Quantum walk based search methods and algorithmic applications

M. Sc. Thesis

András Pál Gilyén

Advisor: Katalin Friedl
Department of Computer Science and Information Theory,
Budapest University of Technology and Economics

Faculty Advisor: Dömötör Pálvölgyi
Department of Computer Science

Eötvös Loránd University
Faculty of Science

Budapest, June 2, 2014

Acknowledgement

I would like to express my great appreciation to my advisor Katalin Friedl for the discussions we had on this thesis. I am very grateful for her detailed comments on both the content and the mathematical style.

I also would like to thank Ashley Montanaro who introduced me into this topic during my studies at the University of Cambridge, and with whom I worked on an essay providing a strong starting point for this thesis.

Finally I wish to acknowledge the help of Dömötör Pálvölgyi my faculty advisor.

Contents

Abstract

This thesis provides an introduction to the quantum walk based search method which is the quantum analogue of the classical Markov chain based search. We develop the basic theory of the Markov chain based search together with its quantisation due to Mario Szegedy[Sz2]. We deeply analyse their relationship, and sketch some of the possible algorithmic applications.

The thesis also addresses the question of the implementation, which is largely hidden by the high level description. We show how to implement the quantum walk based search algorithm in the case of the element distinctness problem. We implement the necessary operations using only very basic data structures, mainly arrays. Also while we use a weak and basic computational model we could still significantly reduce the exponent of the polylog terms arising from counting non oracle calls. Another interesting achievement is that we could parallelise the Setup operation reducing its circuit depth form $\mathcal{O}(N^c)$ to $\mathcal{O}(\text{polylog}(N))$. Probably the most important tool we introduced for the implementation is the use of reversible sorting networks, but we introduce some other tricks as well.

The optimal quantum query algorithm for the element distinctness problem was first described by Andris Ambainis [Amb3] and provided the basis of the generalised method. In this seminal paper Ambainis also addressed the question of implementation. However that implementation method does not fully cast into the general framework which was introduced later. The operations described by Ambainis use some involved data structures, such as skip lists and hash table. Also it uses a strong computational model for example, a qRAM query has cost only 1. In contrast with Ambainis's single node implementation method ours uses bipartite style approach fitting the general framework of Szegedy. At first sight it may seem unnecessary since it results in a duplication of a large amount of data stored during the walk steps, but in fact it simplifies the implementation process.

On the structure of the thesis: Section 1 covers some background on quantum walks and basic techniques related to the quantum walk based search method. Section 2-5 gives a comprehensive description of the general scheme introduced by Szegedy and derives the main theorems showing the quadratic speed-up compared to the classical case. These Sections roughly follow the structure of Szegedy's original paper [Sz2], but the proofs are restructured, so that I could state the main theorem in a slightly stronger form with a much improved constant. Section 6 addresses the implementation issues already mentioned. Finally, Section 7 gives a brief overview of the further generalised scheme introduced by Frédéric Magniez, Ashwin Nayak, Jérémie Roland and Miklós Sántha.

1. Introduction

Random walks on graphs or more generally Markov chains are interesting processes and worth exploring just for their beauty. Mathematicians have long been studying them, and built a diverse theory of random processes. Based on the theory various important practical applications were developed.

Random walks caught the attention of quantum information theorists as well. In the 90's, when they started thinking about how to define the quantum analogue for them, the topic rapidly became popular. Quantum random walks were constructed using various approaches, and their behaviour showed striking differences from their classical counterparts. Knowing that Random Walk based search is a highly successful paradigm of Computer Science, researchers tried to use these differences to gain algorithmic benefits for quantum computers. It was a long way to go, but finally quantum walk based search became one of the main resources of the speed-up of quantum algorithms.

1.1. Random Walks in classical Computer Science: Schöning's algorithm

To provide some intuition about how to use classical random walks to construct search algorithms, we shortly describe Schöning's famous algorithm [Sch]. This algorithm provides the basis of the best known algorithm for solving the 3-SAT problem. To indicate the importance of this problem we note that it is NP-complete. It means that if we could solve 3-SAT efficiently, then it would also provide efficient solutions to other important problems like the Travelling Salesman, Subgraph Isomorphism and Hamilton Circle Finding Problems. The question of 3-SAT is surprisingly simple though: given a "well organised" Boolean formula, can we assign TRUE and FALSE values to its Boolean variables such that the whole formula evaluates to TRUE?

For 3-SAT "well organised" Boolean formula means that it is presented in 3-CNF form. I.e. it should be given as a conjunction of clauses, where each clause is a disjunction of at most 3 literals, where a literal is either a Boolean variable or its negation.

3-CNF formula: $F(x_1,...,x_n) = (\underbrace{\underbrace{x_1}_{variable} \vee \underbrace{\bar{x_2}}_{negation}}_{literals} \vee x_3) \wedge \underbrace{(\bar{x_1} \vee \bar{x_4} \vee x_5)}_{clause\ of\ 3\ literals} \wedge \ldots \wedge (x_j \vee \bar{x_k} \vee \bar{x_l})$

Suppose we have a formula F with n variables. Then we will denote an assignment with $b = (b_1, b_2, \cdots, b_n)$ where each b_i is either TRUE or FALSE i.e. 1 or 0 correspondingly. Then the evaluation of F according to b can be simply written as $F(b)$.

Suppose that we have a Boolean formula F as above, then Schöning's algorithm works as the following:

- Pick a random assignment (or binary string) $b^0 \in \{0,1\}^n$.
- Repeat $3n$ times:
  - Check whether the actual assignment is a satisfying one: $F(b^i) = 1$?

  - ∗ If yes then stop and output b^i.
  - ∗ If not then take the first clause which is not satisfied, and flip the value of a randomly selected variable in the clause giving a new assignment $b^{(i+1)}$.
- Do this whole process $\left(\frac{4}{3}\right)^n$ times or until a satisfying assignment found. If after $\left(\frac{4}{3}\right)^n$ repetition still have not found a satisfying assignment, then simply output that F is not satisfiable.

The analysis of the random walk shows this process finds a good assignment with high probability if there is any. Note that throughout this thesis we use the term with high probability in the sense that the probability is greater than some universal constant c independent of the parameters of the problem. We use this term just in cases when only one sided error can occur. E.g. in this case it might happen that we say F is not satisfiable, because we did not find any good assignment, but in fact there is one (false negative). But we never say that F is satisfiable when it is in fact not (false positive). Then in the case of one sided error by only constantly many repetition of the process we can reach arbitrarily high probability, which explains the etymology of the term.

If the formula is $\mathcal{O}(\mathrm{poly}(n)$ sized then the overall time complexity can be bounded by $\mathcal{O}(1.334^n)$, which is not fast enough to transform the above mentioned problems to the easily solvable category. But we do not know a significantly better way to solve it. (As far as I know the best algorithm currently known for 3-SAT is running in time $\mathcal{O}(1.324...^n)$.)

We may think about this algorithm as a random walk based search process on the n-dimensional hypercube. The n dimensional hypercube has vertex set $\{0,1\}^n$, and two vertex is connected whenever the corresponding binary strings differ at only one place. We can also interpret satisfying assignments as marked vertices.

To see the main ideas behind consider the worst case, when there is only one marked vertex v. With probability $1/2$ we start no further than $n/2$ from v, as at the beginning we select a vertex uniformly at random. Then in each cycle we step towards v with probability at least $1/3$. It is because v is satisfying, and so in every clause at least one literal takes value TRUE according to the assignment described by v. But while $v \neq b$ we have at least one wrong clause for which every literal takes FALSE. Then with probability at least $1/3$ we flip the value of a variable for which the value of our current assignment b differed from v. Then it can be shown that after $3n$ steps the probability that we have found v is around $(3/4)^n$. But it is also very likely that we moved in the wrong direction many times, so if we have not already found v it is better to start again from a random position.

1.2. Some basic concepts of quantum and reversible computing

We are going to use the standard notations and concepts of quantum computing in this thesis. Due to limitations on the size we do not discuss the very basics in this thesis. An excellent introduction may be found e.g. in [Jozs].

We do not want to spend too much word dealing with the description of quantum circuits, we rather use the fact that any classical circuit can be transformed into a quantum circuit efficiently. First we describe how does the quantum version of a classical circuit work: (We will denote the bitwise addition by $\oplus$.)

Definition ***(Quantised version of a classical circuit)*** Suppose we are given a classical circuit that computes $f(in)$. Then the quantum version of it acts on two registers $|in\rangle\,|out\rangle$, and do the following transformation: $|in\rangle\,|out\rangle \longrightarrow |in\rangle\,|out \oplus f(in)\rangle$ in particular if the $|out\rangle$ register was in the 0 state at the beginning then we get the actual value of $|f(in)\rangle$ in the output register.

It is well-known that the quantum version of any classical circuit consisting N gates can be implemented using $\mathcal{O}(N)$ extra "workspace" qubits and by $\mathcal{O}(N)$ quantum gates [NC, Section 1.4]. To see an actual example check Figures 2,3 of Section 6. (Throughout this thesis we will always assume that we start the quantum computation with all qubits set to 0.)

1.3. Quantum Random Walks

The history of quantum random walks started with the quantisation of the symmetric discrete random walk on the line. The classical process is straightforward, we are walking on the integers staring from zero. Before each step we toss a coin, if it is heads we take one step forward, if it is tails we take one step backward. The coin is unbiased, so the probability of stepping to each direction is 1/2.

For the corresponding quantum version the natural state space would be $|n\rangle : n \in \mathbb{Z}$. Then intuitively we would define one step of the walk as

$$|n\rangle \to a\,|n-1\rangle + b\,|n\rangle + c\,|n+1\rangle\,.$$

As we want to describe a quantum mechanical operation we should set the values of a, b, c such that the transformation becomes unitary. The surprising fact is that only 3 choices of the values describe unitary evolution [Amb1]. Namely when one of a, b, c is 1, and the others are 0. The corresponding operations are shift to the left, identity and shift to the right - not quite like a random walk. So in this case discrete random walk do not naturally quantise on the vertex set - later it turned out that it is the case for almost all graphs.

The solution to this problem was to introduce and additional coin state. In the case of the line we amend the state space by one qubit splitting each position state to two different states $|n\rangle\,|0\rangle$ and $|n\rangle\,|1\rangle$. One step of the walk consist of two phases, a Coin Flip operation C and a Shift operation S. The Coin Flip is a local unitary transformation on the coin state, usually the Hadamard transformation.

$$\begin{array}{rcccccl} C: & |n\rangle\,|0\rangle & \to & d\,|n\rangle\,|0\rangle & + & e\,|n\rangle\,|1\rangle \\ C: & |n\rangle\,|1\rangle & \to & f\,|n\rangle\,|0\rangle & + & g\,|n\rangle\,|1\rangle \end{array}; \qquad H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{1}$$

The Shift corresponds to the actual step, we do a step according to the state of the coin.

$$\begin{aligned} S: \; |n\rangle|0\rangle &\rightarrow |n-1\rangle|0\rangle \\ S: \; |n\rangle|1\rangle &\rightarrow |n+1\rangle|1\rangle \end{aligned}$$

Now we can define one step of the quantum random walk as $W = SC$, for example with $C = H$. Note that quantum random walk defined this way has a unitary evolution, so the walk itself is not a random process, probability is introduced just by the final measurement. So we will usually call the process just quantum walk, dropping the expression "random" from the name.

After applying W a few times to the starting state $|0\rangle|0\rangle$ and measuring the position state we get similar statistics to the classical case. However after running the process for a longer period the interference accumulates, and we get very different results compared to the classical case.

In the classical case the position after T steps is binomially distributed: $B(T, 1/2)$. We also know, that for large T-s $B(T, 1/2)$ can be approximated by $\mathcal{N}(0, T/4)$. This implies that the classical walk on the line is concentrated roughly in $[-\sqrt{T}, \sqrt{T}]$.

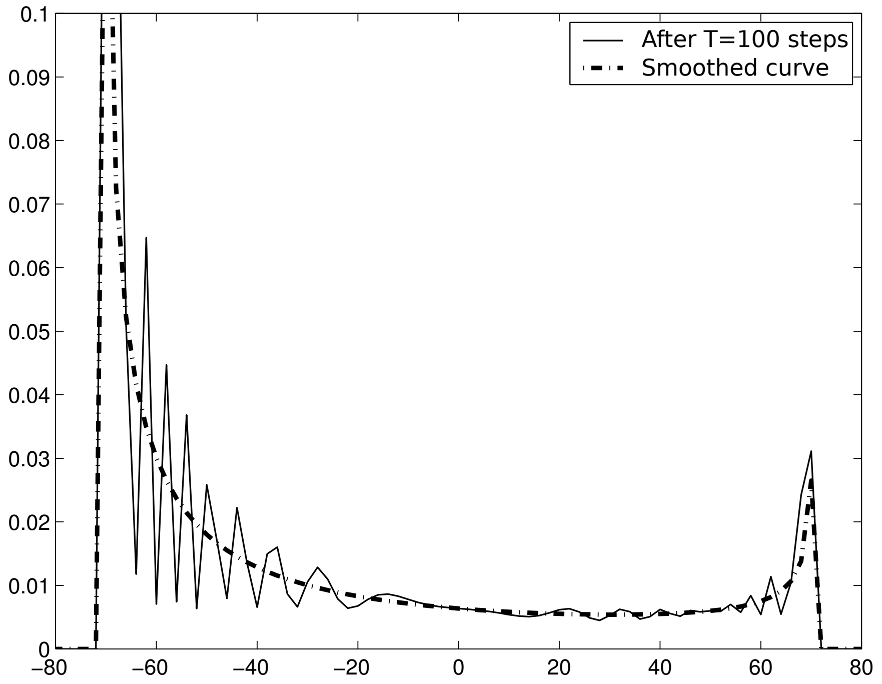


Figure 1: Measurement statistics of the quantum walk after 100 steps [Kem], starting from the initial state $|0\rangle|1\rangle$ and using H as a coin flip operator. (Only probabilities for even positions are plotted, since the probability of being at an odd positions is zero.)

Understanding that classical random walk on the line is fundamentally related to the normal distribution, the quantum walk W we just defined shows striking new features. Probably the most important two difference is the following: [Amb1]

- The maximum probability is reached for $|n| \approx T/\sqrt{2}$. [We may interpret this as the ability of the concentration of amplitude at a distant location due to constructive interference.]

- There are $\Omega(T)$ locations for which measuring n is $\Omega(1/T)$. [Compared to the classical walk we can interpret this as quadratically faster spreading.]

Following similar approaches various versions of quantum walks were defined for different graphs. For example quantum walks on regular graphs were introduced in an early stage of the theory. Suppose we have a graph which is d-regular, then regularity makes it possible to use a d-dimensional coin in a similar setting than in the case of the line. Simply numbering the outgoing edges at each vertex of the graph makes it possible to implement the Shift operation naturally. Let us denote the j-th neighbour of v with $n^j(v)$ according to the enumeration introduced. Then the Shift operation would act according to this numbering:

$$S : |v\rangle\,|j\rangle \to |n^j(v)\rangle\,|j\rangle$$

And we could use for example diffusion operator (D) as a Coin Flip operation:

$$D_d = \begin{pmatrix} -1+\frac{2}{d} & \frac{2}{d} & \cdots & \frac{2}{d} \\ \frac{2}{d} & -1+\frac{2}{d} & \ddots & 2\frac{2}{d} \\ \vdots & \ddots & \ddots & \vdots \\ \frac{2}{d} & \frac{2}{d} & \cdots & -1+\frac{2}{d} \end{pmatrix} \tag{2}$$

Understanding that coins represent somehow the edges of the graph, quantum walk was finally defined for any undirected graph following similar logic. Let $G = (V, E)$ then we can define the state space of the walk as $|v\rangle\,|e\rangle \in \mathbb{C}^{|V|\times|E|}$. Also we can define Shift naturally: (let us denote the edge between vertices u, v by e_{uv})

$$S : |v\rangle\,|e_{uv}\rangle \to |u\rangle\,|e_{uv}\rangle$$

The Coin Flip operation then would be some operation on the edges. Later in this thesis we are going to follow similar logic for quantising random walk on a general graph. However instead of using coin or edge space we are going to duplicate the vertex space.

