Multidimensional Quantum Walks, Recursion, and Quantum Divide & Conquer

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

Galina Pass
QuSoft, Amsterdam, The Netherlands
University of Amsterdam, The Netherlands

Abstract

We introduce an object called a *subspace graph* that formalizes the technique of multidimensional quantum walks. Composing subspace graphs allows one to seamlessly combine quantum and classical reasoning, keeping a classical structure in mind, while abstracting quantum parts into subgraphs with simple boundaries as needed. As an example, we show how to combine a *switching network* with arbitrary quantum subroutines, to compute a composed function. As another application, we give a time-efficient implementation of quantum Divide & Conquer when the sub-problems are combined via a Boolean formula. We use this to quadratically speed up Savitch's algorithm for directed *st*-connectivity.

2012 ACM Subject Classification Theory of computation → Quantum computation theory; Theory of computation → Design and analysis of algorithms

Keywords and phrases Quantum Divide & Conquer, Time-Efficient, Subspace Graphs, Quantum Walks, Switching Networks, Directed st-Connectivity

Digital Object Identifier 10.4230/LIPIcs.STACS.2025.54

Related Version *Full Version*: `https://arxiv.org/abs/2401.08355`

Funding *Stacey Jeffery*: Supported by ERC STG grant 101040624-ASC-Q, NWO Klein project number OCENW.Klein.061. SJ is a CIFAR Fellow in the Quantum Information Science Program. *Galina Pass*: Supported by the National Agenda for Quantum Technologies (NAQT), as part of the Quantum Delta NL programme.

1 Introduction

There are a number of graphical ways of reasoning about how the steps or subroutines of a classical algorithm fit together. For example, it is natural to think of a (randomized) classical algorithm as a (randomized) decision tree (or branching program), where different paths are chosen depending on the input, as well as random choices made by the algorithm. A deterministic algorithm gives rise to a computation *path*, a randomized algorithm to a computation *tree*. The edges of a path or tree, representing steps of computation, might be implemented by some subroutine that is also realized by a path (or tree) – we can abstract the subroutine's details by viewing it as an edge, or zoom in and see those details, as convenient. More generally, we often think of a classical randomized algorithm as a random walk on a (possibly directed) graph, where there may be multiple parallel paths from point a to point b, with the cost of getting from a to b being derived from the expected length of these paths.

This picture appears to break down for quantum algorithms, at least in the standard circuit model. A quantum circuit can be thought of as a path, with edges representing its steps, but it is unclear how to augment this reasoning with subroutines. Consider calling subroutines with varying time complexities $\{T_i\}_i$ in superposition. Even if the subroutines

© Stacey Jeffery and Galina Pass;
licensed under Creative Commons License CC-BY 4.0

are all classical deterministic, in the standard quantum circuit model, we tend to incur a cost of $\max_i T_i$ if we call a superposition of subroutines, since we must wait for the slowest subroutine to finish before we can apply the next step of the computation. This problem was addressed in [13], where the technique of multidimensional quantum walks [15] was used to show how to get an average in place of a max in several settings where a quantum algorithm calls subroutines in superposition: a general setting, as well as the setting of quantum walks. The intuition behind [13] is that a quantum walk does keep the classical intuition of parallel paths representing a superposition of possible computations, and *any* quantum algorithm can be viewed as some sort of a quantum walk on a simple underlying graph (something like a path), but with some additional structure associated with it.

Multidimensional quantum walks, which we study more formally in this paper as an object than has been done previously, are valuable as a way of combining quantum and classical reasoning. A quantum algorithm can be abstracted as a graph with perhaps complicated internal structure, but a simple *boundary* with an "in" and an "out" terminal (called "s" and "t"), that can be seamlessly hooked into other graph-like structures, perhaps representing simple classical reasoning, such as a quantum random walk, or perhaps with their own complicated very quantum parts.

Subspace Graphs

While [15] and [13] use similar techniques, it is not formally defined what a multidimensional quantum walk is. We formally define an object called a *subspace graph* (Definition 2) that abstracts the structures in [15] and [13]. A subspace graph is simply a graph with some subspaces associated with each edge and vertex, where the structure of the graph constrains how the spaces can overlap. Defining what we mean, precisely, by a multidimensional quantum walk (i.e. subspace graph) is the first step to developing a general theory of recursive constructions of subspace graphs.

The recursive structure of subspace graphs is useful for composing quantum algorithms, but as a design tool, it is also convenient to be able to view a subspace graph in varying levels of abstraction. We can "zoom out" and view a complicated process as just a special "edge", or zoom in on that edge and understand its structure as an involved graph with additional structure.

We cannot hope to be able to understand all quantum algorithms using purely classical ideas – quantum computing is *not* classical computing. But perhaps the next best thing is a way to seamlessly combine classical and quantum ideas, extending the classical intuition to its limits, and then employing quantum reasoning when needed, but with the possibility of abstracting out from it when needed as well, using a fully quantum form of abstraction.

In this work, we consider one specific kind of composition of subspace graphs called *switch composition* – another type is implicit in [13] – but we would like to emphasize the potential for more general types of recursion, which we leave for future work.

Time-Efficient Quantum Divide & Conquer

A particular type of recursive algorithm is *divide & conquer*, in which a problem is broken into multiple smaller sub-problems, whose solutions, obtained by recursive calls, are combined into a solution for the original problem. As a motivating example, consider the recursively defined *nand-tree function.* Let $f_{k,d} : \{0,1\}^{d^k} \to \{0,1\}$ be defined $f_{0,d}(x) = x$, and for $k \geq 1$,

$$f_{k,d}(x) = \text{NAND}\left(f_{k-1,d}(x^{(1)}), \dots, f_{k-1,d}(x^{(d)})\right) := 1 - f_{k-1,d}(x^{(1)}) \dots f_{k-1,d}(x^{(d)}),$$

where each $x^{(j)} \in \{0,1\}^{d^{k-1}}$, and $x = (x^{(1)}, \dots, x^{(d)})$. There is a natural way to break an instance x of $f_{k,d}$ into d sub-problems $x^{(1)}, \dots, x^{(d)}$ of $f_{k-1,d}$, and combine the solutions by taking the NAND (negated AND) of the d sub-problem solutions. Grover's algorithm computes this NAND in $O(\sqrt{d})$ queries, so we might hope for a speedup by recursive calls to this quantum algorithm. Unfortunately, since we recurse to depth k, the constant in front of $\sqrt{d}$ is raised to the k-th power. This kills the quantum speedup completely when d is constant (for example, the most common setting of $d = 2$), and that is not even touching on the fact that we would seem to need to amplify the success probability of the subroutine, turning those constants into log factors. On the other hand, it is known [20] that $f_{k,d}$ can be evaluated in $O(\sqrt{d^k})$ quantum queries, even though our attempt to use classical divide-&-conquer reasoning combined with the basic Grover speedup failed.

