set of quantum algorithms, where $\mathcal{A}^{(k)}$ acts on $S_k \leq S_{\max}$ qubits and decides $f_k : \{0,1\}^{m_k} \to \{0,1\}$ with bounded error with query complexity Q_k and time complexity $T_k \leq T_{\max}$. Suppose that we have uniform access to the algorithms in $\mathcal{A}$ through the oracles $\mathcal{O}_{\mathcal{A}}$, $\mathcal{O}_{\mathcal{Q}}$ and $\mathcal{O}_x$, as elaborated upon in Section 2.2. Then we can implement a quantum algorithm that decides $f = \bigvee_{k=1}^n f_k$ with bounded error, with the following properties:*

[3] We require λ to be quite small here. One way to achieve this from an arbitrary span program is to convert it to an algorithm, reduce the error to $\mathcal{O}(1/n)$ at the expense of a $\mathcal{O}(\log n)$ multiplicative factor, and then convert that back to a span program using the construction in Section 4.1. Furthermore, we can just as well use partial functions here, but we don't for notational simplicity.

1. *The number of calls to* $\mathcal{O}_x$ *is* $\mathcal{O}\left(\sqrt{\sum_{k=1}^n Q_k^2} \cdot \log\left(\sum_{k=1}^n Q_k^2\right)\right)$.
2. *The number of calls to* $\mathcal{O}_{\mathcal{A}}$ *and* $\mathcal{O}_{\mathcal{Q}}$ *is* $\mathcal{O}\left(\sqrt{\sum_{k=1}^n T_k^2} \cdot \log\left(\sum_{k=1}^n Q_k^2\right)\right)$.
3. *The number of extra gates is* $\mathcal{O}\left(\sqrt{\sum_{k=1}^n T_k^2} \cdot \operatorname{polylog}(T_{\max}, n)\right)$.
4. *The number of auxiliary qubits is* $\mathcal{O}\left(\operatorname{polylog}(T_{\max}, n) + S_{\max}^{o(1)}\right)$.

If we additionally require that the error probabilities of the $\mathcal{A}^{(k)}$*'s are all* $o(1/\sum_{k=1}^n Q_k^2)$*, then the* $\log(\sum_{k=1}^n Q_k^2)$ *factors and the* $S_{\max}^{o(1)}$ *term can be dropped. We can also drop the term* $S_{\max}^{o(1)}$ *if* $T_k = S_k^{1+\Omega(1)}$ *for all* $k \in [n]$.

The proof of this theorem follows from converting the algorithms $\mathcal{A}^{(k)}$ into span programs using the construction in Section 4, applying Theorem 6 to their OR span program defined in Definition 11, and subsequently providing sufficiently efficient implementations of $\mathcal{R}_{\ker \mathcal{A}}$, $\mathcal{C}_{|w_0\rangle}$, $\mathcal{R}_{\mathcal{H}(x)}$ and $\mathcal{R}_{|0\rangle}$. Further details can be found in [10].

A similar result was reached by Ambainis in [2]. Let us discuss how our result compares to that of Ambainis.

The uniform access model described in Section 2.2 is a slight generalization of the model considered by Ambainis, as explained in [2, Appendix A]. Unlike Ambainis, we differentiate between query and non-query time steps in the algorithms $\mathcal{A}^{(k)}$. Therefore, Ambainis only considers the algorithm oracle $\mathcal{O}_{\mathcal{A}}$ and includes the queries to x as part of this oracle, whereas we also assume to have explicit access to the oracles $\mathcal{O}_{\mathcal{Q}}$ and $\mathcal{O}_x$.

One can obtain the same number of calls to $\mathcal{O}_x$ or to $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{Q}}$ separately using Ambainis's construction. If one counts every query to x in the original algorithms as having unit cost, then Ambainis's construction yields an algorithm that evaluates $f = \bigvee_{k=1}^n f_k$ with $\mathcal{O}(\sqrt{\sum_{k=1}^n Q_k^2})$ queries to $\mathcal{O}_x$. Similarly, if one assigns a unit cost to every gate in the original algorithms, then the algorithm that follows from Ambainis's construction performs $\mathcal{O}(\sqrt{\sum_{k=1}^n T_k^2})$ calls to $\mathcal{O}_{\mathcal{A}}$, $\mathcal{O}_{\mathcal{Q}}$ and $\mathcal{O}_x$. We show that one can attain both desired scalings in the number of calls to $\mathcal{O}_x$, $\mathcal{O}_{\mathcal{Q}}$ and $\mathcal{O}_{\mathcal{A}}$ simultaneously, up to a single logarithmic factor. Moreover, our construction is also efficient with respect to the time and space complexities, as we show that we only suffer from polylogarithmic overhead in the number of extra gates and auxiliary qubits. As our construction goes via a rather simple composition of span programs, this exemplifies the power of the span program framework.

There are, however, some aspects to Ambainis's work that we did not reproduce. Ambainis proved a version of his theorem for the search problem whereas we only consider a decision version. By a standard reduction from the search version to the decision version, we also recover the analogous search result, but with an extra factor of $\log(n)$ overhead in the query and time complexities. Ambainis also gives a result for the case where the costs of the original algorithms are unknown. We leave modifying our approach to reproduce this result for future research.