Simultaneously to the evolution of the theory of discrete quantum walks, continuous time quantum walks were also studied on a wide range of graphs. The two approaches were rather different, for example in the continuous case there were no need to introduce coin states. In spite of the differences in the setting the behaviour of this two type of quantum walk was pretty similar.

Regardless the type of the walk or the actual graph, the two main differences we listed during the analysis of the discrete quantum walk on the line seemed to remain the most important features from an algorithmic point of view. The two main types of speed up gained by quantum walk based algorithms can be related to these:

- Exponentially faster hitting time - the ability of distant constructive interference
- Quadratic speed up in search problems - quadratically faster spreading

In this thesis we restrict our attention to the algorithmic implications of the "quadratically faster spreading" phenomena in the discrete quantum walk model.

1.4. Grover Search

Grover Search is an unstructured search method, meaning that it search for elements without any extra information about the elements. Suppose we have a large database, for which the elements are indexed for example with all the k bit length binary strings. And there is a set of marked elements M we are looking for, but we have no idea what indices belong to them. Then Grover Search can find a marked element with high probability quadratically faster than it is possible classically.

To see an example of unstructured search imagine the case when we have a large telephone book sorted alphabetically and that we are looking for the name of a person given only his/her telephone number.

For a computation based example consider the case when we have a very large number $n = a \cdot b$ where a, b are large prime numbers, and the marked set M is $\{a, b\}$. Then we can easily check for any $d \in \mathbb{Z}$ whether $d|n$ i.e. we can decide whether $d \in M$ knowing only n. We can even implement a quantum circuit which does efficiently calculate it for superpositions of multiple qubits, and changes the value of some register based on the result. But finding a, b knowing only n is very hard, no classical algorithm is known to be able to do that in polynomial time. (However the famous Shor algorithm can solve it efficiently using a quantum computer.)

Theorem 1 (Grover Search) *Suppose we have quantum oracle access to the indicator function of a marked set $M \subseteq \{0,1\}^k$, then we can find a marked element (which is a binary string) using $\mathcal{O}(\sqrt{2^k/|M|})$ oracle calls.*

Proof ***(Sketch)*** The quantum oracle describing M is given in the form of a quantum circuit (or black box) denoted by B. B can mark qubits describing a binary string $b \in M$, for now let us define this feature in the following form (which is basically equivalent to the calculation of an indicator function in quantum sense, see e.g. [Jozs]):

$$B : |b\rangle \to \begin{cases} -|b\rangle & \text{if } b \in M \\ |b\rangle & \text{if } b \notin M \end{cases} \tag{3}$$

Then Grover search simply iterates the process $G = D_{2^k} B$ for $\mathcal{O}(\sqrt{2^k/|M|})$ times starting from the uniform distribution over all bit strings $|\psi_0\rangle$, and finally measure the state.

The starting state

$$|\psi_0\rangle = \frac{1}{\sqrt{2^k}} \left(\sum_{\text{all } b \in \{0,1\}^k} |b\rangle \right) = H_k |0\rangle \tag{4}$$

can be easily prepared from $|0\rangle$ using a Hadamard transformation (1) on each qubit (shortly denoted by H_k). Also we can observe that the diffusion operator D_{2^k} can be interpreted as a reflection through the line corresponding ψ_0: $D_{2^k} = \mathrm{ref}_{\psi_0} = 2|\psi_0\rangle\langle\psi_0| - Id$. It also shows how to implement it: $Id - 2|\psi_0\rangle\langle\psi_0| = H_k F H_k$ where F

is the flip operation which flips the amplitude of $|0\rangle$ and acts as Id on the orthogonal complement of $|0\rangle$.

Using similar arguments it can be shown that $G = \mathrm{ref}_{\psi_0}\mathrm{ref}_{M^\perp}$ is a product of reflections. Also it follows that starting from $|\psi_0\rangle$ we never leave the subspace $(\psi_0, \Pi_M(\psi_0))$, where Π_M denotes the orthogonal projection to the subspace generated by the marked binary strings. Restricted to this subspace G becomes a simple rotation as it is a product of reflections on the plane. For small values of $|M|/2^k$ the rotation angle is $\approx \sqrt{|M|/2^k}$, which yields that after $\mathcal{O}(\sqrt{2^k/|M|})$ rotations our state gets close to $|\Pi_M(\psi_0)/|\Pi_M(\psi_0)|\rangle$. Then this provides good chance to find a marked element after measuring the sate. $\square$

1.5. Grover Search as random walk

Before we go for further generalisations we show the connection between random walks and Grover search:

A version of Grover Search can be viewed as a quantisation of a uniform random walk on the complete graph K_n. If we add loops for each node and define the quantum walk as we did for regular graphs then both the vertex space and the coin space becomes n dimensional: $|x\rangle|y\rangle \in \mathbb{C}^{n\times n}$. Then the Shift operation can be simply defined as $|x\rangle|y\rangle \leftrightarrow |y\rangle|x\rangle$. So we may omit the Shift operation and simply apply the diffusion operator alternately to the right and left registers.

We are going to generalise Markov chains following similar logic.

1.6. Amplitude Amplification and the speed-up of Schöning's algorithm

Amplitude Amplification is simply a generalisation of Grover Search obtained by replacing the starting state $|\psi_0\rangle$ (4) with the output of an arbitrary quantum algorithm. Suppose we have a quantum circuit C which, starting from the zero state can prepare a state $|\phi\rangle = C|0\rangle$. If this state reveals a marked element with probability p after measurement, then normally we would need to prepare and measure the state $\mathcal{O}(1/p)$ times to get one marked with high probability. Amplitude Amplification reduces the time of this process quadratically.

Theorem 2 (Amplitude Amplification) *Suppose we have a quantum circuit C together with its inverse C^{-1} and suppose that measuring $|\phi\rangle = C|0\rangle$ reveals a marked element with probability at least p. If we have also access to another circuit B which can check marked element as described in the Grover Algorithm* (3)*, then we can find a marked element with high probability using $\mathcal{O}(\sqrt{1/p})$ calls to C, C^{-1} and B.*

Proof ***(Sketch)*** We use the notations of the proof of the Grover Search. In the case of Amplitude Amplification we use a modified Grover operator $G_C = \mathrm{ref}_\phi\mathrm{ref}_{M^\perp}$, and start the process with state $|\phi\rangle$. Following similar lines than in the proof of Grover Search it can be shown that applying the modified operator $\mathcal{O}(\sqrt{1/p})$ times we can find an element with high probability after measuring the state. Considering that ref_ϕ can be implemented in the form $-CFC^{-1}$ it yields a quadratic reduction in the number of repetitive usage of C. (Note that quantum circuits are fundamentally reversible due to their unitary nature, so implementing C^{-1} should be no harder than implementing C.) $\square$

More is true, to state the result concerning the exact version of Amplitude Amplification we use the following notation: Let M be the subspace spanned by the computational basis states corresponding to the marked binary string. Let us call M the good subspace and $M^{\perp}$ the bad subspace, then (see e.g. [Jozs, ES 3]):

Theorem 3 (Exact version of Amplitude Amplification) *Suppose we have the starting state for the AA process: $|\phi\rangle = \alpha |\phi_g\rangle + \beta |\phi_b\rangle$ where α and β are real and positive, and $|\phi_g\rangle$ and $|\phi_b\rangle$ are the good and bad projections of $|\phi\rangle$ re-normalised to unit length. Suppose the indicator function $f(x) = 0$ resp. 1 for x good resp. bad, can be computed. Suppose also that the value of α is known. Then using one extra workspace qubit and a constant number of additional unitary gates the Amplitude Amplification can be made exact, i.e. to output a state being a superposition of only marked computational basis states.*

To illustrate how powerful this Amplitude Amplification paradigm is, we show that it can speed up Schöning's algorithm quadratically. (This was first observed by Ambainis [Amb2].)

We need to deal with classical randomness. A classical circuit which can use random steps is usually defined as a standard circuit with an input register and a random bit register. Then it generates the necessary probability distributions for itself based on the assumption, that its random bit register contains independent uniform random bits. Then it is an easy fact that Schöning algorithm can be efficiently implemented in such a classical circuit model.

After having the quantum version of the above described circuit much of the work is done. We should only feed its random bit registers with uniformly distributed random bits. It can be easily done by using a Hadamard transformation for each qubit of the random bit register. For the checking step we should only implement a conditional phase flip controlled by the output qubits of the circuit, which is also an easy task.

As we already discussed the success probability is greater than $c(3/4)^n$ for some $c \in \mathbb{R}^+$ constant, so using Amplitude Amplification we can solve the problem using $\mathcal{O}(\sqrt{4/3}^n)$ repetition, as we stated. If the formula is $\mathcal{O}(\text{poly}(n))$ sized than it reduces the overall running time to $\mathcal{O}(1.155^n)$.

Corollary 4 *For $\mathcal{O}(poly(n))$ sized $3-CNF$ formulas there is a quantum algorithm that solves $3-SAT$ in time $\mathcal{O}(\sqrt{4/3}^n)$ with bounded one side error.*

Having seen that Amplitude Amplification can speed up classical and quantum algorithms in that generality, we might wonder if there is any need to consider the possibility of quadratic speed up using random walks. However there is a part of the algorithms that Amplitude Amplification can not enhance, namely the steps needed to prepare the state $|\phi\rangle$. In the case of Schöning's algorithm this part was insignificant compared to the repetition number, but this is not always the case as we will see for example in the case of the Element Distinctness Problem.

1.7. From classical Random Walks to Markov chains

Discrete Random Walk on a graph is a random process, consisting discrete steps. Between each step we are resting at some node of the graph. At the beginning we start at some node of the graph, often this starting position is also being chosen randomly according to some initial distribution. Then at each step we go to a neighbouring node (or stay at the node) according to some (not necessarily uniform) probability distribution. We will only consider the case, when the individual steps are independent of each other, and the transition probabilities are unchanged over time. This kind of random walk can be descried as a stationary Markov chain with the states being the nodes of the graph.

Definition ***(Stationary Markov Chain)*** A stationary Markov chain is a discrete time stochastic process $(M_i) : i \in \mathbb{N}_0^+$ on a discrete state set X (which is going to be always finite throughout this thesis), such that $\forall i \in \mathbb{N}_0^+$: $M_i \in X$. The probability of moving from one state $x \in X$ to another $y \in X$ in one step is $p_{xy} = P(\text{we step to } y|\text{we are at } x)$ described by a transition matrix P indexed by the elements of X: $P = \{p_{xy}\}_{x,y\in X}$. P is stationary i.e. unchanged over time, and fully describes the dynamics of the chain: $\forall x, y \in X\ \forall i \in \mathbb{N}_0^+ : P(M_{i+1} = y|M_i = x) = p_{xy}$.

We consider only stationary Markov chains on finite state sets throughout this thesis. For simplicity we will always use the plain term Markov chain, but we refer to the stationary and finite case. Note that P acts from the right, so probability distributions on X will be represented by row vectors.

We are going to use several definitions about Markov chains, we list them here in advance for later reference.

Definition ***(Symmetricity)*** We say that a Markov chain is symmetric if $P = P^T$.

Definition ***(Underlying graph)*** The underlying graph for a Markov chain is a directed graph which have vertex set X, and the edges correspond to the transitions which have non-zero probability. So replacing the non-zero elements of P with 1-s we get the adjacency matrix of the underlying directed graph.

Note that symmetricity correspond to undirected graphs, and even stronger to walks for which the the transition probabilities depend only on the edges, not on the endpoints.

Definition ***(Irreducibility)*** A Markov chain is said to be irreducible if the underlying graph is strongly connected, i.e. every state is reachable from every other state.

Definition ***(Periodicity)*** A state $x \in X$ has period k if any return to state x must occur in multiples of k steps. Formally, the period of a state is defined as $\gcd(n|(P^n)_{xx} > 0)$ A Markov chain is aperiodic if every state has period 1.

For irreducible Markov chains every state has the same period. So we can also define the period of an irreducible Markov chain.

Definition ***(Ergodicity)*** A Markov chain is said to be ergodic if it is irreducible and aperiodic.

Definition ***(State transitivity)*** A Markov chain is state transitive if any $x \in X$ can be carried to any other $y \in X$ by such a permutation of X that leaves the transition matrix unchanged.

Definition ***(Stationary distribution)*** The stationary distribution of a Markov chain denoted by τ is a non negative vector, for which the elements sum to 1, and satisfies: $\tau P = \tau$

An irreducible chain always have a unique stationary distribution which is also strictly positive (see for example [Mey]). If the chain is symmetric then the vector having all coordinates $1/|X|$ is always a stationary distribution.

Definition ***(Time reversed chain)*** The time reversal of an irreducible Markov chain is denoted by P^* and is defined by the following equality:

$$\tau_x p_{xy} = \tau_y p^*_{yx}$$

Definition ***(Reversibility)*** An irreducible Markov chain is said to be reversible if $P = P^*$.

The transition matrix has non-negative elements, and the sum of the elements in any row is 1. Such matrices are called stochastic matrices. Every stochastic matrix describes some Markov chain, so we can identify a stationary Markov chain with its stochastic transition matrix P. We will often call a stochastic matrix just Markov chain. So the above concepts translate to definitions for (stochastic) matrices as well.

2. Bipartite Walks and their quantisation

In the previous Sections we have seen that discrete random walks do not naturally quantise on the vertex space. However Szegedy showed [Sz1, Sz2] that this is not the case for bipartite walks.

2.1. Classical Bipartite Walks

A bipartite walk is a two phase walk on the finite and disjoint vertex sets X, Y. In the first phase we start from X and do a transition to Y according to the probabilities of the transition matrix P. In the second phase we do a transition from Y to X according to an other matrix Q. Here P, Q are stochastic matrices: $P \in \mathbb{R}_0^{+|X|\times|Y|}, Q \in \mathbb{R}_0^{+|Y|\times|X|}$, $\forall x \in X : \sum_{y\in Y} p_{xy} = 1$ and similarly $\forall y \in Y : \sum_{x\in X} q_{yx} = 1$. We are going to denote this bipartite walk by (P, Q). (For transition matrices we are working with row (density) vectors so we apply them from the left, and we index the rows and columns of P, Q by the elements of X, Y.)

One step of the walk in terms of probability distributions: If we start from an initial distribution ϱ_X after the first phase we get the distribution $\varrho_Y = \varrho_X P$ on Y and after the second phase we get a new distribution on X: $\varrho'_X = \varrho_Y Q = \varrho_X PQ$.

Note that every finite Markov chain M can be viewed as a bipartite walk. If M is described by the transition matrix P on state space X, then we should simply duplicate the state space X and set $Q = P$. One step of the resulting bipartite walk then corresponds to a double step of M.

2.2. Quantisation of Bipartite Walks

As we indicated bipartite walks quantise naturally on the space of vertices, so the quantised version of the bipartite walk (P, Q) takes place in the Hilbert space, labelled by the elements of X and Y. We call the space of labels for X and Y correspondingly left and right walk registers:

$$H = \{|x\rangle |y\rangle \, | x \in X, y \in Y\} \simeq \mathbb{C}^{|X|\times|Y|}$$

We would like to define our walk in a way that one step (or phase) of it transfers amplitudes only between states corresponding to neighbour vertices. This means that the process mirrors the structure of the graph, i.e. we are in some sense ”walking” on the graph. (The constraint to move amplitudes only along edges is often called the local or spatial search problem.)

Szegedy defined one step of the walk using right and left transition states.