More recently, [9] showed how to employ divide-&-conquer reasoning in the study of quantum query complexity, in which one only counts the number of queries to the input. They obtained their query upper bounds by composing *dual adversary solutions*. The key to their results is that dual adversary solutions exhibit *perfect* composition: no error, no log factors, not even constant overhead. However, their result were not constructive, as dual adversary solutions do not fully specify algorithms, and in particular, the time complexity analysis of their results was unknown. In this work, we use the framework of subspace graphs to give a time-complexity version of some of the query complexity results obtained in [9]. In particular, we show (see Theorem 16):

▶ **Theorem 1** (Informal). *Let $\{f_{\ell,n} : D_{\ell,n} \to \{0,1\}\}_{\ell,n}$ be a family of functions. Let φ be a symmetric Boolean formula on a variables, and suppose $f_{\ell,n} = \varphi(f_{\ell/b,n}, \dots, f_{\ell/b,n}) \vee f_{aux,\ell,n}$, for some $b > 1$ and some auxiliary function $f_{aux,\ell,n}$ with quantum time complexity $T_{\mathrm{aux}}(\ell,n)$. Then the quantum time complexity of $f_{\ell,n}$ is $\widetilde{O}(T(\ell,n))$ for $T(\ell,n)$ satisfying:*

$$T(\ell,n) \le \sqrt{a}T(\ell/b,n) + T_{\mathrm{aux}}(\ell,n).$$

Our framework also handles the case where $f_{\ell,n} = \varphi(f_{\ell/b,n}, \dots, f_{\ell/b,n}, f_{\mathrm{aux},\ell,n})$ for any formula φ on $a+1$ variables, but then some extra, somewhat complicated looking costs need to be accounted for, and there is also a scaling in the depth, although this is not an issue if the formula has been preprocessed to be balanced [8].

Comparing this with the analogous classical statement, which would have a instead of $\sqrt{a}$, we get an up to quadratic speedup over a large class of classical divide-&-conquer algorithms. As an application, we show a quadratic speedup of Savitch's divide-&-conquer algorithm for directed st-connectivity [22].

To achieve these results, it is essential that we compose subspace graphs, rather than algorithms. When we convert a subspace graph to a quantum algorithm, we get constant factors in the complexity, and these seem necessary without at least specifying what gateset we are working in. By first composing in the more abstract model of subspace graphs, and then only converting to a quantum algorithm at the end, we ensure these factors only come into the complexity once. This is similar to compositions done with other abstract models, such as span programs (of which dual adversary solutions are a special case) [19], and transducers [6].

Switching Networks

A switching network is a graph with Boolean variables associated with the edges that can switch the edges on or off. Originally used to model certain hardware systems, including automatic telephone exchanges, and industrial control equipment [23], a switching network has an associated function f that is 1 if and only if two special vertices, s and t, are connected by a path of "on" edges. Shannon [23, 24] showed that series-parallel switching networks are equivalent to Boolean formulas, and Lee [16] showed that switching networks can model branching programs. These theoretical results have given this model a place in classical complexity theory, where they can be used to study classical space complexity and circuit depth (see [18]).

Quantum algorithms for evaluating switching networks[1] given query access to the edge variables were given in [14], using a span program construction based heavily on [7]. Let $\mathcal{R}_{s,t}(G(x))$ be the effective resistance in the subgraph of "on" edges whenever $f(x) = 1$, and whenever $f(x) = 0$, let $F_x \subseteq E$ be the minimum weight st-cut-set consisting of only edges that are "off". Then there is a quantum algorithm evaluating the switching network using

$$\sqrt{\max_{x \in f^{-1}(1)} \mathcal{R}_{s,t}(G(x)) \max_{x \in f^{-1}(0)} \sum_{e \in F_x} \mathsf{w}_e},$$

queries, with matching time complexity assuming we can implement a certain reflection related to the particular switching network in unit time. From this it follows that if we can query the variable associated with an edge e in time T_e, we can evaluate the switching network in time

$$\sqrt{\max_{x \in f^{-1}(1)} \mathcal{R}_{s,t}(G(x)) \max_{x \in f^{-1}(0)} \sum_{e \in F_x} \mathsf{w}_e}. \max_{e \in E} T_e.$$

Here we improve this to:

$$\sqrt{\max_{x \in f^{-1}(1)} \mathcal{R}_{s,t}(G(x)) \max_{x \in f^{-1}(0)} \sum_{e \in F_x} \mathsf{w}_e T_e^2}.$$

This is analogous to results of [13], which showed a similar statement, but for quantum walks rather than switching networks[2]. Because switching networks perfectly model Boolean formulas, without the constant overhead we get when we convert to a quantum algorithm, they can be used as a building block for our divide-&-conquer results.

Application to DSTCON

Quantum algorithms for st-connectivity on *undirected graphs*, which are closely related to evaluating switching networks, are well studied [10, 7, 14, 12, 3], including quantum algorithms that achieve optimal time- and space-complexity simultaneously, in both the edge-list access model, and the adjacency matrix access model.[3] In contrast, quantum algorithms for *directed st*-connectivity (DSTCON) – the problem of deciding if there is a directed path from s to t in a directed graph – is less well understood. The algorithm of [10] also applies to directed graphs, deciding connectivity in $\widetilde{O}(n)$ time and space[4]. This algorithm has optimal time complexity, whereas its space complexity is far from optimal.

[1] Switching networks have never been directly referred to in prior work on quantum algorithms, as far as we are aware, but the "st-connectivity problems" referred to in [14] are, in fact, switching networks.
[2] Both our result, and the one for quantum walks in [13] include *variable-time quantum search* [2] as a special case.
[3] We do not make a distinction between various access models, because they can simulate one another in poly(n) time and log(n) space, so our result, which includes a $2^{O(\log n)}$ term, is the same in all models.
[4] In the edge-list access model.

Directed *st*-connectivity, also called *reachability*, is a fundamental problem in classical space complexity. In particular, understanding if this problem can be solved in $\log(n)$ space by a quantum algorithm would resolve the relationship between quantum logspace complexity and NL, as DSTCON is NL-complete.

The best known classical (deterministic) space complexity of DSTCON is $O(\log^2(n))$, using Savitch's algorithm. We apply quantum divide & conquer (Theorem 1) to give a quadratic speedup to Savitch's algorithm, achieving $2^{\frac{1}{2}\log^2(n)+O(\log(n))}$ time, while still maintaining $O(\log^2(n))$ space (see Theorem 18).

Model of Computation

We work in the same model as [15], where we allow not only arbitrary quantum gates, but also assume subroutines are given via access to a unitary that applies the subroutine's t-th gate controlled on the value t in some time register. This is possible, for example, with quantum random access gates.

Related Work

While writing this manuscript, we became aware of an independent work that also achieves time-efficient quantum divide & conquer [1] when either (1) φ is an OR (equivalently, an AND) or (2) φ is a minimum or maximum. OR is a Boolean formula, while minimum/maximum is not. In that sense, our results are incomparable. Ref. [1] applies their framework to problems that are distinct from ours. The framework of [1] also differs from our work in that they explicitly treat the complexity of computing sub-instances (the cost of the "create" step), whereas we assume sub-instances are computable in unit cost. In one of the applications of [1], this cost is not negligible, and is even the dominating cost, so our framework, as stated, would not handle this application. This is not an inherent limitation of our techniques – it would be possible to take this cost into account in our framework as well.

We also mention that Ref. [9] analyzes the quantum query complexity of divide & conquer where φ is an arbitrary Boolean formula, as well as in settings where the function combining the sub-problems is more general. While our techniques do apply to composing quantum algorithms for arbitrary functions (already studied in [13]), an issue is a poor scaling in the error of subroutines. If we start with a bounded-error quantum algorithm for some function, we need to amplify the success probability, as it will be called many times, incurring logarithmic factors. This becomes a serious problem if the function is called recursively to depth more than constant. We get around this in the case of Boolean formulas by using a switching network construction (from which a quantum algorithm could be derived) rather than a bounded-error quantum algorithm for evaluating the formula. Our techniques would thus also readily apply to functions for which there is an efficient quantum algorithm derived from a switching network.

2 Technical Overview

Here we give a technical overview of our results, with full details available in the full version of the paper.

2.1 Subspace Graphs

Multidimensional quantum walks were introduced as such in [15], although they generalize various quantum algorithms that have appeared previously, including [25, 21, 4, 5, 11]. We wish to consider very general kinds of composition of multidimensional quantum walks, and in order to be clear about the precise types of objects we are composing, we give a more formal definition than has appeared previously.

