Week 4: Span Programs

Stacey Jeffery

February 23, 2025

4.1 Introduction

Branching programs are a classical model of computation, in which a computation is represented by a directed acyclic graph with a source (only has outgoing edges) s and a pair of sinks (only has incoming edges) t_0 and t_1. Each non-sink node has an associated index $i \in [n]$, and two outgoing edges, labelled by 0 and 1. For any $x \in \{0,1\}^n$, you can compute the branching program's value on input x by starting from s, and whenever you're at a vertex v with associated variable i, travel down the out-edge labelled by the bit x_i, until you eventually reach t_{b_x} for some $b_x \in \{0,1\}$. We say the branching program computes a function $\mathrm{F} : \{0,1\}^n \to \{0,1\}$ if $b_x = \mathrm{F}(x)$ for all x.

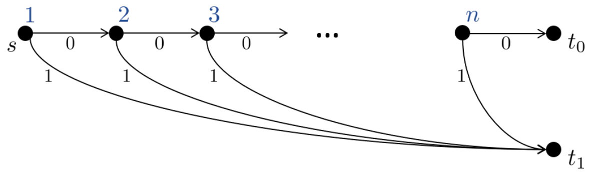


Figure 4.1: A branching program for $\textsc{Or}_n$.

For later analogy, it is useful to think of each edge of a branching program as being labelled by a pair $(i,b) \in [n] \times \{0,1\}$, and then we can let $E(x)$ be the set of edges of the graph labelled by (i, x_i) for some $i \in [n]$. Then we can say that x is "accepted" by the span program if and only if there is a path from s to t_1 in the subgraph consisting of only those edges in $E(x)$.

This is a good model for classical determinstic algorithms (there is also a randomized version of the model) because the number of input queries used to compute $\mathrm{F}(x)$ is nicely captured by the length of the longest path from s to $t_{\mathrm{F}(x)}$ for any x, and the amount of memory needed to "run" a branching program is the logarithm of the number of vertices – the number of bits needed to write down the label of a vertex. Because of this, branching programs have been used to design algorithms for certain problems and show lower bounds on the time and space needed to solve certain problems. However, more than that, branching programs have been used to prove statements about classical deterministic algorithms *in general*. For example, the famous *Barrington's theorem* uses branching programs to show how to convert *any* classical circuit of depth d to a computation that uses $O(d)$ bits of memory – of which all but 3 bits are only used to encode a counter that starts in the all 0s state, and can only be read and incremented – and time $O(4^d)$.

We're not going to talk about branching programs, but they are a useful analogy to motivate us as we wade into the weeds of *span programs* this week. Like branching programs for classical computation, span programs are a model of computation that can be used to design quantum algorithms, or undersand their general properties. As we will see shortly, span programs are fully general for (bounded-error) quantum algorithms, meaning that without loss of generality, we can always assume any bounded-error quantum algorithm for a decision problem is actually a span program. Said another way: there is a correspondance between span programs and bounded-error quantum algorithms that captures the algorithm's query complexity, time complexity, and space complexity.

Span programs were first defined in the context of classical complexity theory [KW93], where they were studied over various fields (we will study them over $\mathbb{R}$ or $\mathbb{C}$, which turn out to be equivalent). Years later, seemingly unrelated, Ambainis introduced a lower bound technique for quantum query complexity called the adversary bound [?], which was generalized [HLŠ07] to its modern form (called the *negative weights* adversary at the time, but now usually just known as the adversary bound). To prove a lower bound on $\mathsf{Q}(\mathrm{F})$ for a fixed F, one must exhibit an *adversary matrix* satisfying certain *semidefinite* constraints.

From the other side, Reichardt and Špalek [RŠ12] first connected span programs to quantum algorithms by showing how a span program could be turned into a quantum algorithm, and later Reichardt showed that the optimization problem of finding an optimal span program (of a certain *canonical* form) for some F is precisely the dual of the optimization problem for finding the best lower bound via an adversary matrix [Rei09]. This showed that both are tight: (1) the adversary bound can be used to prove tight lower bounds on the quantum query complexity of any problem; and (2) span programs can be used to give query-optimal quantum algorithms for any problem.

4.2 Span Programs

A span program is defined as follows.

Definition 4.2.1 (Span Program). *A* span program on $\{0,1\}^n$ *consists of a vector space V over $\mathbb{C}$, a special* target vector $|\tau\rangle \in V$*, and a set of vectors in V, each labelled by some $(i,b) \in [n] \times \{0,1\}$:*

$$\{|v_{i,b,\ell}\rangle : i \in [n], b \in \{0,1\}, \ell \in [d_{i,b}]\}.$$

Letting

$$V(x) = \mathrm{span}\{|v_{i,x_i,\ell}\rangle : i \in [n], \ell \in [d_{i,x_i}]\},$$

we say that the span program accepts $x \in \{0,1\}$ *if $|\tau\rangle \in V(x)$. We say the span program* decides a function $\mathrm{F} : D \to \{0,1\}$ *for $D \subseteq \{0,1\}^n$ if it accepts $x \in D$ if and only if* $\mathrm{F}(x) = 1$.

Example 4.2.2 (Span Program for OR). Let $\textsc{Or}_n : \{0,1\}^n \to \{0,1\}$ be the function that takes value 0 on the all-0s string, and 1 on all other strings. We will define a span program that evaluates $\textsc{Or}_n$. Let $V = \mathbb{C}$, and $|\tau\rangle = 1$. For $i \in [n]$, let $d_{i,0} = 0$, $d_{i,1} = 1$ and $|v_{i,1,1}\rangle = |v_{i,1}\rangle = 1$. Then $V(x) = \mathrm{span}\{|v_{i,1}\rangle : x_i = 1\}$. If x is the all 0s string, $V(x) = \{0\}$. Otherwise, $V(x) = \mathrm{span}\{1\} = \mathbb{C}$. Thus, this span program accepts all strings except the all-0s string, so it evaluates $\textsc{Or}_n$.

For easily reasoning about span programs as mathematical objects, going forward it will be useful to think of a span program as a matrix A with the vectors $\{|v_{i,b,\ell}\rangle\}$ as its columns. Thus, we will consider a span program to be characterized by the parameters $(H, V, A, |\tau\rangle)$, where:

- H is a finite-dimensional inner product space that can be decomposed into pairwise orthogonal subspaces: $H = \bigoplus_{i\in[n]}(H_{i,0} \oplus H_{i,1}) \oplus (H_{\mathrm{true}} \oplus H_{\mathrm{false}})$ over $\mathbb{C}$. We can think of $H_{i,b}$ as $\mathrm{span}\{|i,b,\ell\rangle : \ell \in [d_{i,b}]\}$. The spaces H_{true} and H_{false} are not included in Definition 4.2.1, but we can add them without really changing the definition by pretending every string has an $n+1$-th bit set to 1, and $H_{\mathrm{true}} = H_{n+1,1}$, $H_{\mathrm{false}} = H_{n+1,0}$. We let $H(x) = \bigoplus_{i=1}^{n} H_{i,x_i} \oplus H_{\mathrm{true}}$.
- V is a vector space over $\mathbb{C}$, and $|\tau\rangle \in V$ is the target vector.
- A is a linear map $H \to V$, with $A = \sum_{i\in[n+1],b\in\{0,1\},\ell\in[d_{i,b}]} |v_{i,b,\ell}\rangle\langle i,b,\ell|$ in the notation of Definition 4.2.1.

In the above notation, a span program decides $\mathrm{F} : D \to \{0,1\}$ for $D \subseteq \{0,1\}^n$ if for every $x \in D$,

$$\mathrm{F}(x) = 1 \Leftrightarrow |\tau\rangle \in A(H(x)). \tag{4.1}$$

[2]

An input x such that $|\tau\rangle \in A(H(x))$ is accepted by the span program, and all other inputs are *rejected.*

Example 4.2.3 (Span Program for $\textsc{Or}_n$)**.** We will define a span program that evaluates $\textsc{Or}_n$ in the new notation. For $i \in [n]$, let $H_{i,0} = \{0\}$, and $H_{i,1} = H_i = \mathrm{span}\{|i\rangle\}$ (importantly, these are all orthogonal), so $H = \mathrm{span}\{|i\rangle : i \in [n]\}$. Let $V = \mathbb{C}$, and $|\tau\rangle = 1$. Let $A = \sum_{i=1}^{n} \langle i|$.

Note that $H(x) = \mathrm{span}\{|i\rangle : x_i = 1\}$, which is non-trivial as long as x is not the all-0s string. Thus, as long as x is not the all-0s string, $H(x)$ contains some $|i\rangle$, and A maps this to $|\tau\rangle$. Thus, this span program accepts all strings except the all-0s string, so it evaluates $\textsc{Or}_n$.

To see that the span program in Example 4.2.3 decides $\textsc{Or}$, we found, for each $x \in f^{-1}(1)$, a vector $|i\rangle \in H(x)$ that maps to $|\tau\rangle$ under A, which "witnesses" that $|\tau\rangle \in A(H(x))$. We also had to argue that there can never exist such a vector whenever $x \in f^{-1}(0)$. One way of doing that is to exhibit a *negative witness*, defined below.

Definition 4.2.4 (Span Program Witnesses and Witness Complexity)**.** *Let $P = (H, V, A, |\tau\rangle)$ be a span program on $\{0,1\}^n$, and $x \in \{0,1\}^n$. A* positive witness *for x in P is a vector $|w\rangle \in H(x)$ such that $A|w\rangle = |\tau\rangle$. The* positive witness complexity *of x (in P) is defined*

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

A negative witness *for x in P is a vector $|\omega\rangle \in V$ such that $(\langle\omega|A)^\dagger \in H(x)^\perp$ (equivalently $\langle\omega|A\Pi_{H(x)} = 0$) and $\langle\omega|\tau\rangle = 1$. The* negative witness complexity *of x (in P) is defined*

$$w_-(x) = w_-(x, P) = \min\{\|\langle\omega|A\|^2 : (\langle\omega|A)^\dagger \in H(x)^\perp, \langle\omega|\tau\rangle = 1\}.$$

The positive and negative complexities *of the span program are defined:*

$$\mathcal{W}_+(P) = \max_{x \in \mathrm{F}^{-1}(1)} w_+(x, P) \qquad \textit{and} \qquad \mathcal{W}_-(P) = \max_{x \in \mathrm{F}^{-1}(0)} w_-(x, P)$$

where F *is the unique total function decided by the span program.*[1] *The* complexity *of the span program is*

$$\mathcal{C}(P) = \sqrt{\mathcal{W}_+(P)\mathcal{W}_-(P)}.$$

Exercise 4.2.1. *Show that there exists a negative witness for x if and only if there is no positive witness for x.*

The definition of negative witnesses may appear puzzling at first, so we make some remarks. Note that to witness that x is rejected, what we actually require is that $\langle\omega|\tau\rangle \neq 0$, however, then any multiple of a negative witness would also be a negative witness, and so the negative witness complexity would not be defined. Thus, we fix $\langle\omega|\tau\rangle$ to be 1. Also note that the negative witness complexity could not have been defined with $\||\omega\rangle\|^2 = \|\langle\omega|\|^2$ instead of $\|\langle\omega|A\|^2$. First: V is an unstructured space, so it has no specified norm. We could, of course, specify a norm for V, by fixing a basis, but scaling the vectors in that basis by arbitrary non-zero scalars would allow one to change the values of $\||\omega\rangle\|^2$ arbitrarily. That is to say: any norm in V is meaningless, and so any meaningful measure of complexity should instead use the norm in H.