Definition ***(Left and Right Transition States)***

$$\begin{aligned} \forall x \in X : \quad |r_x\rangle &= \textstyle\sum_{y\in Y} \sqrt{p_{xy}}\, |x\rangle |y\rangle \\ \forall y \in Y : \quad |l_y\rangle &= \textstyle\sum_{x\in X} \sqrt{q_{yx}}\, |x\rangle |y\rangle \end{aligned} \tag{5}$$

In the first phase of the quantum bipartite walk we flip the phase of the transition states for every $|r_x\rangle$. Assuming the quantum system is in state $|\psi\rangle = \gamma_1 |r_x\rangle^{\perp} + \gamma_2 |r_x\rangle$ (where $|r_x\rangle^{\perp}$ is some orthogonal state to $|r_x\rangle$) then we put the system into state $\gamma_1 |r_x\rangle^{\perp} - \gamma_2 |r_x\rangle$. In other worlds we apply the unitary transformation $(I - 2|r_x\rangle\langle r_x|)$. Since $\langle t_x | t_{x'}\rangle = \delta_{x,x'}$ these operators commute for all $x \in X$, and their product can be written as $\left(I - 2\sum_{x\in X} |r_x\rangle\langle r_x|\right)$. The operation for the second phase is defined symmetrically: $\left(I - 2\sum_{y\in Y} |l_y\rangle\langle l_y|\right)$.

Definition ***(Bipartite Walk Operator)***
The walk operator describing one step of the quantised bipartite walk (P, Q) is defined as

$$W_{(P,Q)} = \left(I - 2\sum_{y\in Y} |l_y\rangle\langle l_y|\right)\left(I - 2\sum_{x\in X} |r_x\rangle\langle r_x|\right) = \left(2\sum_{y\in Y} |l_y\rangle\langle l_y| - I\right)\left(2\sum_{x\in X} |r_x\rangle\langle r_x| - I\right)$$

Observe that $\left(2\sum_{x\in X} |r_x\rangle\langle r_x| - I\right)$ is the reflection on the subspace spanned by the transition vectors of X derived from P: $T_R^P = \mathrm{Span}(|r_x\rangle : x \in X)$ and similarly $\left(2\sum_{y\in Y} |l_y\rangle\langle l_y| - I\right)$ is the reflection on the other "transition" subspace $T_L^Q = \mathrm{Span}(|l_y\rangle : y \in Y)$. Using these observations we can finally write our walk operator in a cleaner form:

$$W_{(P,Q)} = \mathrm{ref}_{T_L^Q} \cdot \mathrm{ref}_{T_R^P} \tag{6}$$

In the case when the bipartite walk (P, P) is a duplicated Markov chain we denote the corresponding quantum walk operator $W_P = \mathrm{ref}_{T_R^P} \cdot \mathrm{ref}_{T_L^P}$. Note that in general $T_R^P \neq T_L^P$.

3. Spectra of product of reflections

The following theorem is at the core of the quadratic speed-up phenomena and provides the basis for the analysis of the quantised walk $W_{(P,Q)}$. It is a higher dimensional generalisation to the fact that the product of two reflections on the plane is a rotation. The proof roughly follows the one described in [Sz2], however I have completed and amended both the statement and the proof to fit the framework better. Note that this theorem is a variant of a result due to C. Jordan [Jor], and may be derived from it.

Theorem 5 (Real Spectral Lemma) *Let H be a Hilbert space, and let A and B be matrices whose columns form orthonormal systems spanning the subspaces $V_A = Span(A_{.1}, \dots, A_{.n})$ and $V_B = Span(B_{.1}, \dots, B_{.m})$. Let $W = ref_B \cdot ref_A = (2BB^\dagger - I)(2AA^\dagger - I)$, then H can be decomposed to W invariant orthogonal subspaces in terms of the discriminant matrix $D(A,B) = A^\dagger \cdot B$. Let $(\lambda_l, (u_l, v_l) : 1 \le l \le h)$ be the system of positive singular values in decreasing order with the corresponding (left,right) singular vector pairs. Then $\lambda_1 \le 1$, and let k be the largest index such that $\lambda_k = 1$. Then the "busy" subspace $Span(V_A, V_B)$ can be decomposed using these vector pairs:*

(i) On $V_A \cap V_B = Span(Au_l : 1 \le l \le k) = Span(Bv_l : 1 \le l \le k)$ the operator W acts as Id.

(ii) For every $k+1 \le l \le h$ let $\lambda_l = cos(\theta_l)$ then on the subspace $V_l = Span(Au_l, Bv_l)$ W acts as a rotation by $2\theta_l$, and in the basis (Au_l, Bv_l) $W|_{V_l}$ has real coefficients.

(iii) On $V_A \cap V_B^\perp$ the operator W acts as $-Id$. Let $(u_l | h+1 \le l \le n)$ be the system of left singular vectors for $D(A,B)$ then $V_B^\perp \cap V_A = Span(Au_l | h+1 \le l \le n)$.

(iv) On $V_A^\perp \cap V_B$ the operator W acts as $-Id$. Let $(v_l | h+1 \le l \le m)$ be the system of right singular vectors for $D(A,B)$ then $V_A^\perp \cap V_B = Span(Bv_l | h+1 \le l \le m)$.

The invariant subspaces above are pairwise orthogonal, and span the "busy" subspace. The "idle" subspace $V_A^\perp \cap V_B^\perp = Span(V_A, V_B)^\perp$ is also invariant and W acts on it as Id.

Moreover if $D(A,B) \in \mathbb{R}^{n \times m}$ then all the above subspaces are spanned by real linear combinations of the column vectors of A, B. In particular if A, B have real coefficients in the computational basis then W can be transformed to the real Jordan normal form by an orthogonal transformation of the computational basis.

Proof On the "idle" subspace $V_A^\perp \cap V_B^\perp (= \mathrm{Span}(V_A, V_B)^\perp)$ W acts as $(-Id)\cdot(-Id) = Id$. So we can concentrate on its orthogonal complement the "busy" subspace $\mathrm{Span}(V_A, V_B)$. Take the singular value decomposition of $A^\dagger B = U \Lambda V^\dagger$, where U, V are unitary matrices and Λ is a diagonal matrix with the singular values in the diagonal in decreasing order. In other words $\Lambda = \mathrm{diag}_{n\times m}(\lambda_1, \lambda_2, \dots, \lambda_{\min(n,m)})$. Now define $\tilde{A} = AU, \tilde{B} = BV$. Since U, V are unitary operators $V_A = V_{\tilde{A}}, V_B = V_{\tilde{B}}$ and we can express W equally well using $\tilde{A}, \tilde{B}$: $W = (2BB^\dagger - I)(2AA^\dagger - I) = (2BVV^\dagger B^\dagger - I)(2AUU^\dagger A^\dagger - I) = (2\tilde{B}\tilde{B}^\dagger - I)(2\tilde{A}\tilde{A}^\dagger - I)$. Also the discriminant matrix becomes much simpler $D(\tilde{A}, \tilde{B}) =$

$U^\dagger D(A,B)V = \Lambda$. Finally the column vectors for $\tilde{A}, \tilde{B}$ still form orthonormal systems: $I_n = A^\dagger A = U^\dagger A^\dagger A U = \tilde{A}^\dagger \tilde{A}$, $I_m = B^\dagger B = V^\dagger B^\dagger B V = \tilde{B}^\dagger \tilde{B}$.

Let denote $\tilde{A} = (|\tilde{a}_i\rangle, \ldots, |\tilde{a}_n\rangle)$, $\tilde{B} = (|\tilde{b}_i\rangle, \ldots, |\tilde{b}_m\rangle)$. Using this formulation we can alternatively write $W = (2\sum_{j=1}^m |\tilde{b}_j\rangle\langle\tilde{b}_j| - I)(2\sum_{i=1}^n |\tilde{a}_i\rangle\langle\tilde{a}_i| - I)$, and the above relations can be expressed in the following simple form:

$$\langle\tilde{a}_i|\tilde{a}_j\rangle = \delta_{i,j} = \langle\tilde{b}_i|\tilde{b}_j\rangle \tag{7}$$

$$\langle\tilde{a}_i|\tilde{b}_j\rangle = \delta_{i,j}\lambda_i = \langle\tilde{b}_j|\tilde{a}_i\rangle \tag{8}$$

Using (7,8) we can see that $W|\tilde{a}_i\rangle = -|\tilde{a}_i\rangle + 2\lambda_i|\tilde{b}_i\rangle$ and $W|\tilde{b}_i\rangle = -2\lambda_i|\tilde{a}_i\rangle + (4(\lambda_i)^2 - 1)|\tilde{b}_i\rangle$. Using the unitarity of W we see that $1 = \langle\tilde{b}_i|\tilde{b}_i\rangle\langle W\tilde{a}_i|W\tilde{a}_i\rangle \geq |\langle\tilde{b}_i|W|\tilde{a}_i\rangle|^2 = |2\lambda_i - 1|^2$ so $\lambda_i \leq 1$ for every i.

(ii): For $k+1 \leq l \leq h$ $V_l = \mathrm{Span}(|\tilde{a}_l\rangle, |\tilde{b}_l\rangle)$ is two dimensional because $\langle a_l|b_l\rangle \neq 1$, and invariant as the above formulae for $W|\tilde{a}_l\rangle, W|\tilde{b}_l\rangle$ show. W_{V_l} can be written as $(2|\tilde{b}_l\rangle\langle\tilde{b}_l| - I)(2|\tilde{a}_l\rangle\langle\tilde{a}_l| - I)$ which is a product of two reflections on the plane V_l. So it is a (real) rotation by angle $2\theta_l$ where $\cos(\theta_l) = \langle a_l|b_l\rangle = \lambda_l$.

(iii-iv): $V_A \cap V_B^\perp = \mathrm{Span}\Big(|\tilde{a}_i\rangle \mid h+1 \leq i \leq n\Big)$ and $V_A^\perp \cap V_B = \mathrm{Span}\Big(|\tilde{b}_j\rangle \,|h+1 \leq j \leq m\Big)$ is also apparent from (8) and it is also clear that $W = \mathrm{ref}_B \cdot \mathrm{ref}_A$ acts as $-Id$ on these subspaces.

(i): $V_A \cap V_B \supseteq \mathrm{Span}\Big(|\tilde{a}_l\rangle = |\tilde{b}_l\rangle \,|1 \leq l \leq k\Big)$ since $\langle a_l|b_l\rangle = \lambda_l = 1$ which means $|a_l\rangle = |b_l\rangle$. The action of $W = \mathrm{ref}_B \cdot \mathrm{ref}_A$ on $V_A \cap V_B$ is obviously Id.

The relations (7,8) show the orthogonality of the above decomposition of $\mathrm{Span}(V_A, V_B)$. For the proof completeness of the decomposition observe that $\dim(\mathrm{Span}(V_A, V_B)) = \dim(V_A) + \dim(V_B) - \dim(V_A \cap V_B) = n + m - \dim(V_A \cap V_B)$. Also note that we have already described disjoint orthogonal subspaces of dimension $2*(h-k) + (n-h) + (m-h) + k = n + m - k$. So $n + m - \dim(V_A \cap V_B) \geq n + m - k \Leftrightarrow \dim(V_A \cap V_B) \leq k$. It implies that equality holds everywhere, $V_A \cap V_B = \mathrm{Span}\Big(|\tilde{a}_l\rangle = |\tilde{b}_l\rangle \,|1 \leq l \leq k\Big)$ and that our decomposition is complete.

For the real case note that if $D(A,B)$ is real then U, V can be chosen to be real orthogonal matrices.

Finally observe that the singular vector pairs of $D(A,B)$ are the columns of U, V: $(U_{.i}, V_{.i})$ corresponding to the singular values λ_i. Substituting $|\tilde{a}_i\rangle = AU_{.i}$ and $|\tilde{b}_i\rangle = BV_{.i}$ completes the proof. ■

Remark 6 *If A, B from the Real Spectral Lemma has common columns, we can simply remove them, the reduced matrices A', B' result in the same transformation W.*

Proof Let $C = \mathrm{Span}(c_1, \ldots, c_g)$ be the subspace spanned by the common columns of A, B. On the invariant subspace $C^\perp$ $\mathrm{ref}_A = \mathrm{ref}_{A'}$ and $\mathrm{ref}_B = \mathrm{ref}_{B'}$. And on the invariant subspace C $\mathrm{ref}_A = -\mathrm{ref}_{A'}$, $\mathrm{ref}_B = -\mathrm{ref}_{B'}$. So $\mathrm{ref}_A\mathrm{ref}_B = \mathrm{ref}_{A'}\mathrm{ref}_{B'}$ is unchanged on the whole Hilbert space. The only thing happens we transfer vectors from $V_A \cap V_B$ to $V_{A'}^\perp \cap V_{B'}^\perp$. ■

The following lemma is mine and is intended to fill a hole in the proof about the quantum hitting time in [Sz2]. To figure out the proof I used the insight given by the Real Spectral Lemma, that for any $v \in V_A^{\perp}$ every component of v in the invariant "rotation subspaces" of W is also in $V_A^{\perp}$.

Lemma 7 *Let A, B, V_A, V_B as in the Real Spectral Lemma, then for $W = W_{A,B} = ref_{V_B} ref_{V_A}$ and $u \in V_A^{\perp}$ we have $(W^t + (W^{\dagger})^t)u \in V_A^{\perp}$.*
Similarly $v \in V_B^{\perp} \Rightarrow (W^t + (W^{\dagger})^t)v \in V_B^{\perp}$.

Proof Since W is unitary $W^{\dagger} = W^{-1} = \mathrm{ref}_{V_A}\mathrm{ref}_{V_B}$,
also observe $u \in V_A^{\perp} \Leftrightarrow \mathrm{ref}_{V_A} u = -u$:

$$(W^t + (W^{\dagger})^t)u = W^t u + (W^{\dagger})^t(-\mathrm{ref}_{V_A} u) = \underbrace{\mathrm{ref}_{V_B}\mathrm{ref}_{V_A}, \ldots, \mathrm{ref}_{V_B}\mathrm{ref}_{V_A}}_{t} u -$$

$$\mathrm{ref}_{V_A} \underbrace{\mathrm{ref}_{V_B}\mathrm{ref}_{V_A}, \ldots, \mathrm{ref}_{V_B}\mathrm{ref}_{V_A}}_{t} u = (W^t - \mathrm{ref}_{V_A} W^t)u = -\mathrm{ref}_{V_A}(W^t - \mathrm{ref}_{V_A} W^t)u$$

where the last equality shows that $(W^t + (W^{\dagger})^t)u \in V_A^{\perp}$. The second case is given by the symmetry: $W_{A,B} + W_{A,B}^{-1} = \mathrm{ref}_{V_B}\mathrm{ref}_{V_A} + \mathrm{ref}_{V_A}\mathrm{ref}_{V_B} = W_{B,A}^{-1} + W_{B,A}$ $\blacksquare$

4. Hitting Times for classical Markov chains and their relation to the corresponding Quantum Walk

This Section covers the main result of Szegedy. I present restructured, corrected and amended versions of the proofs what also makes me able to state the main theorem in a slightly stronger form. The proofs only roughly follow the ones in [Sz2], and I also filled a gap (see Remark 13).

4.1. Hitting Time for classical Markov chains and the Leaking Walk Matrix

The problem is the following: We have a Markov chain described by the transition matrix P on the state space X, and a set of marked items $M \subseteq X$. We start by picking an element $x \in X$ uniformly at random. The question is how long should we run our random process to find an element of M.

Definition ***(Classical Hitting Time)*** The classical hitting time $\mathcal{H}(P, M)$ is the expected number of steps needed to reach M

$$\mathcal{H}(P, M) = \mathbb{E}(\text{Numer of steps done before reaching } M)$$

where the expectation is both over the initial (uniform) distribution and over the random process described by the Markov chain.

Note that this expression is called Average Hitting Time in the literature of Random Walks, since it is an average over all the individual hitting times of the nodes of the graph.

We can formulate the search process as a modified Markov chain. Before each step we check whether $x \in M$, if yes we stay at x (forever), otherwise we do a transition to a new state y according to the probabilities described by P_{xy}.

Definition ***(Leaking Walk Matrix)*** The transition matrix of the above described modified Markov chain is the so called leaking walk matrix.

$$P_L = \begin{pmatrix} P_M & P' \\ 0 & I \end{pmatrix} \tag{9}$$

Where P_M is the restriction of P to the subset $X \setminus M$, and P' is similarly the restriction of P to the $(X \setminus M) \to M$ transitions.