▶ **Definition 2** (Subspace Graph). *A* subspace graph *consists of a (undirected) graph $G = (V, E)$, a boundary $B \subseteq V$, and the following subspaces of a space $H = H_G$:*

Edge and Boundary Spaces *We assume H can be decomposed into a direct sum of spaces as follows: $H = \bigoplus_{e\in E} \Xi_e \oplus \bigoplus_{u\in B} \Xi_u$.*

Edge and Boundary Subspaces *For each $e \in E \cup B$, let $\Xi_e^{\mathcal{A}}$ and $\Xi_e^{\mathcal{B}}$ be subspaces of Ξ_e. These need not be orthogonal, and they may each be $\{0\}$, all of Ξ_e, or something in between.*

Vertex Spaces and Boundary Space *For each $u \in V$, let $\mathcal{V}_u \subseteq \bigoplus_{e\in E(u)} \Xi_e$ be pairwise orthogonal spaces. Let $\mathcal{V}_B \subseteq \bigoplus_{u\in B} \Xi_u$.*

Then we define

$$\mathcal{A}_G = \bigoplus_{e\in E\cup B} \Xi_e^{\mathcal{A}} \text{ and } \mathcal{B}_G = \bigoplus_{u\in V} \mathcal{V}_u + \mathcal{V}_B + \bigoplus_{e\in E\cup B} \Xi_e^{\mathcal{B}}.$$

To motivate this perhaps complicated-looking definition, we mention some concrete examples of subspace graphs that already exist in the quantum algorithms literature:

1. A discrete-time quantum walk in Szegedy's framework [25], or the more general electric network framework [4] is a subspace graph.
2. The span programs for *st*-connectivity in [14], based on [7], are subspace graphs. These are actually precisely switching networks, which we will discuss more shortly (although the basis constructions for Boolean switching networks in this work are new).
3. In [13], certain subspaces are defined from any quantum algorithm, and these actually make up a subspace graph (see Section 2.2.3).

We will allow subspace graphs to implicitly depend on some input, and associate them with a computation. Specifically, if we fix some initial state $|\psi_0\rangle \in H_G$, then this, along with $\mathcal{A}_G$ and $\mathcal{B}_G$, define a phase estimation algorithm that decides if $|\psi_0\rangle \in \mathcal{A}_G + \mathcal{B}_G$ by doing phase estimation of $(2\Pi_{\mathcal{A}_G} - I)(2\Pi_{\mathcal{B}_G} - I)$, where $\Pi_{\mathcal{A}_G}$ is the orthogonal projector onto $\mathcal{A}_G$, and similarly for $\mathcal{B}_G$. With this in mind, we say a subspace graph that depends on some input x, *computes a function f* (with respect to $|\psi_0\rangle$) if $f(x) = 0$ if and only if $|\psi_0\rangle \in \mathcal{A}_G + \mathcal{B}_G$. Let us give some intuition. The quantum algorithm corresponding to a subspace graph described above distinguishes between two cases:

Negative Case: $|\psi_0\rangle \in \mathcal{A}_G + \mathcal{B}_G$

Positive Case: $|\psi_0\rangle$ has a (large) component, called a *positive witness* in $(\mathcal{A}_G + \mathcal{B}_G)^\perp = \mathcal{A}_G^\perp \cap \mathcal{B}_G^\perp$

Then we can see all the vectors in $\mathcal{A}_G$ and $\mathcal{B}_G$ as linear constraints on the initial state – it must have a large component orthogonal to all of them in the positive case. $\mathcal{A}_G$ and $\mathcal{B}_G$ are each decomposed into sums of subspaces, representing subsets of constraints. The graph structure of G restricts how the different sets of constraints are allowed to overlap – the constraints associated with vertex v only overlap constraints associated with edges that are incident to v. This graph structure can then be used to help analyse the algorithm. For example, a positive witness is related to a *flow* on G.

Though it is possible to consider more general states $|\psi_0\rangle$, throughout this paper, we will assume G has a pair of special "terminal" vertices s and t, and let $|\psi_0\rangle = \frac{1}{\sqrt{2}}(|s\rangle - |t\rangle)$.

To implement the phase estimation algorithm, we need a pair of orthogonal bases for $\mathcal{A}_G$ and $\mathcal{B}_G$ that are easily generated, in order to implement their respective reflections. We therefore always assume we have such a basis pair associated with G, which we call the *working bases.*

To analyze a phase estimation algorithm, we need to exhibit positive and negative witnesses, which have a particular form in the case G has a property called *canonical st-boundary* (see Definition 8), as will always be the case in this paper. In particular, st-boundary implies that $\bigoplus_{u\in B}\Xi_u=\Xi_s\oplus\Xi_t$ has four degrees of freedom, which we denote $|s\rangle, |\leftarrow,s\rangle, |t\rangle, |\rightarrow,t\rangle$.

▶ **Definition 3** (Positive Witness for a Graph). *We say $|\hat{w}\rangle\in\Xi_E$ is a positive witness for G if*

$$|s\rangle-|\leftarrow,s\rangle+|\hat{w}\rangle+|\rightarrow,t\rangle-|t\rangle\in\mathcal{A}_G^{\perp}\cap\mathcal{B}_G^{\perp}.$$

We let $\hat{W}_+(G)$ be an upper bound (over some implicit input) on the minimum $\||\hat{w}\rangle\|^2$ of any positive witness for G.

A negative witness for a phase estimation algorithm is a vector $|w_{\mathcal{A}}\rangle\in\mathcal{A}$ such that $|w_{\mathcal{B}}\rangle:=|\psi_0\rangle-|w_{\mathcal{A}}\rangle\in\mathcal{B}$, which exists if and only if $|\psi_0\rangle\in\mathcal{A}_G+\mathcal{B}_G$. For a subspace graph, a negative witness is defined as follows.

▶ **Definition 4** (Negative Witness for a Graph). *We say $|\hat{w}_{\mathcal{A}}\rangle\in\mathcal{A}_G$ is a negative witness for G if $|\hat{w}_{\mathcal{A}}\rangle+|\leftarrow,s\rangle-|\rightarrow,t\rangle\in\mathcal{B}_G$. We let $\hat{W}_-(G)$ be an upper bound (over some implicit input) on the minimum $\||\hat{w}_{\mathcal{A}}\rangle\|^2$ of any negative witness for G.*

This leads to the following theorem associating a quantum algorithm to a subspace graph.

▶ **Theorem 5.** *Let G be a subspace graph that computes $f:\{0,1\}^n\rightarrow\{0,1\}$, with working bases that can be generated in time T. Then there exists a quantum algorithm that decides f in time complexity $O\left(T\sqrt{\hat{W}_+(G)\hat{W}_-(G)}\right)$ and space $O(\log\dim H_G+\log(\hat{W}_+(G)\hat{W}_-(G)))$.*

This justifies defining $\hat{\mathcal{C}}(G):=\sqrt{\hat{W}_+(G)\hat{W}_-(G)}$ as the *complexity* of a subspace graph.

Switches and Switching Networks

Next, we introduce switching networks. In the classical model of switching networks, a switching network is a graph with a literal (variable or negated variable) $\varphi(e)$ associated with each edge e, and two terminals $s,t\in V$. A switching network computes a function $f:\{0,1\}^E\rightarrow\{0,1\}$ if for any $x\in\{0,1\}^E$, $f(x)=1$ if and only if s and t are connected in $G(x)$, the subgraph consisting only of edges e such that $x_e=1$ if $\varphi(e)$ is a positive literal, and $x_e=0$ otherwise – that is, edges labelled by literals that are true when the variables are set according to x. Here, we let "switching network" refer to a special kind of subspace graph, but this subspace graph implements the switching network, in the sense that the algorithm referred to in Theorem 5 decides if s and t are connected in $G(x)$.