[1]The span program also decides *partial* functions defined only on $D \subset \{0,1\}^n$ that agree with F on D. Technically we can define $\mathcal{W}_+$ and $\mathcal{W}_-$ with respect to any such function, which might be smaller if the x with largest witness sizes are not in D.

You may be wondering if the notion of positive and negative witnesses for span programs are related to the positive and negative witnesses introduced in in the context of Phase Estimation Algorithms. In Exercise 4.2.3, you will answer this question in the affirmative.

The complexity of witnesses should be thought of, informally, as representing the difficulty that the span program has in determining that x is accepted, or rejected, as the case may be. Hopefully the following example is helpful in illustrating this.

Example 4.2.5 (Span Program for $\textsc{Or}_n$, Witnesses)**.** Recall that in the span program in Example 4.2.3, we saw that whenever $x_i = 1$, $|i\rangle \in H(x)$. Then $|i\rangle$ is a positive witness for x, since $A|i\rangle = 1 = |\tau\rangle$. This makes, intuitive sense, since an i such that $x_i = 1$ is exactly what's needed for $\textsc{Or}_n(x) = 1$. Since $\||i\rangle\|^2 = 1$, this already tells us that $w_+(x) \leq 1$. Can we do better? It turns out we can, when there are multiple indices i such that $x_i = 1$. In that case, a positive witness, which turns out to be optimal (try proving this) is $\sum_{i:x_i=1} \frac{1}{|x|}|i\rangle$, where $|x| = |\{i : x_i = 1\}|$ is the Hamming weight of x. Thus, $w_+(x) = \sum_{i:x_i=1} \frac{1}{|x|^2} = 1/|x|$. Thus, the positive witnesses are smaller when x has more 1s, representing an "easier" instance. We still have $\mathcal{W}_+ = \max_{x:\textsc{Or}_n(x)=1} w_+(x) = 1$, but if we are promised that the Hamming weight of positive instances is at least some threshold, t, then this can be improved to $\mathcal{W}_+ = 1/t$, reflecting that the problem becomes easier as t increases.

What about negative witnesses? There is only one negative input: the all-0s string. When x is the all-0s string, $H(x) = \{0\}$. In that case, $1 \in V$ is a negative witness: it has overlap 1 with $|\tau\rangle = 1$, and $1A\Pi_{H(x)} = 0$, since $\Pi_{H(x)} = 0$. From this we see that $\mathcal{W}_- \leq \|1A\|^2 = \|\sum_{i=1}^n \langle i|\|^2 = n$.

Exercise 4.2.2 (Span Program for $\textsc{And}_n$)**.** *Let $\textsc{And}_n : \{0,1\}^n \to \{0,1\}$ be the function that takes value 1 on the all-1s string, and 0 on all other strings. Give a span program that evaluates $\textsc{And}_n$ with complexity $\mathcal{C} \leq \sqrt{n}$, and prove that it does indeed evaluate $\textsc{And}_n$ with this complexity.*

Now you know that a span "program" can "decide" a function $\mathrm{F} : \{0,1\}^n \to \{0,1\}$, but you are likely wondering what that actually means. Is there some kind of realistic machine that can run a span program on an input x and output $\mathrm{F}(x)$? It turns out that one such machine is a quantum computer – specifically, a quantum computer running a 0-Phase Estimation Algorithm (as in Theorem 2.2.9), and the query complexity of this algorithm will be $O(\mathcal{C}(P))$. It also turns out that this particular type of quantum algorithm can compute any $\mathrm{F} : \{0,1\}^n \to \{0,1\}$ (or partial function F) with optimal query complexity. We have the following.

Theorem 4.2.6 (Span Programs and Quantum Algorithms)**.** *Let $\mathsf{SPC}(\mathrm{F})$ be the minimum $\mathcal{C}(P)$ of any span program that decides F. Then for any problem F, $\mathsf{SPC}(\mathrm{F}) = \Theta(\mathsf{Q}(\mathrm{F}))$, where $\mathsf{Q}(\mathrm{F})$ is the bounded-error quantum query complexity of F.*

This correspondance is especially remarkable considering that span programs were first introduced in classical complexity theory with no intended connection to quantum computing.

In the following exercise, you will prove one direction of Theorem 4.2.6.

Exercise 4.2.3 (Span Programs and Phase Estimation Algorithms)**.** *Let $P = (H, V, A, |\tau\rangle)$ be a span program. Let $\mathcal{A} = H(x)$ and $\mathcal{B} = \ker(A)$. Let $|w_0\rangle = A^+|\tau\rangle$, where A^+ is the* pseudoinverse *of A – see Appendix .1. You may assume that $\||w_0\rangle\| = 1$, and $\mathcal{W}_-(P)$ is bounded from above by a constant, because it is always possible to ensure this without changing $\mathcal{C}(P)$ by more than a constant factor (you do not need to show this, but it's a nice exercise). Let $|\psi_0\rangle = |w_0\rangle$.*

1. *Let $\Psi^{\mathcal{A}}$ and $\Psi^{\mathcal{B}}$ be orthonormal bases for $\mathcal{A}$ and $\mathcal{B}$, respectively. Show that $|\psi_0\rangle \in \mathcal{B}^\perp$, proving that $\Phi = (H, |\psi_0\rangle, \Psi^{\mathcal{A}}, \Psi^{\mathcal{B}})$ are parameters of a 0-Phase Estimation Algorithm as in Theorem 2.2.9.*

2. *Show that $|w\rangle$ is a positive witness for x in P if and only if $|w_{\mathcal{A}}\rangle = |w\rangle$ and $|w_{\mathcal{B}}\rangle = |w_0\rangle - |w\rangle$ form a negative witness (Definition 2.2.6) for the Phase Estimation Algorithm with parameters*

Φ. (Hint: What is A^+A?).

3. *Show that $a|\omega\rangle$ is a negative witness for x in P for some scalar a, if and only if $(\langle\omega|A)^\dagger$ is a positive witness (Definition 2.2.4) for the 0-Phase Estimation Algorithm with parameters Φ.*

4. *Show that $U = (2\Pi_{\mathcal{A}} - I)(2\Pi_{\mathcal{B}} - I)$ can be implemented with $O(1)$ queries to $\mathcal{O}_x$.*

5. *Conclude that if* F *is decided by P, there exists a quantum algorithm that decides* F *with bounded error using $O(\mathcal{C}(P))$ queries to x, meaning that* $\mathsf{Q}(\mathrm{F}) = O(\mathsf{SPC}(\mathrm{F}))$.

Let us mention some further nice correspondances between span programs and quantum algorithms. Query complexity is a nice measure because it is mathematically easy to work with, but it does not always represent the realistic cost of an algorithm, which may use asymptotically more additional gates than it uses queries. There is no nice measure of a span program that corresponds to the time complexity of the function it decides, (the reflection $(2\Pi_{\mathcal{B}} - I)$ used in the 0-Phase Estimation Algorithm in Exercise 4.2.3 costs 0 queries, but in general there is no detail about how to implement it in a small number of elementary gates), and such a measure is probably too much to hope for. However, we have perhaps the next best thing: For any F, there is a span program that can be evaluated in optimal time complexity – this is because any algorithm for F with time complexity T can be turned into a span program that can be evaluated in time complexity $O(T)$ [CJOP20][2]. This doesn't mean it's easy to find a time-efficient implementation of any span program, though in some particularly structured cases, the time complexity can be analyzed.

Finally, in addition to the time or number of queries needed to evaluate F, we are sometimes interested in how much memory is needed. The *space* complexity of F is related to the smallest $\log(\dim H)$ of any span program deciding F [Jef22]. The measure $\dim H$, called the *span program size*, was the measure of span programs first studied, in the classical literature, whereas span program complexity, $\mathcal{C}(P)$, was only introduced later in connection with quantum query complexity.

4.3 Span Program Composition and Formula Evaluation

We have tried to drive home the point that span programs are a model of computation that captures quantum algorithms very well, but why work with span programs instead of a more standard model like quantum circuits? One reason is that span programs can be combined in some very nice ways. Here we will see how this can be used to get a span program that decides a Boolean formula.

Boolean Formulas A (Boolean) formula on $\{0,1\}^N$ is a rooted tree with N leaves, where each non-leaf node is labelled by $\wedge$ (AND) or $\vee$ (OR), and the i-th lead is labelled by a Boolean variable x_i or its negation $\neg x_i = 1 - x_i$. Formulas are similar to Boolean *circuits*, except that we don't allow "fan out", meaning the output of a gate can only be used as input to one gate (i.e. the graph is a tree). We require that all negations happen at the leaves, rather than allowing NOT gates anywhere in the formula. This is without loss of generality, because by *de Morgan's law*, $\neg(x_1 \vee \cdots \vee x_d) = (\neg x_1) \wedge \cdots \wedge (\neg x_d)$ and $\neg(x_1 \wedge \cdots \wedge x_d) = (\neg x_1) \vee \cdots \vee (\neg x_d)$.