From now on we are assuming $P = P^T$ throughout this Section, and introduce several notations to simplify formulae. Let e_x be the vector with all 0 coordinates except for x where it has value 1. For any $S \subseteq X$ let $\mathbb{1}_S = \sum_{x \in S} e_x$ and $\mathbb{1} = \mathbb{1}_X$. For a (row) vector v define $\|v\|_1 = \sum_{x \in X} |v_x|$. If v is non negative then we can write it $\|v\|_1 = v \cdot \mathbb{1}^T$. Let $\rho_0 = \frac{1}{|X|}\mathbb{1}$, and in general let us denote the probability distribution of the walk after t steps with ρ_t, so that $\|\rho_t\|_1 = 1$ and the probability of being in the set S is $\rho_t \mathbb{1}_S^T$.

Theorem 8 (Classical Hitting Time) *Let $P = P^T$ be a symmetric Markov chain on the set X. Then for any set of marked elements $M \subseteq X$:*

$$\mathcal{H}(P, M) = \sum_{c_i \neq 0} c_i^2 \frac{1}{1 - \lambda_i} \tag{10}$$

where the numbers (c_i, λ_i) correspond to the decomposition of the vector $\sqrt{\frac{1}{|X|}}\mathbb{1}_{X \setminus M}$ of the form $\sum_{i=1}^{|X \setminus M|} c_i \cdot v_i$ according to an orthonormal eigenvalue system (λ_i, v_i) of the leaking walk matrix P_M.

If P is irreducible as well then for every $\varnothing \neq M \subseteq X$ this expression is finite.

Proof We will use a neat formula for calculating expectation. Suppose X is a random variable taking values in $\mathbb{N}_0^+$. Then we can calculate its expectation as follows:

$$\mathbb{E}[X] = \sum_{i=0}^{\infty} iP(X = i) = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} P(X = i \,\&\, i > j) = \sum_{j=0}^{\infty}\sum_{i=0}^{\infty} P(X = i \,\&\, i > j) = \sum_{j=0}^{\infty} P(X > j)$$

Note that this formula is correct even in the case when the expectation is infinite.

This formula and the notations we introduced makes the calculation of $\mathcal{H}(P, M)$ formally simple:

$$\mathbb{E}(\text{Number of steps done before reaching } M) = \sum_{t=0}^{\infty} P(M \text{ is not reached after } t \text{ steps}) =$$

$$\sum_{t=0}^{\infty} \rho_t \mathbb{1}_{X\setminus M}^T = \sum_{t=0}^{\infty} \frac{1}{|X|}\mathbb{1}(P_M)^t \mathbb{1}_{X\setminus M}^T = \sum_{t=0}^{\infty} \left(\sqrt{\frac{1}{|X|}}\mathbb{1}_{X\setminus M}\right)(P_M)^t \left(\sqrt{\frac{1}{|X|}}\mathbb{1}_{X\setminus M}^T\right)$$

Let $\hat{u} = \sqrt{\frac{1}{|X|}}\mathbb{1}_{X\setminus M}$. Since P is symmetric P_M is symmetric as well so it can be diagonalised using an orthogonal transformation, and its eigenvalues are real numbers. Let $u_i : 1 \leq i \leq |X \setminus M|$ be an orthonormal eigenvector system with the corresponding eigenvalues λ_i. We can now express $\hat{u} = \sum_{i=1}^{|X\setminus M|} c_i u_i$. We can continue our calculations:

$$\mathcal{H}(P, M) = \sum_{t=0}^{\infty} \hat{u}(P_M)^t \hat{u}^T = \sum_{t=0}^{\infty} \sum_{i=1}^{|X\setminus M|} c_i^2 \lambda_i^t = \sum_{c_i \neq 0} c_i^2 \frac{1}{1 - \lambda_i} \tag{11}$$

Here the last equation needs some caution, to justify it we need to bound the eigenvalues of P_M first.

Since for all rows $p_{x.}^M$ of P_M $\|p_{x.}^M\|_1 \leq 1$ we get that for any vector $v \in \mathbb{C}^{|X\setminus M|}$:

$$\|vP_M\|_1 = \sum_{y \in X\setminus M} \left|vp_{.y}^M\right| = \sum_{y}^{X\setminus M} \left|\sum_{x}^{X\setminus M} v_x p_{xy}^M\right| \leq \sum_{x,y}^{X\setminus M} \left|v_x p_{xy}^M\right| = \sum_{x}^{X\setminus M} |v_x| \|p_{x.}^M\|_1 \leq \|v\|_1 \tag{12}$$

which means that the eigenvalues of P_M are bounded by 1 in absolute value.

If $|\lambda| = 1$ for some v eigenvector then we have equality everywhere, in particular $\sum_x |v_x| \|p^M_{x.}\|_1 = \|v\|_1$ i.e. $\|p^M_{x.}\|_1 = 1$ for all x such that $v_x \neq 0$. Let $S = \{x \in X \setminus M : v_x \neq 0\}$, since no cancellations can occur in (12) we have that $\forall x \in S : 1 = \sum_{y \in S} p^M_{xy}$. It implies that the probability of an $S \to X \setminus S$ transition is 0 according to P. Since $M \subseteq X \setminus S$ it can not happen if $M \neq \varnothing$ and P is irreducible.

Returning to the final step of (11): if for all $c_i \neq 0$ $|\lambda_i| \leq 1$ then the sum is absolute convergent, and we can use the $\frac{1}{1-\lambda_i}$ formula for the individual geometric series. The sum has a (possibly infinite) limit anyway, because the summation on t contains only non-negative terms. If there is a $c_i \neq 0$ for an eigenvector with $\lambda_i = 1$, then the sum must be infinite - just as the expression $\frac{c_i^2}{0}$ appearing on the right hand side. If there is a $c_j \neq 0$ for an eigenvector with $\lambda_j = -1$ there must be also an eigenvector v_i with $\lambda_i = 1, c_i \neq 0$, because we are working with probability distributions which can not eventually became negative at some indices.

We justified (11) and thus completed the proof. ■

Remark 9 *By looking at the proof we can say more precisely that the expression above is finite iff $\forall x \in X \setminus M \ \exists\ m \in M$ such that there is a path from x to m in the graph corresponding to P.*

Note that P and P_M might have negative eigenvalues which can cause problems in the analysis of (10) because $\frac{1}{1-\lambda}$ and $\frac{1}{1-|\lambda|}$ can differ in magnitudes, but the corresponding singular values for $\lambda, -\lambda$ are the same: $|\lambda|$. However intuitively we feel that for a probability distribution $\sum_{c_i \neq 0} c_i^2 \frac{1}{1-|\lambda_i|}$ can not be significantly larger than $\sum_{c_i \neq 0} c_i^2 \frac{1}{1-\lambda_i}$. The heuristic argument behind is that the coefficient for any negative eigenvalue is at most as big as the sum of coefficients for larger positive eigenvalues, otherwise the probability density vector would eventually contain negative entries. The rigorous version of the above argument provides the following lemma for the uniform distribution:

Lemma 10 *Using the decomposition from Theorem 8 of $\hat{u} = \sqrt{\frac{1}{|X|}} \mathbb{1}_{X \setminus M} = \sum_{i=1}^{|X \setminus M|} c_i \cdot v_i$, we have the following inequalities concerning the c_i coefficients and the corresponding eigenvalues of P_M:*

$$\sum_{c_i \neq 0} c_i^2 \frac{1}{1-|\lambda_i|^2} \leq \sum_{c_i \neq 0} c_i^2 \frac{1}{1-\lambda_i} \leq 2 \sum_{c_i \neq 0} c_i^2 \frac{1}{1-|\lambda_i|^2} \tag{13}$$

Proof Following the proof of Theorem 8 and looking at (11) we see that $P(M$ is not reached after t steps$) = \sum_{i=1}^{|X \setminus M|} c_i^2 \lambda_i^t$. Since these probabilities are non-increasing in t, we can conclude that for $t \in \mathbb{N}_0^+$

$$\sum_{i=1}^{|X \setminus M|} c_i^2 \lambda_i^{2t} \leq \sum_{i=1}^{|X \setminus M|} c_i^2 \lambda_i^{2t} + \sum_{i=1}^{|X \setminus M|} c_i^2 \lambda_i^{(2t+1)} \leq 2 \sum_{i=1}^{|X \setminus M|} c_i^2 \lambda_i^{2t}$$

which after summing up for all $t \in \mathbb{N}_0^+$ yields the following geometric sequences

$$\sum_{t=0}^{\infty} \sum_{i=1}^{|X \setminus M|} c_i^2 \lambda_i^{2t} \leq \sum_{t=0}^{\infty} \sum_{i=1}^{|X \setminus M|} c_i^2 \lambda_i^{t} \leq 2 \sum_{t=0}^{\infty} \sum_{i=1}^{|X \setminus M|} c_i^2 \lambda_i^{2t}. \quad ■$$

4.2. Hitting Time for the Quantum Leaking Walk

For a Markov chain P on state set X in the presence of a marked set $M \subseteq X$ we define the quantum leaking walk W_{P_L} as the quantisation of the leaking walk matrix P_L (9).

Before defining hitting time for the quantum case we should first find the quantum analogue for the uniform starting distribution. A natural definition for the initial state $|\phi_0\rangle$ can be given using the transition states of (5), which is independent of the marked set M:

$$|\phi_0\rangle = \frac{1}{\sqrt{|X|}} \sum_{x \in X} |r_x\rangle$$

We define quantum hitting time in terms of this initial state and the quantised leaking walk:

Definition ***(Quantum Hitting Time)*** For quantised walk W_P and marked set M we define the hitting time $\mathcal{H}(W_P, M) = T'$ to be the smallest number of steps T' such that $\forall T \geq T'$:

$$\frac{1}{T+1} \sum_{t=0}^{T} \left| W_{P_L}^t |\phi_0\rangle - |\phi_0\rangle \right|^2 \geq 1 - |M|/|X| \tag{14}$$

Szegedy accounted for this definition based on the observation that the hitting time for classical Markov chains is typically in the order of the time when $\rho_0 P^t$ becomes significantly skewed towards M i.e. when $\|\rho_0 P^t - \rho_0\|_1$ becomes significantly large.

We are going to analyse $\mathcal{H}(W_P, M)$ using our Real Spectral Lemma. We will need to deal with several trigonometric expression and not to interrupt the train of thought later, we prove the following facts beforehand.

Proposition 11

$$\left| \sum_{t=0}^{T} cos(t2\beta) \right| \leq \frac{1}{2} + \frac{1}{2\sqrt{1 - cos^2(\beta)}} \leq \sqrt{\frac{1}{1 - cos^2(\beta)}} \tag{15}$$

Proof To prove (15) we will use the following inequality:

$$|\cos(2\beta + \gamma) - \cos(\gamma)| \leq 2|\sin(\beta)| \tag{16}$$

To show that this inequality holds let us fix β and look at the extremal values of $\cos(2\beta + \gamma) - \cos(\gamma)$ in γ by finding the roots of its derivative: $\sin(\gamma) - \sin(2\beta + \gamma)$. If $\beta \notin \{n\pi | n \in \mathbb{Z}\}$ it has two solutions modulo 2π: $\gamma = \pi/2 - \beta$ and $\gamma = 3\pi/2 - \beta$; the corresponding values of $\cos(2\beta + \gamma) - \cos(\gamma)$ are $2\sin(\beta), -2\sin(\beta)$. (The $\beta = n\pi$ case is trivial.)

Now we start proving (15) by calculating the sum:

$$\sum_{t=0}^{T} \cos(2t\beta) = \sum_{t=0}^{T} \frac{e^{2it\beta} + e^{-2it\beta}}{2} = \frac{1}{2} \left(\frac{e^{2i(T+1)\beta} - 1}{e^{2i\beta} - 1} + \frac{e^{-2i(T+1)\beta} - 1}{e^{-2i\beta} - 1} \right) =$$

$$\frac{1}{2} \left(\frac{(e^{2i(T+1)\beta} - 1)(e^{-2i\beta} - 1) + (e^{-2i(T+1)\beta} - 1)(e^{2i\beta} - 1)}{(e^{2i\beta} - 1)(e^{-2i\beta} - 1)} \right) =$$

$$= \frac{\cos(2T\beta) - \cos(2(T+1)\beta) + 1 - \cos(2\beta)}{2 - 2\cos(2\beta)} = \frac{1}{2}\left(1 + \frac{\cos(2T\beta) - \cos(2T\beta + 2\beta)}{1 - (\cos^2(\beta) - \sin^2(\beta))}\right)$$

Now we apply (16) to the numerator and use in the denominator $1 = \cos^2(\beta) + \sin^2(\beta)$:

$$\left|\sum_{t=0}^{T} \cos(t2\beta)\right| \leq \frac{1}{2}\left(1 + \frac{2|\sin(\beta)|}{2\sin^2(\beta)}\right) = \frac{1}{2}\left(1 + \frac{1}{|\sin(\beta)|}\right) = \frac{1}{2} + \frac{1}{2\sqrt{1-\cos^2(\beta)}} \quad \text{finally}$$

$$\sqrt{1-\cos^2(\beta)} \leq 1 \Rightarrow \frac{\sqrt{1-\cos^2(\beta)}}{2} + \frac{1}{2} \leq 1 \Rightarrow \frac{1}{2} + \frac{1}{2\sqrt{1-\cos^2(\beta)}} \leq \frac{1}{\sqrt{1-\cos^2(\beta)}} \quad \blacksquare$$

We are ready to prove the following nice bound on the quantum hitting time:

Theorem 12 (Quantum Hitting Time)

$$\mathcal{H}(W_P, M) \leq 2\sum_{\tilde{c}_i \neq 0} \tilde{c}_i^2 \sqrt{\frac{1}{1 - |\lambda_i|^2}}$$

where $M \subsetneq X$ *and the* $\tilde{c}_i$ *coefficients correspond to the decomposition of* $\sqrt{\frac{1}{|X\setminus M|}}\mathbb{1}_{X\setminus M} = \sum_{i=1}^{|X\setminus M|} \tilde{c}_i u_i$ *according to the* (u_i, λ_i) *orthonormal eigenvalue system of the symmetric matrix* P_M*.*

Proof To start the estimation of $\mathcal{H}(W_P, M)$, we take a closer look at (14).

$$\left|W_{P_L}^t \left|\phi_0\right\rangle - \left|\phi_0\right\rangle\right|^2 = \langle W_{P_L}^t \phi_0 | W_{P_L}^t \phi_0\rangle - \langle W_{P_L}^t \phi_0 | \phi_0\rangle - \langle \phi_0 | W_{P_L}^t \phi_0\rangle + \langle \phi_0 | \phi_0\rangle = 2 - 2\langle W_{P_L}^t \phi_0 | \phi_0\rangle \tag{17}$$

where in the last equality we used the fact that W_{P_L} and ϕ_0 has real coefficients in the computational basis.