We first define what it means for an edge to be a switch. For a vertex $u\in V$, we let $E(u)$ denote the edges incident to u, which we divide into $E^{\rightarrow}(u)$ – edges "coming out of" u – and $E^{\leftarrow}(u)$ – edges "going into" u. As G is an undirected graph, these edge directions can be chosen arbitrarily.

▶ **Definition 6** (Switch Edge). *Fix a subspace graph G. We call an edge $e\in E$ a* switch *(or* switch edge*) if there is some value $\varphi(e)\in\{0,1\}$ associated with that edge (implicitly depending on the input), such that*

$$\Xi_e=\mathrm{span}\{|\rightarrow,e\rangle,|\leftarrow,e\rangle\},$$
$$\Xi_e^{\mathcal{A}}=\mathrm{span}\{|\rightarrow,e\rangle-(-1)^{\varphi(e)}|\leftarrow,e\rangle\}\quad\textit{and}\quad\Xi_e^{\mathcal{B}}=\mathrm{span}\{|\rightarrow,e\rangle+|\leftarrow,e\rangle\},$$

and moreover, if $e\in E^{\rightarrow}(u)\cap E^{\leftarrow}(v)$, (that is, $e=(u,v)$), then $\mathcal{V}_u\cap\Xi_e=\mathrm{span}\{|\rightarrow,e\rangle\}$ and $\mathcal{V}_v\cap\Xi_e=\mathrm{span}\{|\leftarrow,e\rangle\}$.

The idea behind a switch edge is that if $\varphi(e)=0$, $\Xi_e^{\mathcal{A}}+\Xi_e^{\mathcal{B}}=\Xi_e$, and so $\Xi_e^{\perp}=\{0\}$. Recall that a *positive witness* is some $|\hat{w}\rangle$ such that $|w\rangle := |s\rangle - |\leftarrow, s\rangle + |\hat{w}\rangle + |\rightarrow, t\rangle - |t\rangle \in \mathcal{A}_G^{\perp}\cap\mathcal{B}_G^{\perp}$. If $\Xi_e^{\perp}=\{0\}$, then $|w\rangle$ can have no overlap with Ξ_e, so e is essentially blocked from use, so we say the edge is *switched off*. In switching networks, defined shortly, the positive witness is an st-flow, and it is restricted to edges in the subgraph $G(x)$ of edges that are switched on. Even in subspace graphs that are not switching networks, we can often think of $|w\rangle$ intuitively as a kind of flow.

We next define *simple vertices*, which are the type of vertices of switching networks, and also quantum walks.

▶ **Definition 7** (Simple Vertex). *Fix a subspace graph G with associated edge weights $\{\mathsf{w}_e\}_{e\in E}$. For $u \in V$, define*

$$|\psi_\star(u)\rangle := \sum_{e\in E^{\rightarrow}(u)} \sqrt{\mathsf{w}_e}|\rightarrow, e\rangle + \sum_{e\in E^{\leftarrow}(u)} \sqrt{\mathsf{w}_e}|\leftarrow, e\rangle.$$

A vertex $u\in V$ is simple *if for all $e\in E^{\rightarrow}(u)$, $|\rightarrow, e\rangle\in\Xi_e$, for all $e\in E^{\leftarrow}(u)$, $|\leftarrow, e\rangle\in\Xi_e$, and if $u\in V\setminus B$ $\mathcal{V}_u=\mathrm{span}\{|\psi_\star(u)\rangle\}$; and if $u\in B$, either $|\rightarrow, u\rangle\in\Xi_u$ and $\mathcal{V}_u=\mathrm{span}\{|\rightarrow, u\rangle+|\psi_\star(u)\rangle\}$ or $|\leftarrow, u\rangle\in\Xi_u$ and $\mathcal{V}_u=\mathrm{span}\{|\leftarrow, u\rangle+|\psi_\star(u)\rangle\}$.*

Let us motivate this definition. In a random walk on a weighted graph, in order to take a step from a vertex $u\in V$ to a neighbour, the random walker chooses an edge $e\in E(u)$ to traverse, with probability $\mathsf{w}_e/(\sum_{e'\in E(u)}\mathsf{w}_{e'})$. This is precisely the distribution obtained by measuring $|\psi_\star(u)\rangle/\,\||\psi_\star(u)\rangle\|$. The states $|\psi_\star(u)\rangle$ correspond to *quantum walk states* – alternating a reflection around these with a reflection around the states $\{|\rightarrow, e\rangle+|\leftarrow, e\rangle\}_{e\in E}$ implements a discrete-time quantum walk, as in [25, 17, 4].[5]

Another important notion is that of *canonical st-boundary* for which we require the boundary to consist only of two vertices s and t with additional requirements on their subspaces.

▶ **Definition 8** (Canonical st-boundary). *We say a subspace graph G has* canonical st-boundary *if $B=\{s,t\}$, where:*

- $\Xi_s=\mathrm{span}\{|s\rangle, |\leftarrow, s\rangle\}$, $\Xi_s^{\mathcal{A}}=\mathrm{span}\{|s\rangle+|\leftarrow, s\rangle\}$ *and* $\Xi_s^{\mathcal{B}}=\{0\}$, *where $\mathcal{V}_s$ only overlaps* $|\leftarrow, s\rangle$.
- $\Xi_t=\mathrm{span}\{|\rightarrow, t\rangle, |t\rangle\}$, $\Xi_t^{\mathcal{A}}=\mathrm{span}\{|\rightarrow, t\rangle+|t\rangle\}$ *and* $\Xi_t^{\mathcal{B}}=\{0\}$, *where $\mathcal{V}_t$ only overlaps* $|\rightarrow, t\rangle$.
- $|\leftarrow, s\rangle+|\rightarrow, t\rangle\in\mathcal{B}_G$.

▶ **Definition 9** (Switching Network). *A* switching network *is a subspace graph with canonical st-boundary in which all edges are switches (Definition 6), and all vertices are simple (Definition 7).*

Two more important definitions for the composition results are those of *composable bases* and *st-composable subspace graph* (Definition 10). The first one specifies conditions on working bases so that they can be smoothly composed with each other. We leave the rigorous definition to the full version of the paper since it is quite technical.

[5] In some previous works, it is assumed that the graph is bipartite, and then the walk is implemented by alternating reflections around the states $|\psi_\star(u)\rangle$ of the two parts of the bipartition. We are actually doing the same thing here, we have just ensured the graph is bipartite by inserting a vertex in the middle of each edge.

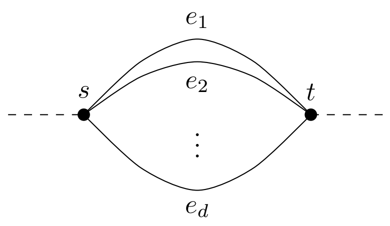

Figure 1 The graph G_{OR}. The dashed lines represent dangling boundary "edges".

▶ **Definition 10.** *We say a subspace graph G is st-*composable *if:*
1. *G has canonical st-boundary (Definition 8);*
2. *G is equipped with composable working bases;*
3. *for each $e \in E \setminus \overline{E}$, $\Xi_e^{\mathcal{B}} = \{0\}$, where $\overline{E}$ is the set of edges that are switches.*

An st-composable subspace graph is a generalization of a switching network to allow for more complicated structures, such as arbitrary quantum algorithms. The main technical contribution of this paper (the composition theorem in Section 2.3) is to generalize a natural way that switching networks compose, as graphs, to any st-composable subspace graph.