Equivalently, formulas can be defined recursively as follows. $\varphi(x_1) = x_1$ and $\varphi(x_1) = \neg x_1$ are formulas on $\{0,1\}$ of *depth* $D = 0$. For d a positive integer, and $\{\varphi_i\}_{i=1}^d$ formulas on $\{0,1\}^{N_i}$ whose maximum depth is $D-1$, $\varphi_1(x^{(1)}) \vee \cdots \vee \varphi_d(x^{(d)})$ and $\varphi_1(x^{(1)}) \wedge \cdots \wedge \varphi_d(x^{(d)})$ are both formulas on $\{0,1\}^{N_1+\cdots+N_d}$ of depth D. A formula on $\{0,1\}^N$ defines a function $\{0,1\}^N \to \{0,1\}$, which we also refer to as φ, but the formula is not merely described by this function, as a function is generally described by more than one formula. For a family of formulas $\varphi = \{\varphi_N\}_N$, we can define the problem of evaluating it:

[2]The resulting span program does not decide F, but instead *approximately decides* it – a notion we will not define.

Problem: $\textsc{Eval}_\varphi$

Input: $x \in \{0,1\}^N$
Output: $\varphi(x)$

For example, if $\varphi(x) = x_1 \vee \cdots \vee x_n$, then $\textsc{Eval}_\varphi$ is exactly the n-bit OR problem $\textsc{Or}_n$. In this section, we will describe how to construct a span program that evaluates any formula φ. For example, if $\varphi(x_1) = x_1$, then the following span program computes φ:

$$\begin{aligned} H = H_{1,1} &= \mathbb{C}, \quad H_{1,0} = \{0\} \\ V &= \mathbb{C}, \quad |\tau\rangle = 1 \\ A &= I_1, \end{aligned} \tag{4.2}$$

where I_1 is the identity on $\mathbb{C}$. When $\varphi(x_1) = \neg x_1$, the following span program computes φ:

$$\begin{aligned} H' = H'_{1,0} &= \mathbb{C}, \quad H'_{1,1} = \{0\} \\ V' &= \mathbb{C}, \quad |\tau'\rangle = 1 \\ A' &= I_1. \end{aligned} \tag{4.3}$$

Exercise 4.3.1. *Show that the span program in (4.2) computes the one-bit function $\varphi(x_1) = x_1$, and the span program in (4.3) computes the one-bit function $\varphi(x_1) = \neg x_1$. Show that the complexity of both span programs is $\mathcal{C}(P) \leq 1$.*

We will use the span programs in (4.2) and (4.3) as a base case, and show how to combine them to get span programs for any Boolean formula. Specifically, we will show that for any span programs $P_1, \dots, P_d$ computing functions $\{f_i : \{0,1\}^{k_i} \to \{0,1\}\}_{i=1}^d$, we can make a span program computing the function $f_1 \vee \cdots \vee f_d : \{0,1\}^{k_1+\cdots+k_d} \to \{0,1\}$ defined $(x^{(1)}, \dots, x^{(d)}) \mapsto f_1(x^{(1)}) \vee \cdots \vee f_d(x^{(d)})$, where for each $i \in [d]$, $x^{(i)} \in \{0,1\}^{k_i}$. We will show a similar construction replacing $\vee$ with $\wedge$. These composition results will allow us to inductively combine formulas of depth $D-1$ to get formulas of depth D.

4.3.1 OR Composition

Fix d span programs $P_1, \dots, P_d$, with $P_i = (H^{(i)}, V^{(i)}, A^{(i)}, \tau^{(i)})$ deciding $f_i : \{0,1\}^{k_i} \to \{0,1\}$. We will describe a span program P that decides $f_\vee = f_1 \vee \cdots \vee f_k : \{0,1\}^{k_1+\cdots+k_d} \to \{0,1\}$. An input x to $f_\vee$ can be expressed as $x = (x^{(1)}, \dots, x^{(d)})$ where $x^{(i)} = (x_{i,1}, \dots, x_{i,k_i}) \in \{0,1\}^{k_i}$. Thus, we need to define spaces $H_{(i,j),b}$ for $i \in [d]$, $j \in [k_i]$, and $b \in \{0,1\}$:

$$H_{(i,j),b} := \mathrm{span}\{|i\rangle\} \otimes H^{(i)}_{j,b}.$$

Then we have:

$$H = \bigoplus_{i\in[d], j\in[k_i], b\in\{0,1\}} H_{(i,j),b} = \bigoplus_{i\in[d]} \mathrm{span}\{|i\rangle\} \otimes \bigoplus_{j\in[k_i], b\in\{0,1\}} H^{(i)}_{j,b} = \bigoplus_{i\in[d]} \mathrm{span}\{|i\rangle\} \otimes H^{(i)},$$

and similarly

$$H(x) = \bigoplus_{i\in[d]} \mathrm{span}\{|i\rangle\} \otimes H(x^{(i)}).$$

For $i \in [d]$, let $r_i = \dim(V^{(i)})$. Then without loss of generality, we can assume, via isomorphism, that $V^{(i)}$ has orthonormal basis $\{|0\rangle, |i,1\rangle, \dots, |i, r_i - 1\rangle\}$, with $|\tau^{(i)}\rangle = |0\rangle$. Then we let:

$$V = \mathrm{span}\{|0\rangle\} \oplus \mathrm{span}\{|i,\ell\rangle : i \in [d], \ell \in [r_i - 1]\}$$

and $|\tau\rangle = |0\rangle$. Finally, we need to define $A : H \to V$, which we do as follows:

$$A = \sum_{i=1}^{d} \langle i| \otimes A^{(i)}$$

meaning that

$$\forall i \in [d], |h\rangle \in H^{(i)},\ A|i\rangle|h\rangle = A^{(i)}|h\rangle.$$

We now argue that P decides $f_\vee$, and analyze its complexity. First, suppose $f_\vee(x) = 1$. That means that there is at least one value $i \in [d]$ such that $f_i(x^{(i)}) = 1$, meaning there is a positive witness $|w_i\rangle \in H^{(i)}(x^{(i)})$ for $x^{(i)}$ in $P^{(i)}$. We can define $|w\rangle = |i\rangle|w_i\rangle \in H(x)$, and see that $A|w\rangle = A^{(i)}|w_i\rangle = |\tau^{(i)}\rangle = |0\rangle$, since $|w_i\rangle$ is a positive witness in $P^{(i)}$. Thus, x is accepted by P, and its positive witness complexity is at most

$$w_+(x, P) \leq \||w\rangle\|^2 = \||w_i\rangle\|^2 = w_+(x^{(i)}, P^{(i)}),$$

assuming we chose $|w_i\rangle$ to be the optimal positive witness. Thus

$$\mathcal{W}_+(P) \leq \max_{i\in[d]} \mathcal{W}_+(P^{(i)}). \tag{4.4}$$

On the other hand, suppose $f_\vee(x) = 0$, meaning that for all $i \in [d]$, $f_i(x^{(i)}) = 0$, so there exists a negative witness $|\omega_i\rangle$ for $x^{(i)}$ in P_i. Since we have for all $i \in [d]$, $1 = \langle\omega_i|\tau^{(i)}\rangle = \langle\omega_i|0\rangle$, we can write $|\omega_i\rangle = |0\rangle + |v_i\rangle$ for some $|v_i\rangle \in \text{span}\{|i, 1\rangle, \dots, |i, dr_i - 1\rangle\}$. Let $|\omega\rangle = |0\rangle + \sum_{i=1}^{n} |v_i\rangle$. Then clearly $\langle\tau|\omega\rangle = \langle 0|\omega\rangle = 1$. We also have, for any $|i\rangle|h\rangle \in H(x)$, $|h\rangle \in H^{(i)}(x^{(i)})$, so

$$\langle\omega|A|i\rangle|h\rangle = \langle\omega|A^{(i)}|h\rangle = (\langle 0| + \langle v_i|)A^{(i)}|h\rangle.$$

This is because $A^{(i)}$ has columnspace $V^{(i)}$, and $\langle\omega|\Pi_{V^{(i)}} = \langle 0| + \langle v_i|$. But since $|0\rangle + |v_i\rangle = |\omega_i\rangle$ is a negative witness in P_i, and $|h\rangle \in H^{(i)}(x^{(i)})$, this is always 0, so $\langle\omega|A\Pi_{H(x)} = 0$. We thus conclude that $|\omega\rangle$ is a negative witness for x in P. It has complexity:

$$\begin{aligned} w_-(x) \leq \|\langle\omega|A\|^2 &= \left\|(\langle 0| + \langle v_1| + \dots + \langle v_d|)\sum_{i=1}^{d} \langle i| \otimes A^{(i)}\right\|^2 = \sum_{i=1}^{d} \left\|(\langle 0| + \langle v_i|)A^{(i)}\right\|^2 \\ &= \sum_{i=1}^{d} \left\|\langle\omega_i|A^{(i)}\right\|^2 = \sum_{i=1}^{d} w_-(x^{(i)}, P^{(i)}), \end{aligned}$$

assuming we chose all negative witnesses $|\omega_i\rangle$ to be optimal, and thus

$$\mathcal{W}_-(P) \leq \sum_{i=1}^{d} \mathcal{W}_-(P^{(i)}). \tag{4.5}$$

Combining (4.4) and (4.5), we would get complexity $\sqrt{\sum_{i=1}^{d} \mathcal{W}_-(P^{(i)}) \max_{i\in[d]} \mathcal{W}_+(P^{(i)})}$, but it turns out this is not optimal – it is particularly bad when there is a lot of variation in $\mathcal{W}_+(P^{(i)})$ over $i \in [d]$. We can improve this by scaling the span programs $P^{(i)}$. In any span program, but replacing A with αA for some scalar α, you can get a span program P' with

$$\mathcal{W}_+(P') = \frac{1}{|\alpha|^2}\mathcal{W}_+(P), \quad \text{and} \quad \mathcal{W}_-(P') = |\alpha|^2\mathcal{W}(P).$$

This does not change the overall complexity $\mathcal{C}(P)$, but simply shifts it between $\mathcal{W}_+$ and $\mathcal{W}_-$.