W_{P_L} is defined in terms of the modified right and left transition states, which are identical on the set of marked elements $\forall m \in M : |r'_m\rangle = |m\rangle|m\rangle = |l'_m\rangle$. The other transition vectors of P are not influenced by the presence of M. By Remark 6 we may view W_{P_L} as the product of reflections on the subspaces spanned by these unchanged transition vectors:

$$T_R^M = \mathrm{Span}(|r_x\rangle : x \in X \setminus M),\ T_L^M = \mathrm{Span}(|l_{x'}\rangle : x' \in X \setminus M); \quad W_{P_L} = \mathrm{ref}_{T_L^M} \cdot \mathrm{ref}_{T_R^M} \tag{18}$$

Inspired by this we break $|\phi_0\rangle$ into two parts $|\phi_{01}\rangle, |\phi_{02}\rangle$: $|\phi_{01}\rangle = \sum_{x \in X\setminus M} |r_x\rangle$, $|\phi_{02}\rangle = \sum_{m\in M} |r_m\rangle$, so

$$\langle \phi_{01} | \phi_{01}\rangle = 1 - |M|/|X|\,, \quad \langle \phi_{02} | \phi_{02}\rangle = |M|/|X| \tag{19}$$

Using this decomposition we get that

$$\langle W_{P_L}^t \phi_0 | \phi_0\rangle = \langle W_{P_L}^t \phi_{01} | \phi_{01}\rangle + \langle W_{P_L}^t \phi_{01} | \phi_{02}\rangle + \langle W_{P_L}^t \phi_{02} | \phi_{01}\rangle + \langle W_{P_L}^t \phi_{02} | \phi_{02}\rangle \tag{20}$$

Using that $|\phi_{01}\rangle, |\phi_{02}\rangle$ have real coefficients: $\langle W_{P_L}^t \phi_{01}|\phi_{02}\rangle = \langle \phi_{02}|W_{P_L}^t \phi_{01}\rangle$ resulting in

$$\langle W_{P_L}^t \phi_{01}|\phi_{02}\rangle + \langle W_{P_L}^t \phi_{02}|\phi_{01}\rangle = \langle ((W_{P_L}^\dagger)^t + W_{P_L}^t)\phi_{02}|\phi_{01}\rangle \tag{21}$$

We would like to show that this equals 0. Observe that $|\phi_{01}\rangle \in T_R^M$ and $|\phi_{02}\rangle \in (T_R^M)^\perp$. This condition compared with my Lemma 7 together with the reduced form of W_{P_L} (18) justifies that the right hand side of (21) is indeed 0.

Using the Cauchy–Schwarz inequality and (19): $|\langle W_{P_L}^t \phi_{02}|\phi_{02}\rangle| \le |W_{P_L}^t \phi_{02}||\phi_{02}| = |\phi_{02}|^2 = |M|/|X|$. Putting this into (20), and using that (21) is zero we get the following lower bound, after substituting into (17):

$$\left|W_{P_L}^t |\phi_0\rangle - |\phi_0\rangle\right|^2 \ge 2 - 2\langle W_{P_L}^t \phi_{01}|\phi_{01}\rangle - 2|M|/|X|$$

This bound simplifies the estimation of $\mathcal{H}(W_P, M)$ (14):

$$1-|M|/|X| \le \sum_{t=0}^{T} \frac{\left|W_{P_L}^t |\phi_0\rangle - |\phi_0\rangle\right|^2}{T+1} \Leftarrow 1-|M|/|X| \le \sum_{t=0}^{T} \frac{2 - 2\langle W_{P_L}^t \phi_{01}|\phi_{01}\rangle - 2|M|/|X|}{T+1}$$

$$\Leftrightarrow -(1-|M|/|X|) \le \sum_{t=0}^{T} \frac{-2\langle W_{P_L}^t \phi_{01}|\phi_{01}\rangle}{T+1} \Leftrightarrow \frac{T+1}{2} \ge \sum_{t=0}^{T} \frac{\langle W_{P_L}^t \phi_{01}|\phi_{01}\rangle}{1-|M|/|X|} \tag{22}$$

It is time to use the Real Spectral Lemma of Section 3. To keep the notations appropriate let us now the identify T_R^M, T_L^M width the matrix formed by the transition vectors of (18) as columns. Using that $\langle r_x|l_y\rangle = \sqrt{p_{xy}}\sqrt{p_{yx}}$ and that $P = P^T$ the discriminant matrix of the Real Spectral Lemma $D(T_R^M, T_L^M)$ becomes $(T_R^M)^\dagger T_L^M = P_M$.

Now we should relate the orthogonal diagonalisation of $P_M = O^T \Lambda O$ to its singular value decomposition. The relation is simple, we should only change the sign of the negative diagonal elements of Λ (shortly denoted by $|\Lambda|$) and change the sign of the corresponding rows of the orthogonal matrix on the right hand side resulting O'. The corresponding singular value decomposition is $P_M = O^T|\Lambda|O'$. It means that the orthogonal spectral decomposition of a vector according to $P_M = O^T\Lambda O$ is the same as the decomposition according to the left singular vectors of $P_M = O^T|\Lambda|O'$ just the singular values have opposite sign if the corresponding eigenvalue was negative.

Using the form $|\phi_{01}\rangle = T_R^M \hat{u}$, where just like before $\hat{u} = \sqrt{\frac{1}{|X|}}\mathbb{1}_{X\setminus M}$, we can further decompose $\hat{u} = \sum_{i=1}^{|X\setminus M|} c_i u_i$ so that u_i has singular value $|\lambda_i|$. This is essentially the same decomposition as discussed in (11). We can finally write $|\phi_{01}\rangle = \sum_{i=1}^{|X\setminus M|} c_i T_R^M u_i$. Now applying the Real Spectral Lemma of Section 3 and noting that the vectors $T_R^M u_i$ lie in pairwise orthogonal W_{P_L} invariant subspaces we get

$$\langle W_{P_L}^t \phi_{01}|\phi_{01}\rangle = \sum_{i=1}^{|X\setminus M|} c_i^2 \langle W_{P_L}^t T_R^M u_i|T_R^M u_i\rangle = \sum_{c_i\neq 0} c_i^2 \cos(2t\theta_i) \quad \text{where } \cos(\theta_i) = |\lambda_i| \tag{23}$$

Using (15) we can bound the following sum: $\left|\sum_{t=0}^{T} \langle W_{P_L}^t \phi_{01}|\phi_{01}\rangle\right| \le \sum_{c_i\neq 0} c_i^2 \sqrt{\frac{1}{1-|\lambda_i|^2}}$ Noting the normalisation $\tilde{c}_i = c_i/\sqrt{1-|M|/|X|}$ due to the fact that $\sum_{i=1}^{|X\setminus M|} c_i^2 = ||\phi_{01}\rangle|^2 = 1-|M|/|X|$ (19), and comparing this bound with (22) completes the proof. ■

Remark 13 *The original proof of this theorem (Lemma 6 in [Sz2]) is quite sketchy. That proof implicitly uses that* (21) *is zero, however this is not even explicitly stated. I could not find a immediate argument that derives this equality and is based on the surrounding statements. Two relevant property listed there probably meant to prove this equality are $\langle\phi_{01}|\phi_{02}\rangle = 0$, and that these vectors have real coefficients. However this is not enough, consider the following simple example, where U is a simple reflection:*
Example

$$|\psi_1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix},\ |\psi_2\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix},\ U = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \Rightarrow U|\psi_1\rangle = |\psi_2\rangle, U|\psi_2\rangle = |\psi_1\rangle$$

$$\textit{then } \langle\psi_1|\psi_2\rangle = 0 \textit{ holds, but } \langle U\psi_1|\psi_2\rangle + \langle U\psi_2|\psi_1\rangle = \langle\psi_2|\psi_2\rangle + \langle\psi_1|\psi_1\rangle \neq 0$$

I used the stronger observation that $|\phi_{01}\rangle \in T_R^M$, $|\phi_{02}\rangle \in (T_R^M)^\perp$ and introduced Lemma 7 which made me able to prove that (21) *is indeed zero.*

4.3. Relation between Classical and Quantum Hitting Times

Now we would like to relate Classical and Quantum Hitting Times. I can state the main theorem of [Sz2] in a slightly more general form an with an improved constant.

Theorem 14 *For every symmetric Markov chain P on state set X, and for every marked set $M \subseteq X$ with $|M| \leq |X|/2$ the Quantum Hitting Time is quadratically smaller than the Classical Hitting Time:*

$$\mathcal{H}(W_P, M) < 3\sqrt{\mathcal{H}(P, M)}$$

Proof We use the notations from the proof of the previous theorem. The upper bound on the Quantum Hitting Time can be further bounden by Jensen's inequality, using that $\sqrt{\ }$ is a concave function and that $\sum_{i=1}^{|X\setminus M|} \tilde{c}_i^2 = 1$:

$$\sum_{\tilde{c}_i \neq 0} \tilde{c}_i^2 \sqrt{\frac{1}{1-|\lambda_i|^2}} \leq \sqrt{\sum_{\tilde{c}_i \neq 0} \tilde{c}_i^2 \frac{1}{1-|\lambda_i|^2}} \leq \sqrt{\frac{1}{1-|M|/|X|}} \sqrt{\sum_{c_i \neq 0} c_i^2 \frac{1}{1-\lambda_i}} \leq \sqrt{2}\mathcal{H}(P, M)$$

The second inequality is due to Lemma 10 using the fact that $\tilde{c}_i = c_i/\sqrt{1-|M|/|X|}$ and the third inequality comes from Theorem 8 and using that $|M|/|X| \leq 1/2$. Finally using Theorem 12 and that $2\sqrt{2} < 3$ we get the sought inequality of the theorem. ■

Note that the amended proofs I presented in this thesis made me able to remove the irreducibility and aperiodicity assumptions on P, as well as improved the constant, which was around 100 in [Sz2].

4.4. A $\delta\epsilon$ bound on hitting times

We already have nice expressions for both Classical and Quantum Hitting Times, however they heavily depend on the actual marked set M. This makes us think, can we prove some sort of a uniform bound? The answer of [Sz1] is that if we have uniform lower bound on the size of M, then we get a uniform upper bound for the hitting times. Before stating the lemma we need some preparation.

Definition ***(Spectral Norm, Eigenvalue Gap)*** For a matrix P let the eigenvalues be $\lambda_i : 1 \leq i \leq k$ listed with multiplicity in decreasing order: $|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_k|$. Then the spectral norm of P denoted by $\|P\|_2$ is $|\lambda_1|$. The eigenvalue gap is defined as $|\lambda_1| - |\lambda_2|$.

We will also need a simple linear algebraic theorem which is at the heart of the theory for discrete time, finite Markov chains:

Theorem 15 (Perron-Frobenius theorem for positive matrices) *Let $P \in (\mathbb{R}^+)^{n\times n}$ then it has a unique largest eigenvalue $\lambda \in \mathbb{R}^+$ which is equal to the spectral norm of P: $\lambda = \|P\|_2$. The corresponding eigenvector can be chosen to be positive coordinate vise.*

The proof is elementary, and can be found in many textbooks, for example in [Mey].

We can also apply a simple limiting argument to show that for non-negative matrices there is a non-negative eigen-pair (λ, ρ) with λ being equal to the spectral norm. Now let P be transition matrix describing our Markov chain. Then we can assume ρ is also normalised such that $\|\rho\|_1 = 1$. Since ρ describes a probability distribution and P is stochastic we have $1 = \|\rho P\|_1 = \lambda v$ implying $\lambda = 1$. So $\|P\|_2 = 1$, and the eigenvalue gap of P is $1 - |\lambda_2|$, where λ_2 is the second largest eigenvalue of P (the eigenvalues are again counted with multiplicity). It is easy to see that this eigenvalue gap controls the speed of the convergence to the limiting distribution of P.

Lemma 16 *Let P be a symmetric Markov chain on state set X, let δ be the eigenvalue gap of P and $M \subseteq X$ such that $|M| > \epsilon|X|$. Then the spectral norm of P_M is at most $1 - \delta\epsilon/2$.*

Proof Let us denote $|X| = n$. Since P_M is non-negative, we have a non-negative eigenpair (λ, ρ), such that $\lambda = \|P_M\|_2$. If $\lambda = 0$ we are done, else suppose that ρ has unit length: $|\rho| = 1$. Since P is symmetric we have an orthonormal system of eigenvalues $(\lambda_i, v_i) : 1 \leq i \leq |X|$ for P. We can further assume $\lambda_1 = 1$ and $v_1 = \sqrt{1/n}\mathbb{1}$. Using this we can decompose ρ: $\rho = \sum_{i=1}^n \alpha_i v_i$, where $\sum_{i=1}^n \alpha_i^2 = 1$ because of the orthonormality. Further using that both ρ and $P - P_M$ are non negative, we have:

$$\lambda^2 = |\rho P_M|^2 \leq |\rho P|^2 = \sum_{i=1}^n \alpha_i^2\lambda_i^2 \leq \alpha_1^2 + \sum_{i=2}^n \alpha_i^2(1-\delta)^2 \leq \alpha_1^2 + (1-\delta)\sum_{i=2}^n \alpha_i^2 = 1 - \delta\sum_{i=2}^n \alpha_i^2$$

Now we should estimate $\sum_{i=2}^n \alpha_i^2$. We use that ρ has at least ϵn zero coordinates:

$$\alpha_1 = \langle \rho, \sqrt{1/n}\mathbb{1}\rangle = \sqrt{1/n}\langle \rho, \mathbb{1}_{X\setminus M}\rangle \leq |\rho|\sqrt{1/n}|\mathbb{1}_{X\setminus M}| \leq \sqrt{1-\epsilon}$$

Hence $\sum_{i=2}^n \alpha_i^2 = 1 - \alpha_1^2 \geq \epsilon$ and finally $\lambda^2 \leq 1 - \delta\epsilon \Rightarrow \lambda \leq 1 - \delta\epsilon/2$ ∎

Corollary 17 (A $\delta\epsilon$ bound) *For any symmetric Markov chain with eigenvalue gap δ, if the proportion of marked elements is at least ϵ, then $\mathcal{H}(P, M) \leq 2/\delta\epsilon$ and*

$$\mathcal{H}(W_P, M) \leq 2/\sqrt{\delta\epsilon}$$

Proof For the classical case consider Theorem 8, for the quantum case compare Theorem 12 with the expression in the last line of the previous proof: $\lambda^2 \leq 1 - \delta\epsilon$. ■

Note that this estimate might be trivial (∞), i.e. the eigenvalue gap δ can be zero for example when the second largest eigenvalue is in fact -1. This occurs for example when the chain is periodic, in such cases the Markov chain may still mix well. In such case we can get rid of the negative eigenvalues by replacing P with the lazy walk matrix $(Id + P)/2$ to get a non trivial estimate on the hitting times.

5. Algorithmic implications of Szegedy's quantisation method

We have deeply analysed the relationship between the hitting times of classical Markov chains and their quantised version. Now let us see how can we use it for constructing and analysing quantum algorithms.

5.1. The Detection Problem

In the case of classical walks we get an element of M with high probability in running time $2\mathcal{H}(P, M)$. It follows from the Markov inequality: $P(\text{we did not reach } M \text{ after } 2\mathcal{H}(P, M) \text{ steps}) \leq 0.5$.

However in the quantum case we only now that $\left|W_{P_L}^t \left|\phi_0\right\rangle - \left|\phi_0\right\rangle\right|^2$ becomes big in average after $\mathcal{H}(W_P, M)$ repetitions. It does not directly imply that we also get an element of M after a measurement. But it does make us able to decide whether there is a marked set in $\mathcal{H}(W_P, M)$ time. First we define the corresponding problem:

Definition ***(Detection Problem)*** Let $\mathcal{M} \subseteq 2^X$ be a system of non-empty subsets of X. Given that the marked set M is either empty or $M \in \mathcal{M}$ decide which is the case.

To solve this problem in the classical case we are going to use the leaking walk P_L, where P_L is the leaking walk corresponding to the (possibly empty) marked set M. In the quantum case we use its quantised version W_{P_L}. More precisely we are going to use the controlled version of it: c-W_{P_L} defined by

$$\begin{aligned} |0\rangle_C |x\rangle |y\rangle &\rightarrow |0\rangle_C (W_{P_L} |x\rangle |y\rangle) \\ |1\rangle_C |x\rangle |y\rangle &\rightarrow |1\rangle_C |x\rangle |y\rangle \end{aligned}$$

Where the first qubit $|i\rangle_C$ is an additional control bit. By linearity the operator is well defined. For simplicity now assume that instead of M we are given directly by a "black-box" implementing c-W_{P_L}. We will address the question of implementation in the next Section.

Theorem 18 *Let $T_{MAX} = \underset{M\in\mathcal{M}}{MAX}(\mathcal{H}(P, M))$ then we can solve the detection problem classically within $2T_{MAX}$ steps and quantumly within $3\sqrt{T_{MAX}}$ steps with bounded one side error.*

Proof In the classical case we run P for $2T_{\text{MAX}}$ steps, checking in each step whether $x \in M$ and say that $M \in \mathcal{M}$ if we find some marked element during the steps of the random walk, otherwise we say $M = \varnothing$. If $M \in \mathcal{M}$ it is an easy consequence of the Markov inequality that we find some $m \in M$ with probability at least 1/2 - as we observed at the beginning of the Section. It is also clear that if $M = \varnothing$ then this process also output $M = \varnothing$, so the error is one-sided.