2.2 Examples

We first give two examples of switching networks – one for computing the OR of d bits (Section 2.2.1) and one for the AND of d bits (Section 2.2.2) – which are also important building blocks for our later results. At a high level, these are quite simple to understand. The switching network for OR is just d parallel edges from s to t (see Figure 1) – s and t are connected if at least one of the edges is present. The switching network for AND is just a path of length d from s to t (see Figure 2) – s and t are connected if all of the edges are present. One can build up a graph that represents any formula by compositions where an edge is replaced by a graph whose terminals s and t are identified with the endpoints of the replaced edge (see also Figure 4). Our main composition theorem (see Section 2.3) generalizes this type of composition to apply to any st-composable subspace graph.

2.2.1 Switching network for OR

A switching network that computes the OR of d Boolean variables consists of two vertices connected by d parallel switch edges. The idea is that there is no flow if all the edges are blocked, that is all the variables take value 0, an there is a flow otherwise. Therefore, a negative witness corresponds to a cut in the graph, and a positive witness corresponds to a flow.

▶ **Lemma 11.** *For any $d \geq 1$, and positive weights $\{\mathsf{w}_i\}_{i\in[d]}$, there is a switching network $G_{\text{OR},d}$ that computes $\bigvee_{i=1}^{d} \varphi(e_i)$ with $\dim H_{G_{\text{OR},d}} = 2d+4$ such that:*
1. *$G_{\text{OR},d}$ has st-composable working bases that can be generated in $O(\log d)$ time, assuming the state proportional to $\sum_{i=1}^{d} \sqrt{\mathsf{w}_i}|i\rangle$ can be generated in time $O(\log d)$;*
2. *if $\varphi(e_i) = 1$ for some $i \in [d]$, $G_{\text{OR},d}$ has positive witness $|\hat{w}\rangle = \frac{1}{\sqrt{\mathsf{w}_i}}(|\rightarrow, i\rangle - |\leftarrow, i\rangle)$; and*
3. *if $\varphi(e_i) = 0$ for all $i \in [d]$, $G_{\text{OR},d}$ has negative witness $|\hat{\omega}_{\mathcal{A}}\rangle = \sum_{i=1}^{d} \sqrt{\mathsf{w}_i}(|\rightarrow, i\rangle - |\leftarrow, i\rangle)$.*

We remark that if $\mathsf{w}_i = 1$ for all i, then Lemma 11 implies $\hat{W}_+(G_{\text{OR},d}) = 2$ and $\hat{W}_-(G_{\text{OR},d}) = 2d$, so by Theorem 5, there is a quantum algorithm for evaluating d-bit OR with time complexity $\widetilde{O}(\sqrt{d})$, which is optimal. This is a good sanity check, but we will mostly be interested in this construction as a building block, rather than in its own right.

Let $G = G_{\text{OR},d}$ be defined, as shown in Figure 1 by:

$$V = \{s,t\} \text{ and } E = \{e_i : i \in [d]\}$$

where each e_i has endpoints s and t, so $E(s) = E^{\rightarrow}(s) = E$ and $E(t) = E^{\leftarrow}(t) = E$. Since G is a switching network, it has boundary $B = V = \{s,t\}$. We will let the graph be weighted, with weights $\mathsf{w}_{e_i} = \mathsf{w}_i$. The only reason to let these vary in i is for later when we replace an edge with a gadget by composition, but for the sake of intuition, the reader may wish to imagine $\mathsf{w}_i = 1$ for all i. Since G is a switching network, every edge is a switch, which fixes the following spaces (we simplify notation by using i to label the edge e_i):

$$\begin{aligned}\forall i \in [d], \Xi_{e_i} &= \text{span}\{|\rightarrow, i\rangle, |\leftarrow, i\rangle\},\\ \Xi_{e_i}^{\mathcal{A}} &= \text{span}\{|\rightarrow, i\rangle - (-1)^{\varphi(e_i)}|\leftarrow, i\rangle\}, \text{ and } \Xi_{e_i}^{\mathcal{B}} = \text{span}\{|\rightarrow, i\rangle + |\leftarrow, i\rangle\}.\end{aligned}$$

Furthermore, as a switching network has canonical st-boundary, the following spaces are fixed:

$$\begin{aligned}\Xi_s &= \text{span}\{|s\rangle, |\leftarrow, s\rangle\}, \quad \Xi_s^{\mathcal{A}} = \text{span}\{|s\rangle + |\leftarrow, s\rangle\}, \text{ and } \Xi_s^{\mathcal{B}} = \{0\}\\ \Xi_t &= \text{span}\{|\rightarrow, t\rangle, |t\rangle\}, \quad \Xi_t^{\mathcal{A}} = \text{span}\{|\rightarrow, t\rangle + |t\rangle\}, \text{ and } \Xi_t^{\mathcal{B}} = \{0\}.\end{aligned}$$

Finally, since all vertices are simple, the following spaces are fixed:

$$\mathcal{V}_s = \text{span}\Bigg\{\underbrace{|\leftarrow, s\rangle + \sum_{i=1}^{d} \sqrt{\mathsf{w}_i}|\rightarrow, i\rangle}_{|\psi_\star(s)\rangle}\Bigg\} \text{ and } \mathcal{V}_t = \text{span}\Bigg\{\underbrace{\sum_{i=1}^{d} \sqrt{\mathsf{w}_i}|\leftarrow, i\rangle + |\rightarrow, t\rangle}_{|\psi_\star(t)\rangle}\Bigg\}.$$

Then we have:

$$\begin{aligned}\mathcal{A}_G &= \bigoplus_{e\in E\cup B} \Xi_e^{\mathcal{A}} = \text{span}\{|\rightarrow, i\rangle - (-1)^{\varphi(e_i)}|\leftarrow, i\rangle : i \in [d]\} \cup \{|s\rangle + |\leftarrow, s\rangle, |t\rangle + |\rightarrow, t\rangle\}\\ \text{and } \mathcal{B}_G &= \mathcal{V}_s \oplus \mathcal{V}_t + \bigoplus_{e\in E} \Xi_e^{\mathcal{B}} = \mathcal{V}_s \oplus \mathcal{V}_t + \text{span}\{|\rightarrow, i\rangle + |\leftarrow, i\rangle : i \in [d]\}. \qquad (1)\end{aligned}$$

We prove the second and the third items of Lemma 11 in the full version of the paper. It remains to find working bases that can be generated efficiently. We show in the full version that there is an st-composable working basis for $\mathcal{A}_G$ that can be generated in unit time and there is a st-composable working basis for $\mathcal{B}_G$ that can be generated in $O(\log d)$ time.

2.2.2 Switching network for AND

In this section, we describe a switching network that computes the AND of d Boolean variables. This switching network consists of two boundary vertices connected by a path of switch-edges of length d. If at least one edge is blocked, that is at least one of the variables takes value 0, then there is no flow, and there is a flow otherwise.

▶ **Lemma 12.** *For any $d \geq 1$, and positive weights $\{\mathsf{w}_i\}_{i\in d}$, there is a switching network $G_{\text{AND},d}$ that computes $\bigwedge_{i=1}^{d} \varphi(e_i)$ with $\dim H_{G_{\text{AND},d}} = 2d + 4$ such that:*

1. *$G_{\text{AND},d}$ has st-composable working bases that can be generated in $O(\log d)$ time, assuming the state proportional to $\sum_{i=1}^{d} \frac{1}{\sqrt{\mathsf{w}_i}}|i\rangle$ can be generated in time $O(\log d)$;*
2. *if $\varphi(e_i) = 1$ for all $i \in [d]$, $G_{\text{AND},d}$ has positive witness $|\hat{w}\rangle = \sum_{i=1}^{d} \frac{1}{\sqrt{\mathsf{w}_i}}(|\rightarrow, i\rangle - |\leftarrow, i\rangle)$; and*

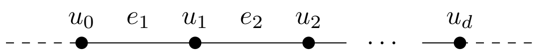

Figure 2 The graph G_{AND}. The dashed lines represent dangling boundary "edges".