Exercise 4.3.2. *By scaling each $A^{(i)}$ in the above construction, describe a span program P' for $f_\vee$ with complexity $\mathcal{C}(P') = \sqrt{\sum_{i=1}^{d} \mathcal{C}(P_i)^2}$.*

4.3.2 AND Composition

Fix d span programs $P_1, \dots, P_d$, with $P_i = (H^{(i)}, V^{(i)}, A^{(i)}, |\tau^{(i)}\rangle)$ deciding $f_i : \{0,1\}^{k_i} \to \{0,1\}$. We will describe a span program P that decides $f_\wedge = f_1 \wedge \dots \wedge f_d : \{0,1\}^{k_1+\dots+k_d} \to \{0,1\}$. As in the case of OR, we can describe an input x to $f_\wedge$ as $x = (x^{(1)}, \dots, x^{(d)})$ where $x^{(i)} = (x_{i,1}, \dots, x_{i,k_i}) \in \{0,1\}^{k_i}$. Thus, we need to define, for each $i \in [d]$, $j \in [k_i]$ and $b \in \{0,1\}$:

$$H_{(i,j),b} := |i\rangle \otimes H^{(i)}_{j,b},$$

which is just like in the OR construction. Thus, as in the OR construction, we get

$$H = \bigoplus_{i\in[d]} \mathrm{span}\{|i\rangle\} \otimes H^{(i)} \quad \text{and} \quad H(x) = \bigoplus_{i\in[d]} \mathrm{span}\{|i\rangle\} \otimes H^{(i)}(x).$$

Next, we define:

$$V := \bigoplus_{i\in[d]} |i\rangle \otimes V^{(i)}, \quad |\tau\rangle := \sum_{i=1}^{d} |i\rangle \otimes |\tau^{(i)}\rangle, \quad \text{and} \quad A := \sum_{i=1}^{d} |i\rangle\langle i| \otimes A^{(i)}.$$

Exercise 4.3.3. *Suppose $f_\wedge(x) = 1$. Describe a positive witness for x in P, and use it to show that $\mathcal{W}_+(P) \le \sum_{i\in[d]} \mathcal{W}_+(P^{(i)})$.*

Exercise 4.3.4. *Suppose $f_\wedge(x) = 0$. Describe a negative witness for x in P, and use it to show that $\mathcal{W}_-(P) \le \max_{i\in[d]} \mathcal{W}_-(P^{(i)})$.*

Just as in Exercise 4.3.2, we can slightly modify the construction of P to get a span program P' as follows.

Lemma 4.3.1. *There exists a span program P' for $f_\wedge$ with complexity $\mathcal{C}(P') \le \sqrt{\sum_{i=1}^{d} \mathcal{C}(P^{(i)})^2}$.*

The proof is left as an exercise.

4.3.3 Formula Evaluation

We now prove the following:

Theorem 4.3.2. *For any family of formulas φ on $\{0,1\}^N$, $\mathsf{Q}(\textsc{Eval}_\varphi) = O(\sqrt{N})$.*

We will show, by induction on $D \ge 0$, that for any formula φ on $\{0,1\}^N$ of depth D, there is a span program P_φ with complexity $\mathcal{C}(P_\varphi) \le \sqrt{N}$, which will suffice to prove Theorem 4.3.2.

Base Case: For the base case, if $D = 0$, then either $\varphi(x_1) = x_1$, in which case the span program in (4.2) computes φ; or $\varphi(x_1) = \neg x_1$, in which case the span program in (4.3) computes φ. By Exercise 4.3.1, the complexity in either case is 1.

For the induction step, suppose φ has depth $D > 0$. There are two cases.

OR Case: If $\varphi(x) = \bigvee_{i=1}^{d} \varphi_i(x^{(i)})$ for some formulas of depth at most $D-1$, then by the induction hypothesis, there are span programs $P_{\varphi_1}, \dots, P_{\varphi_d}$ computing each φ_i with complexity $\mathcal{C}(P_{\varphi_i}) \le \sqrt{N_i}$, where N_i is the number of variables of φ_i. Applying the OR composition construction from Section 4.3.1, we can get a span program P_φ that computes φ with complexity

$$\mathcal{C}(P_\varphi) \le \sqrt{\sum_{i\in[d]} \mathcal{C}(P_i)^2} = \sqrt{\sum_{i\in[d]} N_i} = \sqrt{N},$$

by Exercise 4.3.2.

AND Case: Otherwise, $\varphi(x) = \bigwedge_{i=1}^{d} \varphi_i(x^{(i)})$ for some formulas of depth at most $D-1$. Again, by the induction hypothesis, there are span programs $P_{\varphi_1}, \dots, P_{\varphi_d}$ computing each φ_i with complexity $\mathcal{C}(P_{\varphi_i}) \leq \sqrt{N_i}$, where N_i is the number of variables of φ_i. Applying the AND composition construction from Section 4.3.2, we can get a span program P_φ that computes φ with complexity

$$\mathcal{C}(P_\varphi) \leq \sqrt{\sum_{i\in[d]} \mathcal{C}(P_i)^2} = \sqrt{\sum_{i\in[d]} N_i} = \sqrt{N},$$

by Lemma 4.3.1.

4.3.4 Function Composition

You have no doubt noticed that the span program for OR in Exercise 4.3.2 resembles the construction we use to combine span programs $P_1, \dots, P_d$ in OR composition (and similarly for AND, if you worked out a span program for AND in Exercise 4.2.2). It turns out we can do a similar composition replacing OR or AND with *any* function we have a span program for. Specifically, let $f : \{0,1\}^n \to \{0,1\}$, and $g : \{0,1\}^m \to \{0,1\}$, and define $f \circ g : \{0,1\}^{nm} \to \{0,1\}$ by $f \circ g(x^{(1)}, \dots, x^{(n)}) = f(g(x^{(1)}), \dots, g(x^{(n)}))$. Span programs can be used to show that [Rei09]

$$\mathsf{Q}(f \circ g) = O(\mathsf{Q}(f)\mathsf{Q}(g)). \tag{4.6}$$

Note that if we replace Q with *exact* query complexity, Q_E, then this result becomes obvious. If $\mathcal{P}$ is a quantum algorithm that computes f, and $\mathcal{P}'$ is a quantum algorithm that computes g *with no error*, then we can simply run $\mathcal{P}$, and every time it makes a query to its input, we instantiate that query by running $\mathcal{P}'$. If $\mathcal{P}'$ has bounded error, say $1/3$, this doesn't work, because an expected $1/3$ of the results returned by $\mathcal{P}'$ will be wrong. We can remedy this by amplifying the success probability of $\mathcal{P}'$ to $1 - 1/\mathsf{Q}(f)$, so that with constant probability, none of the $\mathsf{Q}(f)$ calls to $\mathcal{P}'$ is incorrect, but this gives a total query complexity $O(\mathsf{Q}(f)\mathsf{Q}(g)\log \mathsf{Q}(f))$. The fact that we can remove this log factor is quite surprising. The expression in (4.6) also turns out to be tight for all f and g.

4.4 Span Programs for st-Connectivity

In this section, we will see a very nice example of a span program that evaluates a kind of problem called (undirected) st-connectivity. It takes as input a graph G on $V = [n]$, where $s, t \in [n]$ are two fixed vertices (without loss of generality, you can assume $s = 1$ and $t = n$).

Problem: USTCON$_n$

Input: An oracle $\mathcal{O}_G$ that outputs, for any $u, v \in [n]$, a bit indicating if $\{u, v\} \in E(G)$ – called an *adjacency matrix oracle* for G.

Output: A bit indicating whether there is a path from s to t in G.

We can consider a variation of this problem in the *edge-list model* that we saw in the context of random walks, where for each vertex, its neighbours are given as a list that we can query. In that case, it is natural to solve the problem with a quantum walk [AJPW23]. In the *adjacency matrix model*, as defined here, we are not able to efficiently walk on the graph, because even finding neighbours may be difficult. Instead we will give a span program for this problem, and then use Theorem 4.2.6 (or Exercise 4.2.3) to upper bound $\mathsf{Q}(\text{USTCON}_n)$.

For this problem, the "indices" that we query have the form $(u, v) \in [n] \times [n]$, so we need to define $H_{(u,v),b}$ for $b \in \{0,1\}$. In fact, since the graph is undirected, we always have $\mathcal{O}_G(u, v) = \mathcal{O}_G(v, u)$, so we can restrict to queries $(u,v)\in[n]\times[n]$ where $u<v$ ($\mathcal{O}_G(u,u)$ has no bearing on whether s and t are connected). We let

$$H_{(u,v),0}=\{0\}\quad\text{and}\quad H_{(u,v),1}=H_{u,v}=\mathrm{span}\{|u,v\rangle\}$$

for all $u,v\in[n]$ such that $u<v$. We define A by the action

$$A|u,v\rangle=|u\rangle-|v\rangle \tag{4.7}$$

so we are implicitly defining $V=\mathrm{span}\{|u\rangle : u\in[n]\}$. Finally, we let

$$|\tau\rangle=|s\rangle-|t\rangle.$$

This completes the definition of the span program, which we now argue decides USTCON.

Positive Analysis Suppose USTCON$(G)=1$, meaning there is a path from s to t: $s=u_0,\dots,u_\ell=t$ such that for all $i\in[\ell]$, $\{u_{i-1},u_i\}\in E(G)$. Suppose for simplicity that for all i, $u_{i-1}<u_i$. Then

$$|w\rangle:=\sum_{i=1}^{d}|u_{i-1},u_i\rangle\in H(G)=\mathrm{span}\{|u,v\rangle : \{u,v\}\in E(G),u<v\} \tag{4.8}$$

and

$$A|w\rangle=\sum_{i=1}^{d}(|u_{i-1}\rangle-|u_i\rangle)=\sum_{i=0}^{d-1}|u_i\rangle-\sum_{i=1}^{d}|u_i\rangle=|u_0\rangle-|u_d\rangle=|s\rangle-|t\rangle=|\tau\rangle$$

meaning that $|w\rangle$ is a positive witness for G. In the case where there are some values i for which $u_{i-1}>u_i$, we can get the same thing by simply replacing $|u_{i-1},u_i\rangle$ (which is then not in H) with $-|u_i,u_{i-1}\rangle$ (here, just as when we studied quantum walks, a minus sign can be seen as reversing the direction of an edge). This is a logical positive witness, since a path from s to t is precisely what is needed for s and t to be connected (it is a 1-*certificate* for USTCON). That means that if G has an st-path of length ℓ, then

$$w_+(G)\leq\||w\rangle\|^2=\ell.$$

If there is an st-path in G, then there must be one of length at most $n-1$ (i.e., it visits every vertex at most once, so it has at most $n-1$ edges), so we get

$$\mathcal{W}_+\leq n-1. \tag{4.9}$$

Note that for graphs G where s and t are connected by a *shorter* path, we get a smaller witness size. This means that we should think of these as *easier* instances of USTCON, and concretely, if we are promised that the input is such that either s and t are not connected, or they are connected by a path of length at most ℓ, then $\mathcal{W}_+$ and thus the overall complexity will be smaller as ℓ gets smaller. It is somewhat intuitive that it is easier to decide that s and t are connected if the path connecting them is shorter, but there is another factor that might make the problem easier: *many* paths between s and t. It turns out that we can also take advantage of this. If there are multiple paths, we can also get a witness by taking a linear combination of states like the one in (4.8). A linear combination of paths from s to t is exactly an st-flow (Definition 3.6.1).