For the quantum case we construct a process using the controlled quantum walk operator cc-W_{P_L}. We amend the space of two walk registers $|x\rangle |y\rangle$ with an additional control qubit, and start our process from the state $|\phi_0\rangle$ defined like before, with the control bit set to zero. First we apply the Hadamard transformation (1) on the control bit. After that we do the actual walk steps (c-W_{P_L})^t for a randomly selected $1 \leq t \leq 3\sqrt{T_{\mathrm{MAX}}}$ and finally undo the Hadamard transformation:

$$|0\rangle_C |\phi_0\rangle \xrightarrow{H} 1/\sqrt{2}(|0\rangle_C + |1\rangle_C) |\phi_0\rangle \overset{(c-W_{P_L})^t}{\longrightarrow} 1/\sqrt{2}(|0\rangle_C (W_{P_L} |\phi_0\rangle) + |1\rangle_C |\phi_0\rangle) \overset{H=H^{-1}}{\longrightarrow}$$
$$1/2 |0\rangle_C (W_{P_L} |\phi_0\rangle + |\phi_0\rangle) + 1/2 |1\rangle_C (W_{P_L} |\phi_0\rangle - |\phi_0\rangle) \tag{24}$$

At the end of the process we measure the control bit, and the left walk register. If either the control register is 1 or the element found in the left walk register is in M, then we say $M \in \mathcal{M}$. Otherwise we say $M = \varnothing$.

If $M \in \mathcal{M}$ we consider two sub-cases:

If $|M| > 1/2|X|$ then we find an element in the left walk register with probability at least 1/4. For this observe that the last step of (24) is a local transformation on the control bit, so it does not affect the measurement statistics of the left walk register. In the state just before the last step the states $1/\sqrt{2}\,|0\rangle_C (W_{P_L} |\phi_0\rangle)$, $1/\sqrt{2}\,|1\rangle_C |\phi_0\rangle$, do not interfere since they are orthogonal. The probability that the measurement outcome of the left walk register $|\phi_0\rangle$ yield some $m \in M$ is at least 1/2 since $|M| > 1/2|X|$. So the overall probability of measuring some $m \in M$ in the left walk register is at least 1/4.

If $|M| \leq 1/2|X|$ then $3\sqrt{T_{\mathrm{MAX}}} \geq \mathcal{H}(W_P, M)$ according to Theorem 14. I.e. by definition $|W_{P_L}^t |\phi_0\rangle - |\phi_0\rangle|^2 \geq 1 - |M|/|X| \geq 1/2$ averaged over all $1 \leq t \leq 3\sqrt{T_{\mathrm{MAX}}}$ so $|1/2\,|1\rangle_C (|\phi_0\rangle - W_{P_L} |\phi_0\rangle))|^2 \geq 1/8$ in average as well. This implies that the measurement probability of 1 in the control register is at least 1/8 in this case.

We need to show that the error is one sided. Suppose that $M = \varnothing$. Then the discriminant matrix for $W_{P_L} = W_P$ is simply P. And as $P = P^T$ we have $\mathbb{1}P = \mathbb{1}$. Since $|\phi_0\rangle$ was generated according to the uniform distribution $1/\sqrt{|X|}\mathbb{1}$ on the transition states, using the Real Spectral Lemma from Section 3 we get that $W_P |\phi_0\rangle = |\phi_0\rangle$ resulting that the probability of measuring 1 in the control register is 0 in the case when $M = \varnothing$. ■

Corollary 19 (The $\sqrt{\delta\epsilon}$ rule) *Let P be a symmetric Markov chain with eigenvalue gap δ. If for all $M \in \mathcal{M}$: $|M| > \epsilon|X|$ then we can solve the detection problem in time $\mathcal{O}(1/\sqrt{\delta\epsilon})$ with the quantum walk algorithm.*

This is a consequence of Corollary 17.

5.2. High level description of the implementation vs. the quantum query model

The implementation of the actual process depends heavily on the Markov chain we are quantising. To keep things simple but general enough we break down the implementation of the process to the usage of 3 unified unitary operations:

Setup: Creates the starting state from the initial zero state according to the stationary distribution τ of the Markov chain P:

$$S : |0\rangle\,|0\rangle \rightarrow 1/\sqrt{\pi_x}\,|x\rangle\,|0\rangle$$

Update: In fact it covers 4 different operators, one which generates the right transition states, and its symmetric pair: (we use notations introduced at the beginning (5))

$$\begin{aligned} U_R = \; & |x\rangle\,|0\rangle & \rightarrow \; & |r_x\rangle \\ U_L = \; & |0\rangle\,|y\rangle & \rightarrow \; & |l_y\rangle \end{aligned}$$

together with their inverses U_R^{-1}, U_L^{-1}.

Check: Checks whether the left (right) walk register contains an element of M, and conditional on this changes the value of an additional register qubit $|i\rangle_M$ used for storing this information temporarily:

$$\begin{aligned} C_L : \; & |x\rangle\,|y\rangle\,|j\rangle_M & \rightarrow \; & |x\rangle\,|y\rangle\,|j \oplus_{x\in M} 1\rangle_M \\ C_R : \; & |x\rangle\,|y\rangle\,|j\rangle_M & \rightarrow \; & |x\rangle\,|y\rangle\,|j \oplus_{y\in M} 1\rangle_M \end{aligned}$$

where $j \oplus_{x\in M} 1$ is the conditional binary addition, if $x \in M$ it flips the value of j, otherwise does keep the value of j.

Remembering that we need to implement the conditional walk operator we indicate that we also have a control qubit. Using the above operators we can generate the starting state of the process from the zero state: $|0\rangle_C\,|0\rangle\,|0\rangle\,|0\rangle_M \xrightarrow{U_R S} |0\rangle_C\,|\phi_0\rangle\,|0\rangle_M$

We will implement the reflections in the style of the Grover Amplitude Amplification algorithm. To do so we need the controlled phase flip operator. It does flip the sign of the amplitude for all zero vector on the right (left) walk register controlled by both the control and marked qubits:

$$\begin{aligned} \text{c-}F_R : \; & |0\rangle_C\,|x\rangle\,|0\rangle\,|0\rangle_M & \rightarrow \; & -\,|0\rangle_C\,|x\rangle\,|0\rangle\,|0\rangle_M \\ \text{c-}F_L : \; & |0\rangle_C\,|0\rangle\,|y\rangle\,|0\rangle_M & \rightarrow \; & -\,|0\rangle_C\,|0\rangle\,|y\rangle\,|0\rangle_M \end{aligned}$$

and acts as the identity on its orthogonal complement. (In fact the controlled flip operator is just a flip operator acting on the larger space amended by the two control registers.)

Using the reduced form of (18) we can implement the controlled walk operator using the controlled reduced leaking reflections:

$$\begin{aligned} -\text{c-ref}_{T_R^M} &= U_R C_L \text{c-}F_R C_L U_R^{-1} \\ -\text{c-ref}_{T_L^M} &= U_L C_R \text{c-}F_L C_R U_L^{-1} \end{aligned}$$

$$\text{c-}W_{P_L} = \text{c-ref}_{T_L^M} \cdot \text{c-ref}_{T_R^M} = (-\text{c-ref}_{T_L^M}) \cdot (-\text{c-ref}_{T_R^M})$$

We are going to count only the calls to the operators above, and assume that they are implemented by some quantum black box. This is the so called query model - a query is an application of a unitary implemented by the black box. We will shortly denote the costs of the 3 different type of queries S, U, C correspondingly. All other operation is considered to be free.

The costs S, U, C do always depend on the particular problem, but this query model does make the analysis of our search algorithms clear, simple and general. We can argue for this model by that including all the steps we drop this way seem to never change significantly the complexity of the search algorithm we consider. There might be some additional polylog(n) factor though.

We will sometimes use walks with some data stored at the vertices in order to make checking easier. In this case we always maintain some data structure attached to the vertices. So we are going to work with states $|d_x\rangle_D |x\rangle |y\rangle |d_y\rangle_D$. Then we do everything according to the natural isomorphisms of the left $|d_x\rangle_D |x\rangle \leftrightarrow |x\rangle$ and right $|y\rangle |d_y\rangle_D \leftrightarrow |y\rangle$ walk registers.

This isomorphisms connect the cases when we store some data and when we do everything as before at an abstract level, implying that the overall analysis of the search algorithm do not change. However the costs of the various operations might change so in this case we could use the notation S_d, U_d, C_d for the costs of the corresponding operations which also include the time needed to manipulate the data. For simplicity we only use the notations S, U, C, but if there is a data structure involved then this notation will refer to the costs S_d, U_d, C_d.

This trick of including some data in the description usually reduces the cost of checking, but makes the update operator harder to implement: $|d_x\rangle_D |x\rangle |0\rangle |0\rangle_D \to |d_x\rangle_D |r_x\rangle |d_{r_x}\rangle_D$ (For simplicity we assume that the data belonging to the 0 vector is also 0.)

5.3. The Element Distinctness Problem as a consequence

Szegedy originally defined the quantisation described in this thesis inspired by the Element Distinctness Algorithm of Ambainis [Amb3]. Ambainis originally introduced the quantum walk based search method for this problem in 2004.

This was the first search algorithm which did go beyond the Grover search. It also shows why the $\sqrt{\delta\epsilon}$ rule of Corollary 19 is useful. If we consider that ϵ is the proportion of marked elements, then we may ask why is it interesting at all? Grover search should solve the problem in $1/\sqrt{\epsilon}$ time which is even better then the time $1/\sqrt{\delta\epsilon}$ of Corollary 19. However we show that this is not the case for element distinctness.

The question for the element distinctness problem itself is quite simple, we are given numbers $x_1, x_2, \cdots, x_n \in [m]$ are they all distinct? Classically we can formulate this question like this: we are given a function $f : [n] \to [m]$ find out whether there is a pair of indices $1 \leq k, l \leq n$ such that $k \neq l$ but $f(k) = f(l)$.

Classically we would need to check the value of the function on all the inputs in order to be able to decide the question. Then sorting the values we did get would solve the problem with n queries and in overall $n\log(n)$ running time. If we would like to solve it only with high probability we would still need to query a constant percentage of the elements.

However we can perform better in the quantum case. In order to develop quantum algorithms for solving the problem we should define it in terms of some unitary operator, i.e. present as a quantum circuit. In the classical version we are given a black box which, given the number $in \in [n]$ encoded by some binary string calculates and outputs

$f(in) \in [m]$ again encoded as some binary string.

Then the corresponding quantum black box works just like the standard quantised version of this classical circuit (see Section 1.2). Using this quantum black box we can formulate the quantum version of the previous decision problem:

Definition ***(Element Distinctness Problem)*** Suppose we are given a unitary operation U_f (in the form of a quantum black box) which for a given input string $|\text{in}\rangle$ computes the value of $f(\text{in})$ and puts it into the output register, formally:

$$U_f : |\text{in}\rangle\,|\text{out}\rangle \to |\text{in}\rangle\,|\text{out} \oplus f(\text{in})\rangle$$

Find out whether there is some $k \neq l$ such that $f(k) = f(l)$ i.e. $U_f\,|k\rangle$ acts on the output register identically to $U_f\,|l\rangle$.

First we could try to solve it using Grover Search. For example we could form pairs of inputs $|\phi_1\rangle\,|\phi_2\rangle$ and do Grover Search starting from the uniform superposition over all pairs. Then we could use two queries to find out whether f on a pair of inputs is distinct. But if we have only one non-distinct pair then the proportion of pairs we are looking for is $\frac{2}{n^2}$ resulting in an overall $\mathcal{O}(n)$ quantum algorithm, which is no better then if we would simply query all possible inputs. Or we could try to use sets of r inputs. Then the setup cost would be roughly r Then to decide if an r-tuple contains such a pair we are looking would need r query. While in the presence of a unique sought pair the proportion of good r-tuples would be $\binom{n}{r-2}/\binom{n}{r} = \frac{r(r-1)}{(n-r+2)(n-r+1)} \approx r^2/(n-r)^2$, resulting in $\mathcal{O}(r + \frac{n-r}{r} \cdot r)) = \mathcal{O}(n)$ running time using Grover. In fact using a clever two-level construction combined with Grover search it can be solved using $\mathcal{O}(n^{\frac{3}{4}})$ queries [Buh], but it is still far from the optimal solution.

The trick of Ambainis is that he uses a spatial search algorithm on the Johnson graph. The Johnson graph is a graph with vertex set $\binom{n}{r}$, where two edges are connected iff the corresponding r-tuples differ only in one element. Here the value of the locality of the search is apparent: during the algorithm Ambainis stores the values of f for the elements of the r-tuples, and since in every step we move between neighbouring vertices i.e. the r-tuples change only by one element, one query suffices to update data. This reduces the number of queries required by on step of the walk from r to $\mathcal{O}(1)$.

The walk he uses is a uniform walk on the Johnson graph i.e. the transition probabilities on all edges are $\frac{1}{r(n-r)}$. Since this graph is $r(n-r)$-regular the matrix of this uniform walk is simply the adjacency matrix divided by $r(n-r)$. The eigenvalues of the adjacency matrix of the Johnson graph are studied for example in [Knuth] giving that the eigenvalue gap $\delta = \frac{n}{r(n-r)}$ which simplifies to $\delta = \Omega(\frac{1}{r})$ if $r \leq n/2$ (see also [Bel]). If there is a pair $k \neq l$ but $f(k) = f(l)$ then the proportion of good r-tuples is in the order r^2/n^2 if $r \leq n/2$ as we already calculated, i.e. $\epsilon \geq r^2/n^2$. Then Corollary 19 - the $\sqrt{\delta\epsilon}$ rule tells us that running the walk for at most $n/\sqrt{r}$ steps makes us able to answer the element distinctness question with high probability. Considering that one step of the walk did use $\mathcal{O}(1)$ queries, summing them up together with the setup cost yields the overall number of queries: $\mathcal{O}(r + n/\sqrt{r})$. This expression is minimised when $r = n^{\frac{2}{3}}$, and gives the $\mathcal{O}(n^{\frac{2}{3}})$ bound on queries, proving:

Theorem 20 (Ambainis) *There is a quantum query machine deciding the element distinctness problem with high probability using $\mathcal{O}(n^{\frac{2}{3}})$ queries.*

This upper bound on the problem is tight as meets the lower bound given by Aaronson and Shi [ASh] in 2002.

Looking at the proof we see that it can be easily generalised to the case when instead of pairs we are looking for k-tuples having the same value of f. The only thing we should adjust is the proportion of marked r-tuples when we have at least one such k-tuple, yielding $\Omega(r^k/n^k)$. Therefore we get an overall bound $\mathcal{O}(r + \sqrt{n^k/r^{k-1}})$ on queries optimised when $r = n^{k/k+1}$. So we get the generalised version of the previous theorem:

Theorem 21 (Ambainis) *There is a quantum query machine deciding the element k-distinctness problem with high probability using $\mathcal{O}(n^{\frac{k}{k+1}})$ queries.*

5.4. Finding a marked element, quantum

If we have good control over the structure of X then using binary search we can also find an element using the detection algorithm as a subroutine. However it will introduce an additional $\log(|X|)\log(\log(|X|))$ factor in the time complexity. The $\log(|X|)$ factor comes from the binary structure of the search, and the $\log(\log(|X|))$ factor comes from the need for a more precise detection required to keep the summed errors low during the whole process.

Szegedy showed [Sz2] that if the marked set consist of only one element, and the Markov chain is also state transitive, then we can find the unique marked element with good probability.