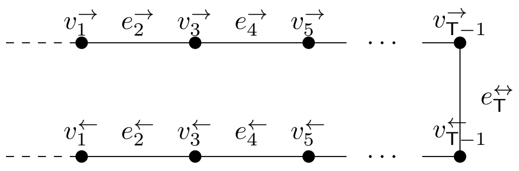

Figure 3 The graph representing a quantum computation with T steps. The top part should be thought of as computing a bit into the phase, and the bottom should be thought of as uncomputing. The dashed lines represent dangling boundary "edges".

3. *if* $\varphi(e_i) = 0$ *for some* $i \in [d]$, $G_{\text{AND},d}$ *has negative witness* $|\hat{w}_{\mathcal{A}}\rangle = \sqrt{\mathsf{w}_i}(|\rightarrow, i\rangle - |\leftarrow, i\rangle)$.

As with the OR switching network, a corollary of this lemma is that there is a quantum algorithm for evaluating AND in optimal time $\widetilde{O}(\sqrt{d})$.

Let $G = G_{\text{AND},d}$ be the graph in Figure 2, defined by

$$V = \{s = u_0, u_1, \ldots, u_d = t\} \text{ and } E = \{e_i = (u_{i-1}, u_i) : i \in [d]\},$$

so $E(s) = E^{\rightarrow}(s) = \{e_1\}$, $E(t) = E^{\leftarrow}(t) = \{e_d\}$, and for all $i \in [d-1]$, $E^{\leftarrow}(u_i) = \{e_i\}$ and $E^{\rightarrow}(u_i) = \{e_{i+1}\}$. Since G is a switching network, it has boundary $B = \{s, t\}$. We will let the graph be weighted, with weights $\mathsf{w}_{e_i} = \mathsf{w}_i$. Since G is a switching network: every edge is a switch, which fixes Ξ_{e_i}, $\Xi_{e_i}^{\mathcal{A}}$ and $\Xi_{e_i}^{\mathcal{B}}$ for all $i \in [d]$; G has canonical st-boundary, which fixes Ξ_s, $\Xi_s^{\mathcal{A}}$, $\Xi_s^{\mathcal{B}}$, Ξ_t, $\Xi_t^{\mathcal{A}}$, and $\Xi_t^{\mathcal{B}}$; and every vertex is simple, which fixes the spaces $\mathcal{V}_{u_i}$ for $i \in \{0, \ldots, d\}$. In particular, letting $\mathsf{w}_0 = \mathsf{w}_{d+1} = 1$, $s = 0$, $t = d+1$, and $i \in [d]$ label e_i, we must have:

$$\forall i \in \{0, \ldots, d\}, \mathcal{V}_{u_i} = \operatorname{span}\left\{\sqrt{\mathsf{w}_i}|\leftarrow, i\rangle + \sqrt{\mathsf{w}_{i+1}}|\rightarrow, i+1\rangle\right\}.$$

This fully defines

$$\mathcal{A}_G = \bigoplus_{e \in E \cup B} \Xi_e^{\mathcal{A}} \quad \text{and} \quad \mathcal{B}_G = \bigoplus_{i=0}^{d} \mathcal{V}_{u_i} + \bigoplus_{e \in E} \Xi_e^{\mathcal{B}}.$$

We prove the second and the third items of Lemma 12 in the full version of the paper. It remains to find working bases that can be generated efficiently. We show in the full version that there is an st-composable working basis for $\mathcal{A}_G$ that can be generated in unit time and there is a st-composable working basis for $\mathcal{B}_G$ that can be generated in $O(\log d)$ time.

2.2.3 Any Quantum Algorithm

Ref. [13] deals with composing arbitrary quantum algorithms, implicitly using subspace graphs for the task. It is shown how to define certain subspaces from an arbitrary quantum algorithm, where the overlap between these various spaces gives rise to a graph as in Figure 3. In the full version of the paper, we show that these subspaces are precisely a subspace graph, so that the following follows from [13].

▶ **Lemma 13.** *From any quantum algorithm computing some* $f : \{0,1\}^n \to \{0,1\}$ *in time* T *with* S *qubits, we can derive an st-composable subspace graph (Definition 10)* G *that computes* f *with* $\dim H_G = O(2^S T)$, $\hat{W}_+(G) \leq 2T$ *and* $\hat{W}_-(G) \leq 2T$*; with st-composable working bases that can be generated in time* $O(\log(T))$.

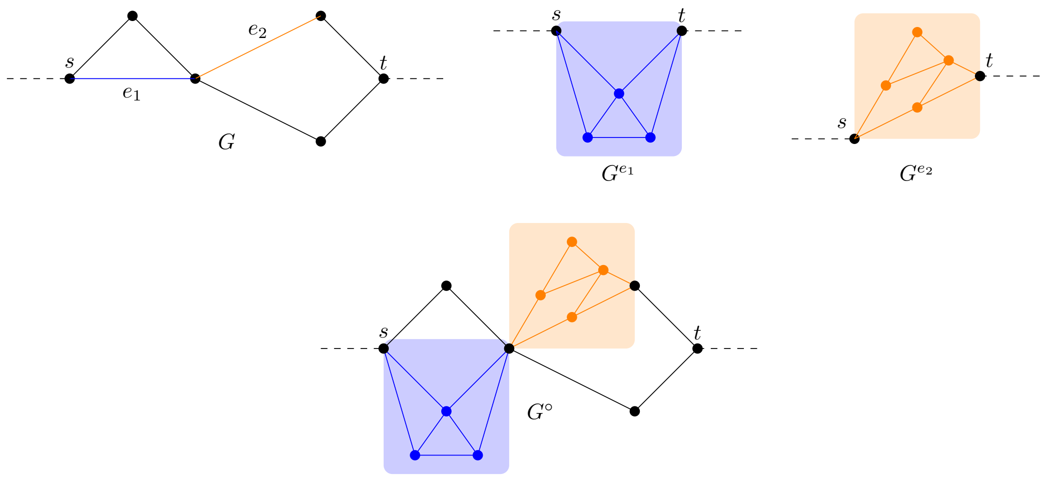

Figure 4 An example of replacing edges e_1 and e_2 in G with graphs G^{e_1} and G^{e_2} to obtain G°. Boundary "edges" are represented by dashed lines. Switching networks already lend themselves to this type of recursion: we can replace an edge with a 2-terminal graph, and then the "edge" is traversable if and only if the terminals are connected. The difference with the more general recursion in Theorem 14 is that the subspace graphs used to replace edges (as well as other parts of G) might have more complicated structure than just their graph structure; for example, they might encode quantum algorithms like in Section 2.2.3.

2.3 Composition

Subspace graphs lend themselves well to very general kinds of recursion. We can compose subspace graphs by identifying some of the vertices on their boundaries. In full generality, this may result in a subspace graph that is difficult to analyze. For example, if we replace two parallel edges with subspace graphs derived from quantum algorithms, "flow" along these edges may have complex phases that interfere on the other side. Our nice classical intuition of flows breaks down in the fully general case, which is not surprising, since quantum algorithms are not classical. However, an important question is in which special cases, and to what extent, we can compose subspace graphs and keep enough classical intuition to analyze them.