Exercise 4.4.1. *Let θ be a unit st-flow in G, and define*

$$|w\rangle=\sum_{\{u,v\}\in E(G):u<v}\theta(u,v)|u,v\rangle$$

1. *Show that $|w\rangle$ is a positive witness for G.*

2. *Conclude that $w_+(G)\leq\mathcal{R}_{s,t}(G)$.*

3. *Show that every positive witness for G can be seen as some st-flow, and therefore conclude that $w_+(G)=\mathcal{R}_{s,t}(G)$.*

In the worst case over all input graphs G, (4.9) is still tight, but if we have a *promise* on the input, we can sometimes use Exercise 4.4.1 to do better.

Negative Analysis Suppose $\textsc{Ustcon}(G) = 0$, meaning there is no path from s to t. When s and t are connected, there is an st-path. What do we need to see to convince ourselves that s and t are *not* connected? The dual of a path is a *cut*, which is a set of missing edges in G that separate the vertices into two disconnected sets, one of which contains s and one of which contains t. That is, let $S \subset [n]$ be such that:

- $s \in S$.
- There is no $\{u, v\} \in E(G)$ such that $u \in S$ and $v \in [n] \setminus S$.
- $t \in [n] \setminus S$.

Then we can define a negative witness as follows:

$$|\omega\rangle := \sum_{u \in S} |u\rangle.$$

To prove that this is a negative witness, we need to show that $\langle\omega|A$ has no overlap with $H(G)$, and that $\langle\omega|\tau\rangle = 1$. The latter easily follows from the fact that $|\tau\rangle = |s\rangle - |t\rangle$ and $s \in S$ but $t \notin S$. For the former, we have for any $|u, v\rangle \in H(G)$ (meaning $\{u, v\} \in E(G)$ and $u < v$):

$$\langle\omega|A|u, v\rangle = \langle\omega|u\rangle - \langle\omega|v\rangle.$$

Since $\{u, v\} \in E(G)$, either $u, v \in S$, in which case $\langle\omega|A|u, v\rangle = 1 - 1 = 0$, or $u, v \in [n] \setminus S$, in which case $\langle\omega|A|u, v\rangle = 0 - 0 = 0$. Thus $|\omega\rangle$ is a negative witness, and we can conclude that:

$$\begin{aligned} w_-(G) \leq \|\langle\omega|A\|^2 = \left\| \langle\omega| \sum_{u,v \in [n] : u < v} (|u\rangle - |v\rangle)\langle u, v| \right\|^2 &= \sum_{u,v \in [n] : u < v} |\langle\omega|u\rangle - \langle\omega|v\rangle|^2 \\ = \sum_{u \in S, v \in [n] \setminus S} |1 - 0|^2 = |S|(n - |S|) \leq n^2, \end{aligned} \tag{4.10}$$

so

$$\mathcal{W}_-(G) \leq n^2. \tag{4.11}$$

Conclusion From (4.9) and (4.11), we have

$$\mathcal{C}(P) = \sqrt{\mathcal{W}_+ \mathcal{W}_-} \leq \sqrt{n^3} = n^{1.5},$$

from which we can conclude that

$$\mathsf{Q}(\textsc{Ustcon}) = O(n^{1.5}).$$

As mentioned, we can do better in the case that there is a promise on the input, such as the promise that if there is a path from s to t, there is one of length at most ℓ, or if there is a path from s to t, then the effective resistance is upper bounded by some known quantity. In fact, a particular kind of promise might be that G is always a subgraph of some fixed $\mathcal{G}$, meaning that instead of querying the adjacency matrix, which is like a string with indices $(u, v) \in [n] \times [n] : u < v$, we can query a string indexed by the edges of $\mathcal{G}$ to ask if they are also present in G (as we are promised that no other edges are present). This gives us something called a *switching network*.

4.4.1 Switching Networks

A switching network is a graph $\mathcal{G}$ on n vertices, two of which are s and t, 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, a switching network has an associated function F that is 1 if and only if s and t are connected by a path of "on" edges.

Formally, for every $e \in E(\mathcal{G})$, there is a literal $\varphi_e(x) = x_i$ or $\varphi_e(x) = \neg x_i$ for some $i \in [N]$. Define $\mathcal{G}(x)$ to be the subgraph of $\mathcal{G}$ that includes the edge e if and only if $\varphi_e(x) = 1$. Then we can define a span program for $\mathsf{F} : \{0,1\}^N \to \{0,1\}$ from any switching network for F as follows. Let $\mathsf{w} : E(\mathcal{G}) \to \mathbb{R}_{>0}$ be any weight function. This is not part of the input, as the weights have no bearing on whether or not s and t are connected, but the weights are parameters of the span program that will impact its complexity. Define for every $i \in [N]$:

$$\begin{aligned} H_{i,0} &= \mathrm{span}\{|u,v\rangle : \{u,v\} \in E(\mathcal{G}), u < v, \varphi_{u,v}(x) = \neg x_i\} \\ \text{and} \quad H_{i,1} &= \mathrm{span}\{|u,v\rangle : \{u,v\} \in E(\mathcal{G}), u < v, \varphi_{u,v}(x) = x_i\}. \end{aligned} \tag{4.12}$$

Then

$$H(x) = \mathrm{span}\{|u,v\rangle : \{u,v\} \in E(\mathcal{G}(x)), u < v\}.$$

We generalize A from (4.7) using the weight function w, defining, for all $\{u,v\} \in E(\mathcal{G})$ such that $u < v$

$$A|u,v\rangle = \sqrt{\mathsf{w}_{u,v}}(|u\rangle - |v\rangle), \tag{4.13}$$

so as before we have $V = \mathrm{span}\{|u\rangle : u \in [n]\}$. Finally, as before we define $|\tau\rangle = |s\rangle - |t\rangle$.

Positive Analysis The positive analysis is identical to the case of a complete graph (i.e. standard USTCON): a path from s to t gives a positive witness, or more generally, an st-flow, as in Exercise 4.4.1. However, now that we have allowed edges to take non-unit weights, we need to take these into account in the witnesses, by using $\frac{1}{\sqrt{\mathsf{w}_{u,v}}}|u,v\rangle$, in place of $|u,v\rangle$. For example, now a path $s = u_0, \dots, u_\ell = t$ with $u_{i-1} < u_i$ for all $i \in [\ell]$ gives the witness:

$$\sum_{i=1}^{\ell} \frac{1}{\sqrt{w_{u_{i-1},u_i}}}|u_{i-1},u_i\rangle \overset{A}{\mapsto} \sum_{i=1}^{\ell} \frac{1}{\sqrt{w_{u_{i-1},u_i}}}\sqrt{w_{u_{i-1},u_i}}(|u_{i-1}\rangle - |u_i\rangle) = |s\rangle - |t\rangle = |\tau\rangle.$$

More generally, a flow θ gives the positive witness

$$|w\rangle := \sum_{\{u,v\} \in E(\mathcal{G}(x)) : u<v} \frac{\theta(u,v)}{\sqrt{\mathsf{w}_{u,v}}}|u,v\rangle$$

whose norm is the energy $\mathcal{E}(\theta)$ in the *weighted* graph. Thus, $w_+(x) \le \mathcal{R}_{s,t}(\mathcal{G}(x))$ (which implicitly depends on the choice of edge weights).

Negative Analysis The negative analysis can be quite different from the complete graph case, because s and t not being connected depends on the absence of edges, and the only edges that can possibly occur in $\mathcal{G}(x)$ are those in $\mathcal{G}$. For example, if $\mathcal{G}$ has some edge e whose removal would disconnect s from t, then the absence of this one edge would be enough to witness that s and t are not connected. More generally, suppose $E_C \subseteq E(\mathcal{G}) \setminus E(\mathcal{G}(x))$ is a set of edges not present in $\mathcal{G}(x)$ such that their removal from $\mathcal{G}$ would cause s and t to be disconnected. Said another way: every st-path in $\mathcal{G}$ uses an edge in E_C. Then let $S \subset [n]$ be a set containing s, and exactly one endpoint of each edge in E_C. Then we can define a negative witness:

$$|\omega\rangle := \sum_{u \in S} |u\rangle,$$

and then similar to (4.10), we have

$$w_-(x) \le \|\langle\omega|A\|^2 = |E_C|,$$

which could potentially be very small. In fact, if we have multiple sets of the form of E_C, we can take linear combinations of their corresponding negative witnesses to get even smaller witnesses.

Example 4.4.1. While a branching program is a directed graph, if you remove the directions on the edges, you get a switching network that computes the same function. This gives us a quantum algorithm from any branching program, but we should of course not expect this to lead to a speedup in general.

For example, if you do this with the branching program in Figure 4.1 for OR, then you get a switching network for OR. There is always a unique path from s to either t_0 or t_1. When $\textsc{Or}(x) = 1$, the length of the path from s to t_1 is ℓ, where ℓ is the smallest value in $[n]$ such that $x_\ell = 1$, so $\mathcal{W}_+ = n$. When $\textsc{Or}(x) = 0$, meaning x is the all 0s string, then there is a unique path from s to t_0, but this path is not a span program witness – instead, the cut consisting of all the n edges labelled by 1 is the (unique) negative witness, so we get $\mathcal{W}_- = n$, and so the overall complexity of the span program is $\mathcal{C} = \sqrt{n \cdot n} = n$, which is not optimal for OR. We will soon see that a more clever switching network gives us a span program for OR with the optimal $\sqrt{n}$ complexity.

Extension to Multigraphs Sometimes it is useful to allow a switching network to be based on a *multigraph* $\mathcal{G}$, meaning a pair of vertices u and v might have multiple edges connecting them in $\mathcal{G}$, with different literals labelling different edges. In that case, we can assume these different edges have distinct labels λ, and we can modify $H_{i,b}$ in (4.12) to:

$$\begin{aligned} &H_{i,0} = \mathrm{span}\{|u,v,\lambda\rangle : \varphi_{u,v,\lambda}(x) = \neg x_i\} \\ \text{and}\quad &H_{i,1} = \mathrm{span}\{|u,v,\lambda\rangle : \varphi_{u,v,\lambda}(x) = x_i\}, \end{aligned} \tag{4.14}$$

and modify A in (4.13) to

$$A|u,v,\lambda\rangle = \sqrt{\mathsf{w}_{u,v,\lambda}}(|u\rangle - |v\rangle).$$

Formulas from Switching Networks Switching networks based on graphs of a particular kind – called series-parallel graphs – correspond to Boolean formulas. This leads to an alternative way of designing span programs for formula evaluation.