Theorem 22 (Finding a unique element) *Let P be a symmetric, state transitive Markov chain and suppose that there is a unique marked element $z \in X$. If $T \geq 10\sqrt{\frac{1}{1-\|P_{\{z\}}\|_2}}$ and we chose some $t \in [1, T]$ uniformly at random then the probability that measuring the second register of $W^t_{P_L}|\phi_0\rangle$ gives z is $\Omega(|X|/\mathcal{H}(P, \{z\}))$.*

This theorem gives nice results for the d dimensional torus $[n]^d$. If $d \geq 3$ then the classical hitting time is $\mathcal{O}(n^d)$ whenever $M \neq 0$. Further if we have a unique marked element z, then using at most $\mathcal{O}\left(n^{d/2}\right)$ steps of the quantum walk we also find this unique element with probability $\Omega(1)$.

A famous early result of Kempe et al. [SKW] about the hypercube becomes an easy consequence as well. Kempe showed that searching for a unique element on the hypercube can be solved in $\mathcal{O}(\sqrt{n})$ steps. Historically it was one of the first positive results about spatial quantum search. To prove this result using our machinery we first introduce a theorem about hitting times [Lov1]:

Theorem 23 *Let $G = (V, E)$ be an undirected connected graph, and let P be the uniform walk on it, i.e. the transition probability from any vertex v to any adjacent vertex is $1/d(v)$. Let $\mathcal{H}(s, t)$ be the expected number of steps needed to reach t starting from s, and* *let τ be the stationary distribution. Finally let $1 = \lambda_1 > \lambda_2 \ldots > \lambda_{|V|}$ be the eigenvalues of P listed with multiplicity, then the following holds:*

$$\sum_{t \in V} \tau_t \mathcal{H}(s,t) = \sum_{k=2}^{|V|} \frac{1}{1-\lambda_k} \tag{25}$$

The hypercube has a rich automorphism group, for example flipping the values on any set of coordinates gives an automorphism. In particular if we select the set of coordinates for which s and t differ, then we get an automorphism which interchanges s and t. It shows that $\mathcal{H}(s,t) = \mathcal{H}(t,s)$ Moreover the uniform walk on the hypercube is also symmetric because it is a regular graph. So the left hand side of (25) coincides with the classical hitting time for a unique marked element s in the case of the hypercube.

To bound the right hand side we need to examine the eigenvalues of the hypercube. It can be easily checked that the d-dimensional hypercube has a set of mutually orthogonal (eigenvector, eigenvalue) pairs which can be expressed in terms of the subsets of $[d]$

$$\forall S \subseteq [d] \;\; v_S := \sum_{b \in \{0,1\}^d} \prod_{i \in S} (-1)^{b_i} e_b, \quad v_S P = \frac{d - 2|S|}{d} v_S; \quad v_{S'} \cdot v_S^T = \delta_{S'S}$$

Using the exact form of the eigenvalues we can bound the hitting time as follows:

$$\begin{aligned}\sum_{k=2}^{2^d} \frac{1}{1-\lambda_k} &= \sum_{m=1}^{d} \binom{d}{m} \frac{d}{2m} \leq 2 \sum_{m=1}^{d/2} \binom{d}{m} \frac{d}{2m} \leq d \sum_{m=1}^{d/8} \binom{d}{m} + 8 \sum_{m=d/8}^{d/2} \binom{d}{m} \\ &\leq d 3^{-d/8} \sum_{m=d/8}^{d/4} \binom{d}{m} + 8 \cdot 2^d \leq (d3^{-d/8} + 8) 2^d \leq c \cdot 2^d \text{ for some universal } c \in \mathbb{R}^+\end{aligned} \tag{26}$$

So we get that for any vertex v of the hypercube $\mathcal{H}(P, \{v\}) = \Omega(n)$, where $n = 2^d$. As the hypercube is state transitive Theorem 22 implies that we can find a unique marked vertex with high probability in time $\mathcal{O}(\sqrt{n})$.

In the case of the Element Distinctness Algorithm Theorem 22 do not help much. However the walk on the Johnson graph is state transitive, the problem is that the vertex we are looking for is fundamentally not a single one, but a bunch of subsets containing a non distinct pair.

These results make us feel, that the finding problem is much more complicated in general than the detection problem. However there were much progress about this question afterwards. In 2006 Magniez, Nayak, Roland and Sántha pushed forward the unified approach of Szegedy by constructing their search algorithm which became known as the MNRS search. The MNRS algorithm improved various aspects of Szegedy's approach. In particular it can find marked elements under more general conditions. In the next Section we give an overview of their results.

6. Implementation of the Quantum Walk Algorithm

In this Section we study the question of implementation of the walk algorithms according to the generalised framework introduced in [MNRS]. In particular we show how to implement Ambainis's Element Distinctness algorithm along the lines of this generalised framework. A possible implementation method was described in the original article [Amb2], however I do not know about improvements on it. This original method gave an $\mathcal{O}(\log^4(N))$ time implementation of the walk steps, while we improve it to $\mathcal{O}(\log(N)\log\log(N))$. More importantly our implementation method is conceptually simpler and also fits well into the new general framework. We also introduce some subroutines/techniques which may be applied to further problems as well.

In the Quantum walk based search one usually stores some data attached to the states which makes easier to check whether a specific state is one we are seeking for. Then the graph usually corresponds to some underlying structure where adjacent vertices correspond to states which have similar data attached. Transitions between adjacent states is preferred since it is simpler to keep track of the changes in the attached data. This requirement of local transitions (spatiality) distinguishes the Quantum walk based search from the unstructured Grover Search.

Dealing with the data attached to the vertex states is a crucial part of the whole algorithm. To include the data structure into this picture elegantly one can simply treat it as part of the (vertex)states, as discussed in 5.2 following [MNRS]. It results in some sense in the duplication of the data stored, and is considered in the [MNRS] paper for simplicity in the description. However we will show that this point of view is not only useful at an abstract level but it may also make the implementation easier.

6.1. Quantum data structures - challenges

Keeping track of the changes in the attached data is an essential part of the quantum walk based search algorithms. However implementing data structures in such quantum algorithms can be challenging (see [Amb2]). There are three main difficulties for the quantum data structure to deal with, which is not present in the classical setting:

- If we store some data then we need to store it exactly the same way regardless of the previous operations we used to construct and modify it.
- All modification of the data structure should be reversible.
- All calculations should end after a constant number of steps, algorithms that work only in good average time are not directly applicable.

The first requirement is in order to let interference occur: $0 = \alpha\left|y\right\rangle\left|d_y\right\rangle - \alpha\left|y\right\rangle\left|d_y\right\rangle$ would normally interfere. But if the data is stored with slightly different structure, then they will not cancel, however we would like them to do so. E.g. let the data stored in some binary tree structure which differ in just the tree representation $\left|t\right\rangle \neq \left|t'\right\rangle$: $0 \neq \alpha\left|y\right\rangle\left|d_y\right\rangle\left|t\right\rangle - \alpha\left|y\right\rangle\left|d_y\right\rangle\left|t'\right\rangle$

The second requirement is a fundamental feature of quantum computing, and together with to the first one is especially challenging. Even one of the most basic operation of data structures: sorting is not allowed in its standard form.

The third requirement is because we want to work with superpositions. Even if only a tiny bit of the state needs more calculation time we can not stop earlier. We can not split operations according to superpositions. In other worlds we need an actual hardware (quantum circuit) built in advance to implement the processes, where the worst running time is the depth of the circuit.

6.2. Elementary operations and their complexity

We will need to use some elementary operations, so we shortly describe here some examples and their complexity in the quantum case. As we already mentioned in Section 1.2 any classical circuit may be converted by a relatively low overhead.

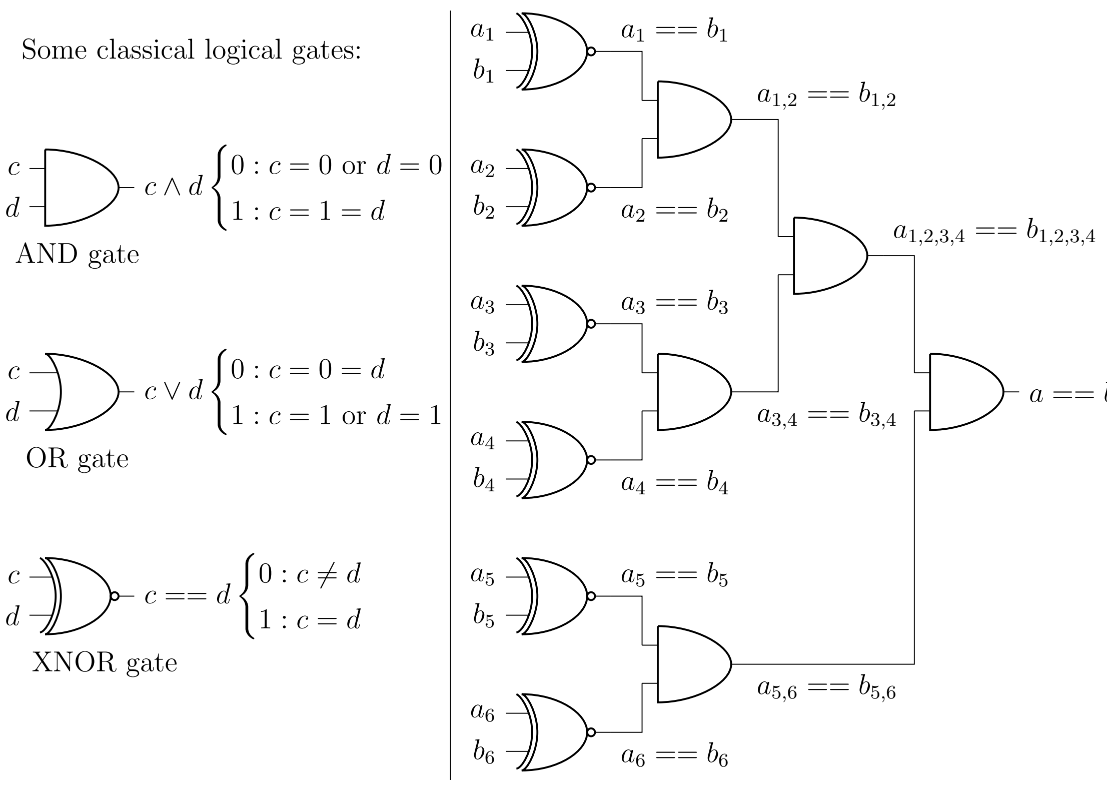

Some classical logical gates:
c
d
$c \wedge d \begin{cases} 0 : c = 0 \text{ or } d = 0 \\ 1 : c = 1 = d \end{cases}$
AND gate
c
d
$c \vee d \begin{cases} 0 : c = 0 = d \\ 1 : c = 1 \text{ or } d = 1 \end{cases}$
OR gate
c
d
$c == d \begin{cases} 0 : c \neq d \\ 1 : c = d \end{cases}$
XNOR gate
a_1
b_1
$a_1 == b_1$
$a_{1,2} == b_{1,2}$
a_2
b_2
$a_2 == b_2$
$a_{1,2,3,4} == b_{1,2,3,4}$
a_3
b_3
$a_3 == b_3$
$a_{3,4} == b_{3,4}$
$a == b$
a_4
b_4
$a_4 == b_4$
a_5
b_5
$a_5 == b_5$
$a_{5,6} == b_{5,6}$
a_6
b_6
$a_6 == b_6$

Figure 2: Notation for some classical logical gates and a Boolean circuit built from them which decides whether the two 6 bit numbers a, b are equal using 6 XNOR (NOT XOR) gates and a binary tree of AND gates of depth $3 = \lceil \log_2(6) \rceil$. See also Figure 3.

More concretely we may assume the classical circuit consists of for example only AND, OR and XNOR gates, since this is a universal set of logical gates. Then we may replace these gates with their quantum analogue, where each new quantum gate introduce a new output qubit. For an example consider the case when we want to implement a circuit which decides whether two binary numbers are equal. Then the classical circuit may described as seen in Figure 2 and its quantised version as depicted in Figure 3.

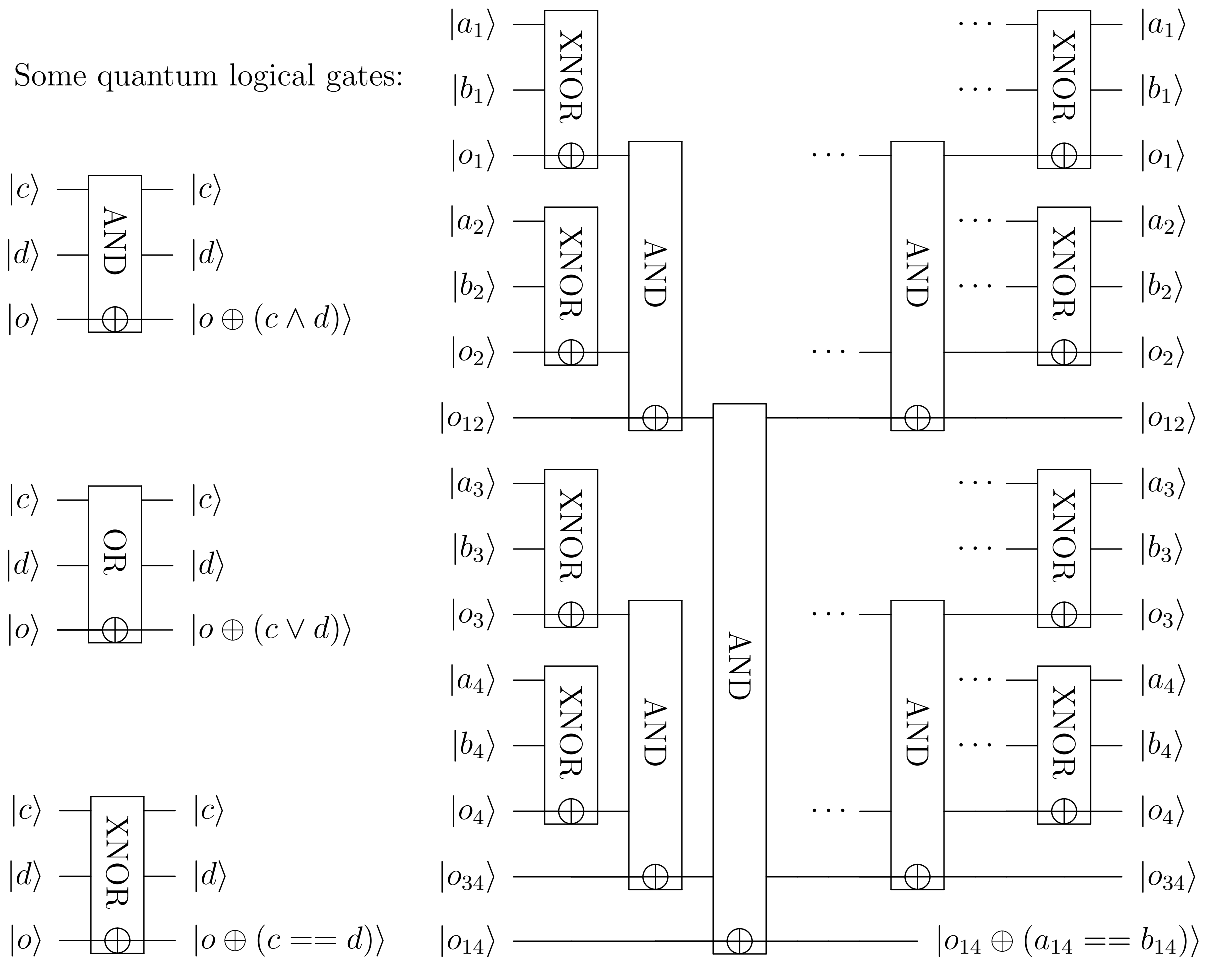


Figure 3: Some quantum (reversible) logical gates, and a quantum circuit built from them which decides whether the two 4 bit numbers a, b are equal using 4 XNOR (NOT XOR) gates and a binary tree of AND gates of depth $2 = \lceil \log_2(4) \rceil$. Compare with Figure 2. It uses additional workspaces qubits $|o_1\rangle$, $|o_2\rangle$, $|o_3\rangle$, $|o_4\rangle$, $|o_{12}\rangle$, $|o_{34}\rangle$ which are assumed to have value zero initially. Note that after the output $|o_{14}\rangle$ is evaluated we uncompute these workspace qubits by applying the circuit (except the last AND gate) in reversed order. We need to uncompute these workspace qubits to avoid unwanted entanglement.