One special case is implicit in [13], and here we present another special case, which we call *switch composition*. Switch composition generalizes a very simple kind of recursion that can be done in switching networks. Since an st-path (or more generally, flow) is allowed to use an edge if and only if $\varphi(e) = 1$, we can replace it with a switching network G^e for some function f_e, which will have an st-path from one endpoint to the other if and only if $f_e(x) = 1$. We can view this as removing e from the graph, and then adding its endpoints to the boundary, to then be identified with the boundary $\{s^e, t^e\}$ of G^e. By applying this type of composition, the OR and AND switching networks in Section 2.2.1 and Section 2.2.2 can be combined to make switching networks for any Boolean formula [14].

Here we investigate how to compose subspace graphs with specific properties that generalize switching networks – st-composable subspace graphs (Definition 10) – into the switches of a graph G. Specifically, if $\overline{E}$ is the set of edges of G that are switches, we will replace $e \in \overline{E}$ with an st-composable graph G^e. We can assume without loss of generality that we replace every $e \in \overline{E}$ with some G^e, since letting G^e be the graph consisting of a single edge from s^e to t^e is just like not replacing e.

▶ **Theorem 14.** *Let G be an st-composable subspace graph with st-composable working bases that can be generated in time T. For each $e \in \overline{E}$, the set of switches of G, let G^e be an st-composable subspace graph with st-composable working bases that can be generated in time at most T'. Then there exists an st-composable subspace graph G° with $\dim H_{G^\circ} \leq \dim H_G - 2|\overline{E}| + \sum_{e\in\overline{E}} \dim H_{G^e}$ such that:*

- *G° has st-composable working bases that can be generated in time $T + T' + O(1)$.*
- *If $|\hat{w}\rangle$ is a positive witness for G (Definition 3), and for each $e \in \overline{E}$ such that $\langle \rightarrow, e|\hat{w}\rangle \neq 0$, $|\hat{w}^e\rangle$ is a positive witness for G^e, then*

  $$|\hat{w}^\circ\rangle = \sum_{e\in\overline{E}} \langle \rightarrow, e|\hat{w}\rangle |\hat{w}^e\rangle + \Pi_{E\setminus\overline{E}}|\hat{w}\rangle$$

  is a positive witness for G°.
- *If $|\hat{w}_\mathcal{A}\rangle$ is a negative witness for G (Definition 4), and for each $e \in \overline{E}$ such that $\langle \rightarrow, e|\hat{w}_\mathcal{A}\rangle \neq 0$, $|\hat{w}^e_\mathcal{A}\rangle$ is a negative witness for G^e, then*

  $$|\hat{w}^\circ_\mathcal{A}\rangle = \sum_{e\in\overline{E}} \langle \rightarrow, e|\hat{w}_\mathcal{A}\rangle |\hat{w}^e_\mathcal{A}\rangle + \Pi_{E\setminus\overline{E}}|\hat{w}_\mathcal{A}\rangle$$

  is a negative witness for G°.

Informally, we obtain G° from G by, for each $e \in \overline{E}$, removing the edge e, and identifying its endpoints with the vertices $s = s^e$ and $t = t^e$ of the graph G^e. This is illustrated in Figure 4. The subspaces associated with G° are mostly inherited from G and G^e, except for those on the glued boundaries. There, intuitively, we replace $|\rightarrow, e\rangle$ with the edges incident to $s = s^e$ in G^e, and $|\leftarrow, e\rangle$ with the edges incident to $t = t^e$ in G^e.

Note that the witnesses in the composed subspace graph have a very logical form. We obtain a positive witness $|\hat{w}^\circ\rangle$ by taking a positive witness for G, and replacing the edge $|\rightarrow, e\rangle + |\leftarrow, e\rangle$ with a positive witness, and similarly for negative witnesses.

Next, we apply the composition construction of Theorem 14 to compose the switching networks for AND and OR in Section 2.2.2 and Section 2.2.1 with other *st*-composable subspace graphs computing some functions f_σ to get a subspace graph computing the composed function $\varphi \circ (f_\sigma)_{\sigma\in\Sigma}$ when φ is a Boolean formula on $\{0,1\}^\Sigma$. This is the function obtained by computing the value of each variable x_σ of the formula φ as f_σ of some input. Subspace graphs for φ can be obtained simply by composing the switching networks for OR and AND (see [14]). Our composition theorem, Theorem 14, generalizes this simple switching network composition.

A formula is a rooted tree, where each internal node represents an AND or an OR gate, and each leaf represents a literal. We let $\overline{\Sigma} \setminus \Sigma$ denote the set of internal nodes, and for each $\sigma \in \overline{\Sigma} \setminus \Sigma$, d_σ is the number of children of σ.

A family of formulas is balanced if there is a constant c such that for every internal node σ, if its subtree has N leaves, then the sub-tree of each of its d_σ children has at most cN/d_σ leaves.

▶ **Lemma 15.** *Let φ be a balanced formula on $\{0,1\}^\Sigma$. Let $\{f_\sigma\}_{\sigma\in\Sigma}$ be Boolean functions, and for each $\sigma \in \Sigma$, let G_σ be an st-composable subspace graph (Definition 10) computing f_σ, with working bases that can be generated in time at most T, and $\log\dim H_{G_\sigma} \leq S$.*

For each $\sigma \in \Sigma$, let C_σ be a known upper bound on $\hat{\mathcal{C}}(G_\sigma)$. Then there is an st-composable subspace graph G° computing $\varphi \circ (f_\sigma)_{\sigma\in\Sigma}$ with $\log\dim H_{G^\circ} = S + O(\log|\Sigma|)$ and

$$\hat{\mathcal{C}}(G^\circ)^2 \leq \sum_{\sigma\in\Sigma} \mathsf{C}_\sigma^2,$$

as long as for each $\sigma \in \overline{\Sigma}\backslash\Sigma$, a superposition proportional to $\sum_{i\in[d_\sigma]}\sqrt{\mathsf{C}^{\pm}_{\sigma,i}}|i\rangle$ can be generated in $O(\log d_\sigma)$ complexity, for certain values $\mathsf{C}^{\pm}_{\sigma,i}$ related to the bounds $\{\mathsf{C}_\sigma\}_{\sigma\in\Sigma}$. Furthermore, the working bases of G° can be generated in time $T + O(\log|\Sigma|)$.

2.4 Quantum Divide & Conquer

In this section, we state our time-efficient quantum divide & conquer results. The following is a time-efficient version of Strategy 1 in [9], restricted to symmetric formulas.

▶ **Theorem 16.** *Fix a function family $f_{\ell,n} : D_{\ell,n} \to \{0,1\}$. Fix unit-time-computable functions $\lambda_1, \lambda_2 : \mathbb{N} \to \mathbb{N}$, and a formula φ on $\{0,1\}^{a+1}$ such that $\varphi = \varphi'(z_1,\dots,z_a) \vee z_{a+1}$ for some symmetric formula φ'. Suppose $\{\mathcal{P}_{\mathrm{aux},\ell,n}\}_{\ell,n\in\mathbb{N}}$ is a family of quantum algorithms such that:*

- *$\mathcal{P}_{\mathrm{aux},\ell,n}$ decides some $f_{\mathrm{aux},\ell,n} : D_{\ell,n} \to \{0,1\}$ with time and space complexities $T_{\mathrm{aux}}(\ell,n)$ and $S_{\mathrm{aux}}(\ell,n)$;*
- *if $\ell \le \ell_0$, $f_{\ell,n}(x) = f_{\mathrm{aux},\ell,n}(x)$;*
- *if $\ell > \ell_0$, $f_{\ell,n}(x) = \varphi \circ (f_i)_{i\in[a+1]}$ where each f_i for $i \in [a]$ is such that $f_i(x) = f_{\lambda_1(\ell),\lambda_2(n)}(x^i)$ for some unit-time-computable instance x^i of $f_{\lambda_1(\ell),\lambda_2(n)}$, and $f_{a+1} = f_{\mathrm{aux},\ell,n}$.*

Then there is a bounded-error quantum algorithm that decides $f_{\ell,n}$ with time complexity $\widetilde{O}(T(\ell,n))$, and space complexity $O(S_{\mathrm{aux}}(\ell,n) + \log T(\ell,n))$, where for all $\ell > \ell_0$:

$$T(\ell,n) := \sqrt{aT(\lambda_1(\ell),\lambda_2(n))^2 + 4T_{\mathrm{aux}}(\ell,n)^2} \le \sqrt{a}T(\lambda_1(\ell),\lambda_2(n)) + 2T_{\mathrm{aux}}(\ell,n)$$

and for $\ell \le \ell_0$, $T(\ell,n) = 2T_{\mathrm{aux}}(\ell,n)$.