A series-parallel graph is a multigraph, defined by the following rules. A single edge between endpoints s and t is a series-parallel graph. If G_1 and G_2 are series-parallel graphs, their serial connection – in which the vertex labelled t in G_1 is identified with the vertex labelled s in G_2 – is a series parallel graph. For example, if G_1 and G_2 each consist of an edge with endpoints s and t, then connecting them in series yields a path of length two from s to t. If G_1 and G_2 are series-parallel graphs, their parallel connection – in which the vertices labelled s in G_1 and G_2 are identified, and the vertices labelled t in G_1 and G_2 are identified – is a series parallel graph. For example, if G_1 and G_2 are each a path of length 2 from s to t, then connecting them in parallel yields a graph that consists of two parallel paths from s to t (see Figure 4.2).

We now describe how to obtain a switching network for φ by induction on the depth. First, suppose φ has depth 0. A switching network $\mathcal{G}$ consisting of a single edge from s to t labelled by x_1 decides the formula $\varphi(x_1) = x_1$, since s and t are connected in $\mathcal{G}(x)$ if and only if $x_1 = 1$, which is if and only if $\varphi(x_1) = 1$. Similarly, a switching network consisting of a single edge from s to t labelled by $\neg x_1$ decides the formula $\varphi(x_1) = \neg x_1$.

If φ has depth $D > 0$, there are two cases. If $\varphi(x) = \bigvee_{i=1}^{d} \varphi_i(x^{(i)})$, then we can obtain a switching network $\mathcal{G}_\varphi$ by connecting switching networks $\mathcal{G}_{\varphi_1}, \dots, \mathcal{G}_{\varphi_d}$ in *parallel*, meaning we connect the graphs in parallel, and edges of the resulting graph inherit their edge labels from the graph they came from. This means that for any x, $\mathcal{G}_\varphi(x)$ is just the parallel connection of $\mathcal{G}_{\varphi_1}(x^{(1)}), \dots, \mathcal{G}_{\varphi_d}(x^{(d)})$, which has a path from s to t if and only if at least one of the $\mathcal{G}_{\varphi_i}(x^{(i)})$ has a path from s to t, which is if and only if $\varphi_i(x^{(i)}) = 1$.

The other case is where $\varphi(x) = \bigwedge_{i=1}^{d} \varphi_i(x^{(i)})$. Then we can obtain a switching network $\mathcal{G}_\varphi$ by connecting $\mathcal{G}_{\varphi_1}, \dots, \mathcal{G}_{\varphi_d}$ in *series*. Then for any x, $\mathcal{G}_\varphi(x)$ is just the serial connection of $\mathcal{G}_{\varphi_1}(x^{(1)}), \dots, \mathcal{G}_{\varphi_d}(x^{(d)})$, which has a path from s to t if and only if all of the $\mathcal{G}_{\varphi_i}(x^{(i)})$ have a path from s to t, which is if and only if $\varphi_i(x^{(i)}) = 1$ for all i.

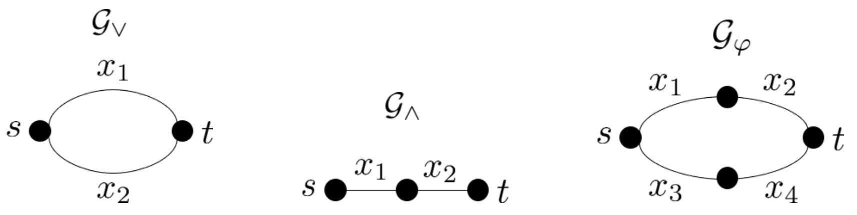
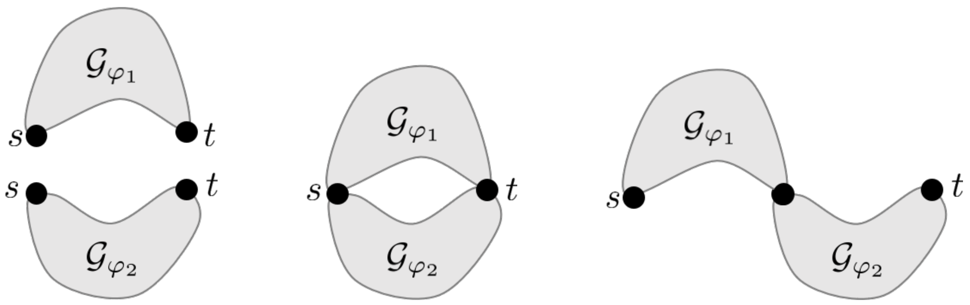


Figure 4.2: s and t are connected in a subgraph of $\mathcal{G}_\vee$ if and only if the edge x_1 *or* the edge x_2 is present. s and t are connected in a subgraph of $\mathcal{G}_\wedge$ if and only if the edge x_1 *and* the edge x_2 are present. s and t are connected in a subgraph of $\mathcal{G}_\varphi$ if and only if the formula $\varphi(x) = (x_1 \wedge x_2) \vee (x_3 \wedge x_4)$ evaluates to 1. Note that $\mathcal{G}_\varphi$ can be obtained by connecting two copies of $\mathcal{G}_\wedge$ in *parallel.*



$\mathcal{G}_{\varphi_1}$
s
t
s
t
$\mathcal{G}_{\varphi_2}$
$\mathcal{G}_{\varphi_1}$
s
t
$\mathcal{G}_{\varphi_2}$
$\mathcal{G}_{\varphi_1}$
s
t
$\mathcal{G}_{\varphi_2}$

Figure 4.3: Two switching networks, $\mathcal{G}_{\varphi_1}$ and $\mathcal{G}_{\varphi_2}$ can be connected in *parallel* (center), or in *series* (right). In the parallel connection, s and t are connected in $\mathcal{G}(x)$ if and only if there is an st-path in $\mathcal{G}_{\varphi_1}(x)$ *or* $\mathcal{G}_{\varphi_2}(x)$. In the series connection, s and t are connected in $\mathcal{G}(x)$ if and only if there is an st-path in $\mathcal{G}_{\varphi_1}(x)$ *and* $\mathcal{G}_{\varphi_2}(x)$.

For any formula φ, we can construct a span program from the switching network $\mathcal{G}_\varphi$, from which we can also obtain a quantum algorithm for evaluating φ. In the next exercise, you will analyze the complexity of this span program in the special case when φ is a *symmetric* formula.

Exercise 4.4.2. *Fix a symmetric formula φ on $\{0,1\}^N$, which is a formula of depth 0, or a formula of depth $d > 0$ of the form $\varphi(x) = \bigvee_{i=1}^{d_0} \varphi'(x^{(i)})$ or $\varphi(x) = \bigvee_{i=1}^{d_0} \varphi'(x^{(i)})$, for some symmetric formula φ', where $x^{(i)}$ is the i-th block of N/d_0 bits of x. That is, a formula is symmetric if it is the OR or AND of identical sub-formulas that are also symmetric. Then there are positive integers $d_0, \dots, d_{D-1}$ and some set $S_\vee \subseteq \{0, \dots, D-1\}$ such that for all $\ell \in S_\vee$, every node at distance ℓ from the root is a $\vee$ gate with d_ℓ children, and for all $\ell \in S_\wedge := \{0, \dots, D-1\} \setminus S_\vee$, every node at distance ℓ from the root is a $\wedge$ gate with d_ℓ children. Note that we must have $N = d_0 \dots d_{D-1}$. Let $\mathcal{G}_\varphi$ be the switching network for φ described above, and let P_φ be the span program derrived from this switching network as in (4.12) and (4.13), using $\mathsf{w}_{u,v} = 1$ for all edges.*

1. *Show that $\mathcal{W}_+(P_\varphi) \le \prod_{\ell \in S_\wedge} d_\ell$.*

2. *Show that $\mathcal{W}_-(P_\varphi) \le \prod_{\ell \in S_\vee} d_\ell$.*

From this we can conclude that $\mathsf{Q}(\text{EVAL}_\varphi) = O(\sqrt{N})$.

The same construction works for arbitrary formulas, also giving complexity $O(\sqrt{N})$, but in that case, non-unit edges weights must be used to balance the relative weights when combining graphs in series or parallel to account for the graphs being different.

A Superpolynomial Speedup There exists a family of formulas and promises on the input, yielding a formula evaluation problem called *k-fault trees* for which there is a superpolynomial separation between its quantum and classical query complexities [ZKH12]. It is possible to achieve the optimal quantum query upper bound using switching networks [JK17], which shows that switching networks can be used to achieve superpolynomial speedups.

4.5 Span Programs from Algorithms

In this section, we will see how to turn a quantum query algorithm into a span program, using a construction from [Rei09], which almost proves the other direction of Theorem 4.2.6. The reason it only "almost" proves it, is that the construction we give will only work for quantum algorithms with one-sided error. To prove one direction of Theorem 4.2.6, we would need a construction that makes a span program out of any bounded-error quantum algorithm, and these are generally allowed to err on both 1-inputs and 0-inputs. There is no known construction that takes any bounded-error quantum algorithm for F and converts it into a span program for F (even though we know from Theorem 4.2.6 that one exists). However, a similar construction to the one shown here can turn any bounded-error quantum algorithm for F into a span program that *approximates* F, a relaxed notion defined in [IJ19]. This construction for bounded-error algorithms can be found in [Jef22].

The algorithm Fix a quantum algorithm $U_1, \dots, U_T, |\psi_0\rangle, \mathcal{M}$, on the space

$$H_{\mathcal{A}} = \mathrm{span}\{|i\rangle|z\rangle : i \in [n] \cup \{0\}, z \in \mathcal{Z}\}$$

for some finite set $\mathcal{Z}$, where for simplicity we assume:

- $|\psi_0\rangle = |0, 0\rangle$
- T is odd
- for every odd $t \in [T]$, U_t is an input-independent unitary
- for every even $t \in [T]$, $U_t = \mathcal{O}_x^{\pm}$, which acts as $\mathcal{O}_x^{\pm}|i\rangle|z\rangle = (-1)^{x_i}|i\rangle|z\rangle$, where we interpret x_0 as 0. Note that this is similar to a controlled query, previously denoted $\mathsf{c}\mathcal{O}_x^{\pm}$, but it acts as the identity on $|0\rangle$ in the query register, instead of conditioned on a particular qubit.