From Theorem 13 we easily deduce that if we have efficient uniform access to a set of algorithms, i.e., the oracles $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{Q}}$ can be implemented in time logarithmic in $T_{\max}$ and n, then the algorithm compiled from P has query complexity $\widetilde{\mathcal{O}}(\sqrt{\sum_{k=1}^n Q_k^2})$ and time complexity $\widetilde{\mathcal{O}}(\sqrt{\sum_{k=1}^n T_k^2})$.

6 Open problems

There remain several open problems related to our results. The main question that arises from Theorem 9 is whether it is possible to remove the logarithmic factors in the number of calls to $\mathcal{O}_x$, $\mathcal{O}_{\mathcal{Q}}$ and $\mathcal{O}_{\mathcal{A}}$.

On the other hand, Theorem 13 leaves several open ends for further research. The most interesting direction that we foresee is whether the relative ease with which span programs can be composed can be exploited to obtain more composition results. The variable time search result composes a set of arbitrary functions with the OR function and obtains a Grover-like speed-up in the query and time complexity of the resulting algorithm. A natural next step would be to investigate if similar types of speed-ups can be obtained when one composes some arbitrary functions with threshold functions.

References

1 Andris Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007. `doi:10.1137/S0097539705447311`.

2 Andris Ambainis. Quantum search with variable times. *Theory of Computing Systems*, 47(3):786–807, October 2010. `doi:10.1007/s00224-009-9219-1`.

3 Agnis Āriņš. Span-program-based quantum algorithms for graph bipartiteness and connectivity. In Jan Kofroň and Tomáš Vojnar, editors, *Mathematical and Engineering Methods in Computer Science (MEMICS 2015)*, volume 9548 of *Lecture Notes in Computer Science*, pages 35–41. Springer International Publishing, 2016. `doi:10.1007/978-3-319-29817-7_4`.

4 Salman Beigi and Leila Taghavi. Quantum speedup based on classical decision trees. *Quantum*, 4(241), 2020. `doi:10.22331/q-2020-03-02-241`.

5 Aleksandrs Belovs. Learning-graph-based quantum algorithm for k-distinctness. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 207–216, October 2012. `doi:10.1109/FOCS.2012.18`.

6 Aleksandrs Belovs. Span programs for functions with constant-sized 1-certificates. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 77–84, New York, NY, USA, 2012. Association for Computing Machinery. `doi:10.1145/2213977.2213985`.

7 Aleksandrs Belovs and Ben W. Reichardt. Span programs and quantum algorithms for *st*-connectivity and claw detection. In Leah Epstein and Paolo Ferragina, editors, *Algorithms – ESA 2012*, volume 7501 of *Lecture Notes in Computer Science*, pages 193–204, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. `doi:10.1007/978-3-642-33090-2_18`.

8 Dominic W. Berry, Andrew M. Childs, Richard Cleve, Robin Kothari, and Rolando D. Somma. Simulating Hamiltonian dynamics with a truncated Taylor series. *Phys. Rev. Lett.*, 114(9):090502, March 2015. `doi:10.1103/PhysRevLett.114.090502`.

9 Chris Cade, Ashley Montanaro, and Aleksandrs Belovs. Time and space efficient quantum algorithms for detecting cycles and testing bipartiteness. *Quantum Information and Computation*, 18(1&2):0018–0050, February 2018. `doi:10.26421/QIC18.1-2`.

10 Arjan Cornelissen, Stacey Jeffery, Maris Ozols, and Alvaro Piedrafita. Span programs and quantum time complexity, 2020. `arXiv:2005.01323`.

11 Tsuyoshi Ito and Stacey Jeffery. Approximate span programs. *Algorithmica*, 81(6):2158–2195, 2019. `doi:10.1007/s00453-018-0527-1`.

12 Michael Jarret, Stacey Jeffery, Shelby Kimmel, and Alvaro Piedrafita. Quantum algorithms for connectivity and related problems. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49:1–49:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.ESA.2018.49`.

13 Stacey Jeffery. Span programs and quantum space complexity. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, volume 151 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:37, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.ITCS.2020.4`.

14 Stacey Jeffery and Shelby Kimmel. Quantum algorithms for graph connectivity and formula evaluation. *Quantum*, 1:26, August 2017. `doi:10.22331/q-2017-08-17-26`.

15 Mauricio Karchmer and Avi Wigderson. On span programs. In *Proceedings of the 8th Annual IEEE Conference on Structure in Complexity Theory*, pages 102–111, May 1993. doi:10.1109/SCT.1993.336536.

16 Troy Lee, Frédéric Magniez, and Miklos Santha. Improved quantum query algorithms for triangle detection and associativity testing. *Algorithmica*, 77(2):459–486, 2017. doi:10.1007/s00453-015-0084-9.

17 Ben W. Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every Boolean function. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2009)*, pages 544–551, October 2009. doi:10.1109/FOCS.2009.55.

18 Ben W. Reichardt and Robert Špalek. Span-program-based quantum algorithm for evaluating formulas. *Theory of Computing*, 8(13):291–319, 2012. doi:10.4086/toc.2012.v008a013.

19 Qisheng Wang and Mingsheng Ying. Quantum random access stored-program machines, 2020. arXiv:2003.03514.