To prove Theorem 16, we make a subspace graph that computes $f_{\ell,n} = \varphi \circ (f_i)_i$ using Lemma 15, which allows us to combine subspace graphs for $f_{\lambda_1(\ell),\lambda_2(n)}$ – built up inductively – and $f_{\mathrm{aux},\ell,n}$ – built by turning the quantum algorithm $\mathcal{P}_{\mathrm{aux},\ell,n}$ into a subspace graph using Lemma 13. Note that Lemma 15 applies to any balanced formula φ, but in the special case we consider in Theorem 16, the values $\mathsf{C}^{\pm}_{\sigma,i}$ referred to in Lemma 15 are well behaved.

2.5 Application to DSTCON

In this section we consider the *directed st-connectivity* problem.

▶ **Problem 17** (DSTCON). *Given access to a directed graph $G = (V,E)$ via the oracle $\mathcal{O}_G$, and two vertices $s,t \in V$, decide whether there is a directed path from s to t in G.*

There is a classical recursive algorithm for DSTCON that operates in the low-space regime due to Savitch [22] that decides DSTCON in time $O((2n)^{\log n} = 2^{\log^2 + O(\log n)})$ and space $O(\log^2 n)$. The algorithm recursively calls a subroutine that decides a function $f_{\ell,n}(G,u,v)$, which is 1 if and only if there is a path of length at most ℓ from u to v in G. We can express $f_{\ell,n}$ recursively, in terms of a symmetric formula φ' in $a = 2n$ variables:

$$f_{\ell,n}(G,u,v) = \underbrace{\bigvee_{w\in V}\left(f_{\ell/2,n}(G,u,w) \wedge f_{\ell/2,n}(G,w,v)\right)}_{\varphi'}.$$

We show a quantum speedup for Savitch's algorithm via application of Theorem 16, which gives us the recursion

$$T(\ell,n) = \sqrt{2n}T(\ell/2,n) + O(1)$$

from which we get:

$$T(n,n) = \sqrt{2n}^{\log n} + O(\log n) = 2^{\frac{1}{2}\log^2(n)+O(\log n)},$$

which is the complexity of computing $f_{n,n}(G,s,t) = \textsc{dstcon}(G)$. This yields the following.

▶ **Theorem 18.** *Let $G = (V,E)$ be a directed graph, $|V| = n$. Then there exists a recursive quantum algorithm that decides* DSTCON *on G with bounded error in time $\widetilde{O}((\sqrt{2n})^{\log n}) = 2^{\frac{1}{2}\log^2 n+O(\log n)}$ and space $O(\log^2 n)$.*

References

1 Jonathan Allcock, Jinge Bao, Aleksandrs Belovs, Troy Lee, and Miklos Santha. On the quantum time complexity of divide and conquer. `arXiv:2311.16401`, 2023.

2 Andris Ambainis. Quantum search with variable times. *Theory of Computing Systems*, 47:786–807, 2010. `arXiv:quant-ph/0609168` `doi:10.1007/s00224-009-9219-1`.

3 Simon Apers, Stacey Jeffery, Galina Pass, and Michael Walter. (No) Quantum Space-Time Tradeoff for USTCON. In *31st Annual European Symposium on Algorithms (ESA 2023)*, pages 10:1–10:17, 2023. `doi:10.4230/LIPIcs.ESA.2023.10`.

4 Aleksandrs Belovs. Quantum walks and electric networks. `arXiv:1302.3143`, 2013.

5 Aleksandrs Belovs, Andrew M. Childs, Stacey Jeffery, Robin Kothari, and Frédéric Magniez. Time-efficient quantum walks for 3-distinctness. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 105–122, 2013. `doi:10.1007/978-3-642-39206-1_10`.

6 Aleksandrs Belovs, Stacey Jeffery, and Duyal Yolcu. Taming quantum time complexity. `arXiv:2311.15873`, 2023. `doi:10.48550/arXiv.2311.15873`.

7 Aleksandrs Blovs and Ben W. Reichardt. Span programs and quantum algorithms for st-connectivity and claw detection. In *Proceedings of the 20th Annual European Symposium on Algorithms (ESA)*, pages 193–204, 2012. `doi:10.1007/978-3-642-33090-2_18`.

8 M. L. Bonet and S. R. Buss. Size-depth tradeoffs for boolean formulae. *Information Processing Letters*, 49:151–155, 1994. `doi:10.1016/0020-0190(94)90093-0`.

9 Andrew M. Childs, Robin Kothari, Matt Kovacs-Deak, Aarthi Sundaram, and Daochen Wang. Quantum divide and conquer. `arXiv:2210.06419`, 2022.

10 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. Earlier version in ICALP'04. `arXiv:quant-ph/0401091` `doi:10.1137/050644719`.

11 Tsuyoshi Ito and Stacey Jeffery. Approximate span programs. *Algorithmica*, 79:2158–2195, 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 *Proceedings of the 26th Annual European Symposium on Algorithms (ESA)*, pages 49:1–49:13, 2018. `doi:10.4230/LIPIcs.ESA.2018.49`.

13 Stacey Jeffery. Quantum subroutine composition. `arXiv:2209.14146`, 2022.

14 Stacey Jeffery and Shelby Kimmel. Quantum algorithms for graph connectivity and formula evaluation. *Quantum*, 1(26), 2017.

15 Stacey Jeffery and Sebastian Zur. Multidimensional quantum walks and application to k-distinctness. In *Proceedings of the 55th ACM Symposium on the Theory of Computing (STOC)*, pages 1125–1130, 2023. `arXiv:2208.13492`

16 C. Y. Lee. Representation of switching functions by binary decision programs. *Bell Systems Technical Journal*, 38(4):985–999, 1959. `doi:10.1002/j.1538-7305.1959.tb01585.x`.

17 Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011. Earlier version in STOC'07. `arXiv:quant-ph/0608026` `doi:10.1137/090745854`.

18 Aaron H. Potechin. *Analyzing monotone space complexity via the switching network model.* PhD thesis, Massachusetts Institute of Technology, 2015.

19 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 IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 544–551, 2009. `arXiv:0904.2759` `doi:10.1109/FOCS.2009.55`.

20 Ben W. Reichardt. Faster quantum algorithm for evaluating game trees. In *Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 546–559, 2011. `doi:10.5555/2133036.2133079`.

21 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`.

22 Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970. `doi:10.1016/S0022-0000(70)80006-X`.

23 Claude E. Shannon. A symbolic analysis of relay and switching networks. *Transactions of the American Institute of Electrical Engineers*, 57(12):713–723, 1938. `doi:10.1109/T-AIEE.1938.5057767`.

24 Claude E. Shannon. The synthesis of two-terminal switching circuits. *Bell System Technical Journal*, 28(1):59–98, 1949. `doi:10.1002/j.1538-7305.1949.tb03624.x`.

25 Mario Szegedy. Quantum speed-up of Markov chain based algorithms. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 32–41, 2004. `arXiv:quant-ph/0401053` `doi:10.1109/FOCS.2004.53`.