We will suppose that this algorithm computes a problem F with one-sided error ε, meaning that if $\mathrm{F}(x) = 1$, then with probability 1, the algorithm outputs 1, and if $\mathrm{F}(x) = 0$, then with probability at least $1 - \varepsilon$, the algorithm outputs 0. (Note that this is slightly different from how one-sided error is usually defined, as the error is more commonly in the 1 case rather than the 0 case).

We will also assume that there is a *unique* accepting state $|\psi_T^{\mathsf{acc}}\rangle$, which is justified by the following exercise.

Exercise 4.5.1. *Fix any quantum algorithm that computes* F *with one-sided error* ε *using* T *unitaries from some set of allowed basic operations* $\mathcal{U}$ *that is closed under inverses, and includes* CNOT*; and assume for simplicity that its final measurement* $\mathcal{M}$ *just measures the last qubit in the computational basis. Show that there is another quantum algorithm that computes* F *with one-sided error at most* ε *using* $2T + 1$ *unitaries from the set* $\mathcal{U}$*, and additional qubit of memory, and such that if* $\mathrm{F}(x) = 1$*, then the final state is* $|\bar{0}1\rangle$ *– the state that is 0s everywhere, but has a 1 in the answer register.*

For any input x, we can define states for the algorithm:

$$\begin{aligned} |\psi_0(x)\rangle &:= |\psi_0\rangle \\ \forall t \in [T],\ |\psi_t(x)\rangle &:= U_t|\psi_{t-1}(x)\rangle. \end{aligned} \tag{4.15}$$

Then we have the following guarantees on $|\psi_T(x)\rangle$:

$$\begin{aligned} &\text{if } \mathrm{F}(x) = 1,\ |\psi_T(x)\rangle = |\psi_T^{\mathsf{acc}}\rangle \\ &\text{if } \mathrm{F}(x) = 0,\ \||\psi_T(x)\rangle - |\psi_T^{\mathsf{acc}}\rangle\|^2 \geq 1 - \varepsilon. \end{aligned} \tag{4.16}$$

The Span Program We now define a span program that decides F based on the quantum algorithm for F. First define, for every $i \in [n] \cup \{0\} \setminus \{1\}$ and $b \in \{0, 1\}$,

$$H_{i,b} := \mathrm{span}\{|t\rangle|i, b, z\rangle : t \in [T - 1], z \in \mathcal{Z}\}.$$

Let $H_{\text{true}} = H_{0,0}$ and $H_{\text{false}} = H_{0,1}$. Then we have $H = \bigoplus_{i\in[n],b\in\{0,1\}} H_{i,b} \oplus H_{0,0}$. Define

$$V := \text{span}\{|t\rangle|i,z\rangle : t \in \{0,\dots,T\}, i \in [n], z \in \mathcal{Z}\}$$

and

$$A|t\rangle|i,b,z\rangle := \begin{cases} |t\rangle|i,z\rangle - (-1)^b|t+1\rangle|i,z\rangle & \text{if } t \text{ is odd} \\ |t\rangle|i,z\rangle - |t+1\rangle U_{t+1}|i,z\rangle & \text{if } t \text{ is even.} \end{cases}$$

Notice then that for any $t \in [T-1]$, $i \in [n] \cup \{0\}$ and $z \in \mathcal{Z}$,

$$A|t\rangle|i,x_i,z\rangle = |t\rangle|i,z\rangle - |t+1\rangle U_{t+1}|i,z\rangle, \tag{4.17}$$

because when t is odd, $U_{t+1}|i,z\rangle = \mathcal{O}_x^{\pm}|i,z\rangle = (-1)^{x_i}|i,z\rangle$. Finally, let

$$|\tau\rangle := |0\rangle|\psi_0\rangle - |T\rangle|\psi_T^{\mathsf{acc}}\rangle.$$

The definition of this span program should remind you of the span program for USTCON (or more generally, switching networks) in Section 4.4: $|t\rangle|i,b,z\rangle$ encodes a *transition*, analogous to an edge, from the state $|t\rangle|i,z\rangle$ to $|t+1\rangle U_{t+1}|i,z\rangle$ (at least in the case of even t), and the starting state $|0\rangle|\psi_0\rangle$ (a register containing a timer set to 0, and another register with our initial state) represents $|s\rangle$; while the final state of an accepting computation $|T\rangle|\psi_T^{\mathsf{acc}}\rangle$ represents $|t\rangle$. We want to know if our computation induces a path from the starting state to this accepting state.

Positive Analysis Just as a path from s to t gives a positive witness for the span program in Section 4.4, a "computation path" gives a positive witness here. For any x, let M_x be the isometry that maps V to $H(x)$ as follows:

$$M_x|t\rangle|i,z\rangle = |t\rangle|i,x_i,z\rangle,$$

where we let $x_0 = 0$. Then

$$|w\rangle := M_x \sum_{t=0}^{T-1} |t\rangle|\psi_t(x)\rangle \in H(x).$$

The state $\sum_t |t\rangle|\psi_t(x)\rangle$ is called a *history state* (of the algorithm on input x), and it plays an important role in quantum computational complexity. We will show that $|w\rangle$ is a positive witness by showing that $A|w\rangle = |\tau\rangle$. To see this, note that

$$AM_x|t\rangle|i,z\rangle = A|t\rangle|i,x_i,z\rangle = |t\rangle|i,z\rangle - |t+1\rangle U_{t+1}|i,z\rangle,$$

by (4.17). Thus, for any $|\psi\rangle \in \text{span}\{|i,z\rangle : i \in [n]\cup\{0\}, z \in \mathcal{Z}\}$,

$$\begin{aligned} AM_x|t\rangle|\psi\rangle &= AM_x|t\rangle \sum_{i,z}\langle i,z|\psi\rangle|i,z\rangle = |t\rangle\sum_{i,z}\langle i,z|\psi\rangle|i,z\rangle - |t+1\rangle\sum_{i,z}\langle i,z|\psi\rangle U_{t+1}|i,z\rangle \\ &= |t\rangle|\psi\rangle - |t+1\rangle U_{t+1}\sum_{i,z}\langle i,z|\psi\rangle|i,z\rangle = |t\rangle|\psi\rangle - |t+1\rangle U_{t+1}|\psi\rangle. \end{aligned}$$

Thus:

$$\begin{aligned} A|w\rangle &= \sum_{t=0}^{T-1}\left(|t\rangle|\psi_t(x)\rangle - |t+1\rangle U_{t+1}|\psi_t(x)\rangle\right) \\ &= \sum_{t=0}^{T-1}\left(|t\rangle|\psi_t(x)\rangle - |t+1\rangle|\psi_{t+1}(x)\rangle\right) \\ &= |0\rangle|\psi_0(x)\rangle - |T\rangle|\psi_T(x)\rangle. \end{aligned}$$

We have $|\psi_0(x)\rangle = |\psi_0\rangle$ for all x, and for all x such that $\mathrm{F}(x) = 1$, $|\psi_T(x)\rangle = |\psi_T^{\mathsf{acc}}\rangle$, so when $\mathrm{F}(x) = 1$, we have $A|w\rangle = |\tau\rangle$, so $|w\rangle$ is a positive witness. We can thus upper bound:

$$w_+(x) \leq \||w\rangle\|^2 = \sum_{t=0}^{T-1}\|M_x|t\rangle|\psi_t(x)\rangle\|^2 = T,$$

since $M_x|t\rangle|\psi_t(x)\rangle$ still has $|t\rangle$ in the first register (making them orthogonal for different t), M_x is an isometry, and $|\psi_t(x)\rangle$ is a unit vector. Thus

$$\mathcal{W}_+ \leq T. \tag{4.18}$$

Negative Analysis A negative witness will also be based on a history state. Let $x \in \mathrm{F}^{-1}(0)$, and define

$$|\tilde{\omega}\rangle := \sum_{t=0}^{T} |t\rangle|\psi_t(x)\rangle.$$

First, note that

$$\langle\tilde{\omega}|\tau\rangle = \langle\psi_0|\psi_0(x)\rangle - \langle\psi_T^{\mathsf{acc}}|\psi_T(x)\rangle.$$

We have $|\psi_0(x)\rangle = |\psi_0\rangle$, and since $\mathrm{F}(x) = 0$, by (4.16), $\langle\psi_T^{\mathsf{acc}}|\psi_T(x)\rangle \neq 1$, so we can conclude $\langle\tilde{\omega}|\tau\rangle \neq 0$. Thus

$$|\omega\rangle := \frac{1}{\langle\tau|\tilde{\omega}\rangle}|\tilde{\omega}\rangle \text{ satisifes } \langle\omega|\tau\rangle = 1.$$

We will show that it is a negative witness by showing $\langle\omega|A$ (and thus $\langle\tilde{\omega}|A$) has no overlap with $H(x)$. We have for any $t \in [T-1]$, $i \in [n] \cup \{0\}$, and $z \in \mathcal{Z}$, using $x_i = 0$, and (4.17):

$$\begin{aligned}\langle\tilde{\omega}|A|t\rangle|i, x_i, z\rangle &= \langle\tilde{\omega}|(|t\rangle|i,z\rangle - |t+1\rangle U_{t+1}|i,z\rangle)\\ &= \langle\psi_t(x)|i,z\rangle - \langle\psi_{t+1}(x)|U_{t+1}|i,z\rangle = \langle\psi_t(x)|i,z\rangle - \langle\psi_t(x)|U_{t+1}^\dagger U_{t+1}|i,z\rangle = 0\end{aligned}$$

where we used that $|\psi_{t+1}(x)\rangle = U_{t+1}|\psi_t(x)\rangle$, by (4.15). This completes the proof that $|\omega\rangle$ is a negative witness, to compute its complexity, we first note that by (4.16),

$$\begin{aligned}1-\varepsilon \leq \||\psi_T(x)\rangle - |\psi_T^{\mathsf{acc}}\rangle\|^2 &= \||\psi_T(x)\rangle\|^2 + \||\psi_T^{\mathsf{acc}}\rangle\|^2 - 2\mathsf{Re}\langle\psi_T(x)|\psi_T^{\mathsf{acc}}\rangle\\ &= 2(1-\mathsf{Re}\langle\psi_T(x)|\psi_T^{\mathsf{acc}}\rangle) \leq 2\,|1-\langle\psi_T(x)|\psi_T^{\mathsf{acc}}\rangle| = 2|\langle\tilde{\omega}|\tau\rangle|.\end{aligned} \tag{4.19}$$

Next, to compute $\|\tilde{\omega}A\|^2$, we first note that for even $t \in [T-1]$, we always have (for any b, not just $b = x_i$):

$$\langle\tilde{\omega}|A|t\rangle|i,b,z\rangle = \langle\tilde{\omega}|(|t\rangle|i,z\rangle - |t+1\rangle U_{t+1}|i,z\rangle) = 0.$$

Thus, we have,

$$\begin{aligned}\|\langle\tilde{\omega}|A\|^2 &= \sum_{t\in[T-1],\text{odd}}\ \sum_{i\in[n]\cup\{0\},z\in\mathcal{Z}} |\langle\tilde{\omega}|A|t,i,\neg x_i,z\rangle|^2\\ &= \sum_{t\in[T-1],\text{odd}}\ \sum_{i\in[n]\cup\{0\},z\in\mathcal{Z}} |\langle\tilde{\omega}|\left(|t\rangle|i,z\rangle - (-1)^{\neg x_i}|t+1\rangle|i,z\rangle\right)|^2\\ &= \sum_{t\in[T-1],\text{odd}}\ \sum_{i\in[n]\cup\{0\},z\in\mathcal{Z}} |\langle\psi_t(x)|i,z\rangle + \langle\psi_{t+1}(x)|(-1)^{x_i}|i,z\rangle|^2\\ &= \sum_{t\in[T-1],\text{odd}}\ \sum_{i\in[n]\cup\{0\},z\in\mathcal{Z}} |\langle\psi_t(x)|i,z\rangle + \langle\psi_t(x)|i,z\rangle|^2\\ &= 4\sum_{t\in[T-1],\text{odd}}\ \sum_{i\in[n]\cup\{0\},z\in\mathcal{Z}} |\langle\psi_t(x)|i,z\rangle|^2\\ &= 4\sum_{t\in[T-1],\text{odd}} \||\psi_t(x)\rangle\|^2 = 4\left(\frac{T-1}{2}\right) = 2(T-1)\end{aligned} \tag{4.20}$$

Combining (4.19) and (4.20), we see

$$w_-(x) \leq \|\langle\omega|A\|^2 = \frac{\|\langle\tilde{\omega}|A\|^2}{|\langle\tilde{\omega}|\tau\rangle|^2} \leq \left(\frac{2}{1-\varepsilon}\right)^2 2(T-1)$$

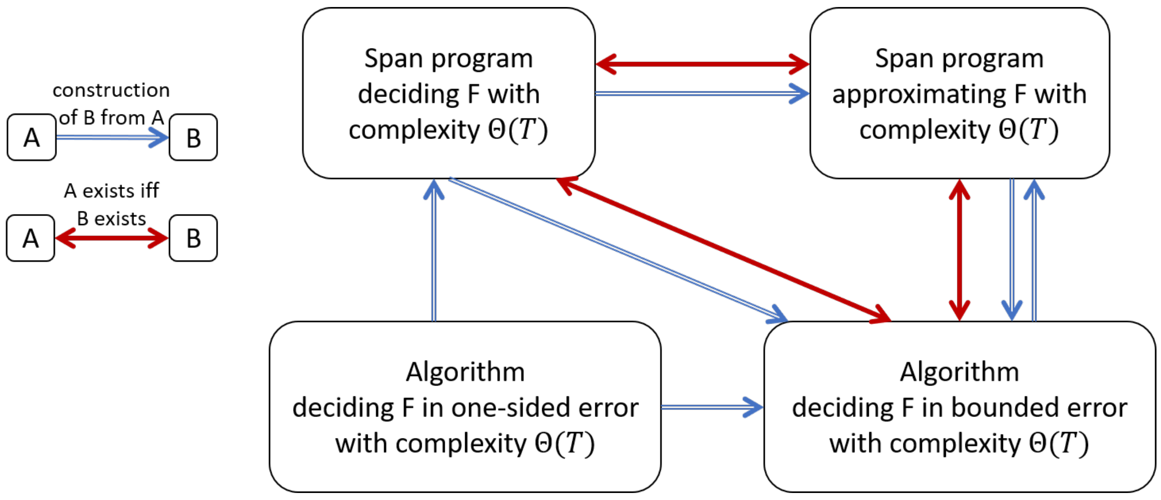

construction of B from A
A
B
A exists iff B exists
A
B
Span program deciding F with complexity Θ(T)
Span program approximating F with complexity Θ(T)
Algorithm deciding F in one-sided error with complexity Θ(T)
Algorithm deciding F in bounded error with complexity Θ(T)

Figure 4.4: Relationships between one-sided error algorithms, bounded-error algorithms, span programs, and approximate span programs. A red arrow indicates two objects either both exist, or both don't exist. A blue arrow indicates a construction from one to the other.

and so

$$\mathcal{W}_- \leq \frac{8}{(1-\varepsilon)^2}(T-1). \tag{4.21}$$

Thus, by (4.18) and (4.21), this span program has complexity

$$\mathcal{C}(P) \leq \sqrt{\frac{8}{(1-\varepsilon)^2}T(T-1)} = O(T)$$

for any constant $\varepsilon < 1$. This is exactly what we should expect. We could not hope to improve on the query complexity of an arbitrary quantum algorithm by getting something smaller than $(T-1)/2$ (the number of queries), but we also know by Theorem 4.2.6 that there exists a span program whose complexity matches the best possible query complexity, so converting any $(T-1)/2$-query algorithm to a span program with complexity $O(T)$ is not too much to hope for. Of course, this construction only works for one-sided error algorithms, whereas Theorem 4.2.6 would also leave open the possibility for such a construction that works for bounded (two-sided) error algorithms. While no such construction is known, it is possible to use a construction like the one we saw here to turn a bounded-error algorithm for F into a span program that *approximately decides* F [IJ19]. A span program that approximately decides F is almost as good as one that decides F, in that it can also be turned into a bounded-error quantum algorithm for F. See Figure 4.4 for a summary.

4.6 Further Directions

As mentioned in the previous section, a span program can be useful even if it doesn't perfectly decide F, but only approximately decides it. By this we mean that for all $x \in \mathrm{F}^{-1}(1)$, not only does there exist a positive witness, but it is less than some bound $\mathcal{W}_+$, whereas for any $x \in \mathrm{F}^{-1}(0)$, it is not necessarily the case that it has a negative witness, but if it instead has a positive witness, that positive witness must have size at least $2\mathcal{W}_+$. This turns out to be equivalent to saying that there is a vector $|\omega\rangle$ that is *close* to being a negative witness for x in the sense that $\left\|\langle\omega|A\Pi_{H(x)}\right\|$ is small (whereas we have been requiring that it be 0). This is described in [IJ19], where this idea is also used to turn any span program into a quantum algorithm that, on input x, outputs an estimate of $w_+(x,P)$. So for example, if this construction is used with the span program for USTCON, it yields a quantum algorithm for estimating $\mathcal{R}_{s,t}(G)$.

Span programs can also be turned into quantum algorithms for generating a quantum state proportional to the optimal positive witness. For example, using the span program for USTCON, we can get a quantum algorithm that generates a superposition over edges with amplitudes proportional to the optimal *flow*. Sampling the edges can be used to find a path from s to t (although this is only efficient in special cases) [JKP23].

Bibliography

[AJPW23] Simon Apers, Stacey Jeffery, Galina Pass, and Michael Walter. (no) quantum space-time tradeoff for USTCON. In *Proceedings of the 31st Annual European Symposium on Algorithms (ESA)*, pages 10:1–10:17, 2023. arXiv: 2212.00094 9

[CJOP20] Arjan Cornelissen, Stacey Jeffery, Maris Ozols, and Alvaro Piedrafita. Span programs and quantum time complexity. In *Proceedings of the 45th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 21:1–26:14, 2020. arXiv: 2005.01323 5

[HLŠ07] Peter Høyer, Troy Lee, and Robert Špalek. Negative weights make adversaries stronger. In *Proceedings of the 39th ACM Symposium on the Theory of Computing (STOC)*, pages 526–535, 2007. arXiv: quant-ph/0611054 2

[IJ19] Tsuyoshi Ito and Stacey Jeffery. Approximate span programs. *Algorithmica*, 81(6):2158–2195, 2019. arXiv: 1507.00432 15, 18

[Jef22] Stacey Jeffery. Span programs and quantum space complexity. *Theory of Computing*, 18(11):1–49, 2022. arXiv: 1908.04232 5, 15

[JK17] Stacey Jeffery and Shelby Kimmel. Quantum algorithms for graph connectivity and formula evaluation. *Quantum*, 1(26), 2017. arXiv: 1704.00765 14

[JKP23] Stacey Jeffery, Shelby Kimmel, and Alvaro Piedrafita. Quantum algorithm for path-edge sampling. In *Proceedings of the 18th Conference on the Theory of Quantum Computation, Communication, and Cryptography (TQC)*, pages 5:1–5:28, 2023. arXiv: 2303.03319 19

[KW93] Mauricio Karchmer and Avi Wigderson. On span programs. In *Proceedings of the IEEE 8th Annual Conference on Structure in Complexity Theory*, pages 102–111, 1993. 2

[Rei09] 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 2, 9, 15

[RŠ12] Ben W. Reichardt and Robert Špalek. Span-program-based quantum algorithm for evaluating formulas. *Theory of Computing*, 8(13):291–319, 2012. 2

[ZKH12] B. Zhan, S. Kimmel, and A. Hassidim. Super-polynomial quantum speed-ups for Boolean evaluation trees with hidden structure. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS 2012)*, pages 249–265, 2012. 14

.1 Singular value decomposition and Pseudoinverse

Every linear map $A : H \to H'$ has a *singular value decomposition*:

$$A = \sum_{i=1}^{r} \sigma_i |\psi_i\rangle\langle\phi_i|$$

for some positive real *singular values* $\sigma_1, \dots, \sigma_r$, orthonormal *left singular vectors* $|\psi_1\rangle, \dots, |\psi_r\rangle \in H$, and orthonormal *right singular vectors* $|\phi_1\rangle, \dots, |\phi_r\rangle \in H'$. From this, we can define the *pseudoinverse* of A:

$$A^+ = \sum_{i=1}^{r} \frac{1}{\sigma_i} |\phi_i\rangle\langle\psi_i|.$$

The right singular vectors are an orthonormal basis for the *rowspace* of A, which is the orthogonal complement of its kernel. The left singular vectors are an orthonormal basis for the *columnspace* of A, which is the image of A.