This example also illustrates that this way of quantising a circuit does not change the number of gates nor the circuit depth or at most doubles them if we want to uncompute auxiliary workspace qubits. (Which we usually want to in fact.) The auxiliary workspace qubits are reusable after uncomputing, and their number is not greater then the number AND, OR, XNOR gates in the circle.

Following is a list of elementary operations and their complexities. The input is one or two n bit binary number a, b. The calculation only changes the value of the output

bit or output number, and the value is changed via binary addition $\oplus$. Also note that we identify logical expressions with binary values **true**: 1, **false**: 0. Again see Figures 2,3.
outbit$\oplus$= $(a == b)$**:** Circuit size: $\mathcal{O}(n)$, depth: $\mathcal{O}(\log(n))$, workspace: $\mathcal{O}(n)$.
outbit$\oplus$= $(a > b)$**:** Circuit size: $\mathcal{O}(n)$, depth: $\mathcal{O}(\log(n))$, workspace: $\mathcal{O}(n)$.
outnumber$\oplus$= $(a + b)$**:** Circuit size: $\mathcal{O}(n)$, depth: $\mathcal{O}(n)$, workspace: $\mathcal{O}(n)$.
outnumber$\oplus$= $(a - b)$**:** Circuit size: $\mathcal{O}(n)$, depth: $\mathcal{O}(n)$, workspace: $\mathcal{O}(n)$.
The following operation is reversible too but here we write the output with mod 2^n addition + corresponding to the group $\mathbb{Z}_{2^n}$ instead of the binary $\oplus$ corresponding to $\mathbb{Z}_2^n$:
outnumber+= a**:** Circuit size: $\mathcal{O}(n)$, depth: $\mathcal{O}(n)$, workspace: $\mathcal{O}(n)$.
outnumber−= a**:** Circuit size: $\mathcal{O}(n)$, depth: $\mathcal{O}(n)$, workspace: $\mathcal{O}(n)$.
We close the list with the easiest operation not needing any extra workspace qubits since it may be done with bitwise application of c-NOT gates:
outnumber$\oplus$= a**:** Circuit size: $\mathcal{O}(n)$, depth: $\mathcal{O}(1)$, workspace: 0).

$$c - NOT : |in\rangle\,|out\rangle \to |in\rangle\,|out \oplus in\rangle \tag{27}$$

6.3. Subroutines and techniques used for the implementation

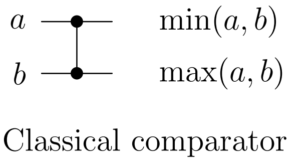
6.3.1. Reversible sorting network

It is a remarkable result in computer science that there exists a sorting network for any number N of elements which has depth $\mathcal{O}(\log(N))$ [AKSz]. Although simple sorting networks are prohibited in quantum computation we can make them easily reversible.

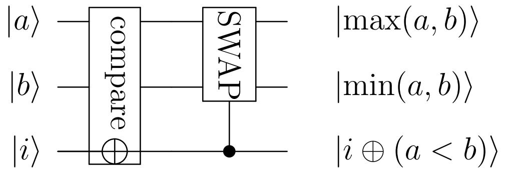
Sorting networks are basically just a bunch of comparators arranged in a clever way. The comparators operate and swap the numbers on the two adjacent lines whenever the number on the top line is greater than the one on the bottom line. We can make this operation reversible by introducing a flag bit for each swap gate. Then the reversible (quantum) gate has two phases:

compare: Flip the value of the flag bit whenever the number on the top line is greater than the other. (Comparison can be done with a circuit of depth $\mathcal{O}(\log(1 + b))$, if we compare numbers of bit length b.)

c-SWAP: Swap the data on the two lines whenever the flag bit is set to 1.



Classical comparator



Note that the above described reversible sorting network has depth $\mathcal{O}(N\log(N)\log(1 + b))$ and need $\mathcal{O}(N(\log(N) + b))$ extra workspace qubits for the implementation. (b is the bit length of the numbers and N is the size of the set to be sorted.) Usually

$\log(N) = \mathcal{O}(b)$, and so the extra workspace $\mathcal{O}(N * b)$ is in the order of the original data storage $N * b$. On the other hand when $b = \mathcal{O}(\log(N))$ then the overall circuit depth is $\mathcal{O}(N \log(N) \log\log(N))$.

6.3.2. Distributing/collecting some data for/after bulk parallel processing

We need several times to do a large number of parallel operations depending only on a few qubits. This can be problematic since in one computational step only one gate can access a particular qubit. In order to let the parallel computation access them in one step we first need to distribute them to the gates which need to read them. If we have k qubits each needed as the (parallel) input of M gates then we may distribute the value of the qubits using a binary tree of c-NOT operation with the additional use of $\mathcal{O}(k * M)$ workspace qubits initially in $|0\rangle$ state. After the parallel computation is done we may uncompute the workspace qubits by reversing the tree. The binary tree has depth $\mathcal{O}(\log(M))$, and we used $\mathcal{O}(k * M)$ extra workspace qubits which is quite reasonable overhead.

The converse operation will be also heavily used by our algorithms. Suppose we have done some massive parallel computation and want to store its result in a predefined register. For example we are interested whether a large set of numbers contains an even number. Then we can calculate a flag bit for each number in the set indicating if the number is even. Then we may collect the value of these indicators using a binary OR tree. (See e.g. Figure 3)

A similar scenario is when we know that there is only one even number in the set, and we want to collect the even number itself. Then we first calculate the flag as before, but now we collect the numbers with k parallel binary tree of OR gates with the extra condition that we only examine/consider a number in a leaf of the binary tree if its flag was set 1. If there was exactly one number that had its flag set 1 then we successfully collect the i-th bit of that number at the root of i-th tree. Finally we also uncompute the tree except probably the root of it where we collected the info.

6.3.3. Generating uniform probability distributions

We will need to generate uniform probability distributions over some numbers $[1, ..., N]$. If $N = 2^s$ for some $s \in \mathbb{N}$ then we can do it with the simultaneous application of s Hadamard gates. Otherwise we may still proceed by generating a uniform distribution over $[1, ..., 2^{\log\lceil N \rceil}]$ and use the exact version of the Amplitude Amplification (Theorem 3) to obtain a uniform distribution over $[1, ..., N]$ – we only need two well tuned controlled rotations in order to implement it. A more general method using $\log(N)$ controlled rotations is described in [GR]. Assuming we have the necessary controlled rotation gates this results in a circuit of depth $\mathcal{O}(\log(N))$ for the generation of such uniform distributions.

6.3.4. Usage of randomised algorithms as a subroutine in quantum computations

Randomised algorithms use some random bit feed as part of their input. Using these random bits the algorithm then computes the result. Assuming the random bits had the right distribution the results will have a nice distribution. From a generalised viewpoint the randomised algorithm transforms the random bit feed distribution into some other distribution over the result.

We will need to generate some non trivial distribution for which natural random algorithms may be used. Since we can turn any randomised algorithm into a quantum one this seems to be good enough for us – however one need to be careful. The random algorithm usually entangles its random bit feed distribution with the result, i.e. we usually get a state like this: $\sum_l \sqrt{p_l}\,|\text{result}_l\rangle\,|\text{random bit pattern}_l\rangle$. This would be ok if we were to measure this state. The problem arises when we want to use this as a subroutine of an algorithm, where we would in fact need the state $\sum_l \sqrt{p_l}\,|\text{result}_l\rangle$.

Our trick is to partially transform the random bits to become a part of the result states and cleverly distribute the remaining part of the random bit feed to obtain a product state: $\sum_l \sqrt{p_l}\,|\text{result}_l\rangle \otimes |\text{remaining random bit pattern}\rangle$. This will be enough for us. (See the implementation of **Setup** below for an example.) One may also figure out some uncomputing process for $|\text{remaining random bit pattern}\rangle$ to get the sought state $\sum_l \sqrt{p_l}\,|\text{result}_l\rangle\,(\otimes|0\rangle)$ exactly.

6.3.5. An application: qRAM

To see the power of the reversible sorting approach we mention that one can use it for implementing a quantum random access gate. A quantum random access gate (qRAM) implements the following transformation:

$$|d_1, d_2, \ldots, d_N\rangle\,|k\rangle\,|v\rangle \longrightarrow |d_1, d_2, \ldots, d_{k-1}, v, d_{k+1}, \ldots, d_N\rangle\,|k\rangle\,|d_k\rangle$$

We are going to store the above state $|d_1, d_2, \ldots, d_N\rangle$ as an array, where the k-th element will be $|d_k\rangle$ amended with an extra flag qubit: $(|d_k\rangle\,|f_k\rangle)$. This flag is normally in the 0 state except while the qRAM operates. So the above state will be represented by $[(|d_1\rangle\,|0\rangle)(|d_2\rangle\,|0\rangle), \cdots, (|d_N\rangle\,|0\rangle)]$.

Now suppose we want to read out the data from the k-th element of our array, we may proceed as follows: (Assume k is given as some binary input $|k\rangle$ of bit length $\log(N)$)

(i) For each index (position) in the array check whether index $= k$ – if so flip the value of the flag bit:

$$[|d_1\rangle\,|0\rangle\,|d_2\rangle\,|0\rangle \cdots |d_N\rangle\,|0\rangle]\,|k\rangle\,|v\rangle \rightarrow$$
$$[|d_1\rangle\,|0\rangle\,|d_2\rangle\,|0\rangle \cdots |d_{k-1}\rangle\,|0\rangle\,\overset{k}{|d_k\rangle}\,|1\rangle\,|d_{k+1}\rangle\,|0\rangle \cdots |d_N\rangle\,|0\rangle]\,|k\rangle\,|v\rangle$$

(ii) Sort the array with reversible sorting network using the flag bits for comparison:

$$\begin{aligned}&[|d_1\rangle\,|0\rangle\,|d_2\rangle\,|0\rangle\cdots|d_{k-1}\rangle\,|0\rangle\,\overset{k}{|d_k\rangle}\,|1\rangle\overset{k}{|d_{k+1}\rangle}\,|0\rangle\cdots\overset{N}{|d_N\rangle}\,|0\rangle]\,|k\rangle\,|v\rangle\to\\&[|d_1\rangle\,|0\rangle\,|d_2\rangle\,|0\rangle\cdots|d_{k-1}\rangle\,|0\rangle\,\underset{k}{|d_{k+1}}\,|0\rangle\rangle\cdots\underset{N-1}{|d_N\rangle}\,|0\rangle\underset{N}{|d_k\rangle}\,|1\rangle]\,|k\rangle\,|v\rangle\end{aligned}$$

(iii) Read out (swap) the sought value from the last element of the array:

$$\begin{aligned}&[\,|d_1\rangle\,|0\rangle\,|d_2\rangle\,|0\rangle\cdots|d_{k-1}\rangle\,|0\rangle\,|d_{k+1}\rangle\,|0\rangle\cdots|d_N\rangle\,|0\rangle\,|d_k\rangle\,\underset{\leftrightsquigarrow}{|1\rangle]\,|k\rangle\,|v\rangle}\to\\&[\,|d_1\rangle\,|0\rangle\,|d_2\rangle\,|0\rangle\cdots|d_{k-1}\rangle\,|0\rangle\,|d_{k+1}\rangle\,|0\rangle\cdots|d_N\rangle\,|0\rangle\,|v\rangle\,|1\rangle]\,|k\rangle\,|d_k\rangle\end{aligned}$$

(iv) Unsort the array using the reversible sorting network:

$$\begin{aligned}&[|d_1\rangle\,|0\rangle\,|d_2\rangle\,|0\rangle\cdots|d_{k-1}\rangle\,|0\rangle\,\overset{k}{|d_{k+1}\rangle}\,|0\rangle\cdots\overset{N-1}{|d_N\rangle}\,|0\rangle\overset{N}{|v\rangle}\,|1\rangle]\,|k\rangle\,|d_k\rangle\to\\&[|d_1\rangle\,|0\rangle\,|d_2\rangle\,|0\rangle\cdots|d_{k-1}\rangle\,|0\rangle\,\underset{k}{|v\rangle}\,|1\rangle\underset{k+1}{|d_{k+1}\rangle}\,|0\rangle\cdots\underset{N}{|d_N\rangle}\,|0\rangle]\,|k\rangle\,|d_k\rangle\end{aligned}$$

(v) Uncompute index $= k$ indicator flags:

$$\begin{aligned}&[|d_1\rangle\,|0\rangle\,|d_2\rangle\,|0\rangle\cdots|d_{k-1}\rangle\,|0\rangle\,|v\rangle\,|1\rangle\,|d_{k+1}\rangle\,|0\rangle\cdots|d_N\rangle\,|0\rangle]\,|k\rangle\,|d_k\rangle\to\\&[|d_1\rangle\,|0\rangle\,|d_2\rangle\,|0\rangle\cdots|d_{k-1}\rangle\,|0\rangle\,\underset{k}{|v\rangle}\,|0\rangle\,|d_{k+1}\rangle\,|0\rangle\cdots|d_N\rangle\,|0\rangle]\,|k\rangle\,|d_k\rangle\end{aligned}$$

Note that for the above task we can use a very basic, incomplete sorting network which only finds the largest element. The total depth of the above implemented as a circuit is $\mathcal{O}(\log(N)+\log m)$, where m is the bit length of the $|d_i\rangle$ data elements stored in the qRAM. At first sight probably it is not obvious why we do not need depth $\mathcal{O}(\log(N)*\log(m))$ instead, but with some little tricks we may parallelise the swaps during the sorting process while still keeping the number of necessary workspace qubits as low as $\mathcal{O}(\log(N)*m)$. (For a direct implementation see [GLM].)

6.3.6. Pseudo code notation for the subroutines introduced

In order to be able to describe our algorithms shortly we introduce a sort of pseudo code notation for the subroutines introduced in this Subsection together with some short notation for other related operations.

First we need some notation regarding arrays. As introduced in the previous qRAM example we will call the state $|d_1,d_2,\ldots,d_N\rangle$, where each d_i represents some register of k qubits an array, denoted by $d[\,]$ and we intuitively think of the registers as physically lined up next to each other. The number of registers in the array is called the length of the array, while the number of qubits in the registers k will be called the bitlength of the array fields. To allocate some workspace qubits which are originally in the zero state into an array $a[\,]$ of length N and bitlength n we are going to use the following notation: $a(n)[N]\equiv 0$. To merge two arrays $d[\,],a[\,]$ and treat them as a logical unit we are going to use the following notation: $A[\,]=(a[\,],e=d[\,])$. After this we may access the elements as $a[\,]=A.a[\,]$, $d[\,]=A.e[\,]$ and $A[i]=(a[i],d[i])=(A.a[i],A.e[i])$.

Now we list the pseudo code commands and notations here.

$c \equiv 0$: This denotes that c is a/some unused or already uncomputed workspace qubit(s) which have value 0 in all superpositions. (Except probably on a small error term treated separately.)

$out \oplus =$ result(in): This simply denotes the output register of a calculation:

$$|in\rangle\,|out\rangle \to |in\rangle\,|out \oplus \text{result}(in)\rangle$$

e-w$(k \in [K-1])$: This denotes element-wise operations, and means that the operations in this block can be done parallely for the array indices in $k \in [K-1]$. Note that if the block contains some operation like $A.b[k] \oplus = A.a[k] == A.a[k+1]$ that means we can do all the neighbour comparison parallely. More precisely in this case we can do it in two steps first for even k-s then for odd k-s to avoid simultaneous data access, which is not allowed in quantum computing. So in general **e-w** denotes operations which may be implemented with a constant depth circuit.

distribute(c): This command is always at the beginning of an **e-w**$(k \in [K])$ block. It means that we need the value of c for the parallel computations so first distribute it, and undistribute immediately after the **e-w**$(k \in [K])$ block ends. As discussed in Subsection 6.3.2 this operation uses $\mathcal{O}(n * K)$ workspace qubits and has circuit depth $\mathcal{O}(\log(K))$, where n is the bitlength of c and K is the size of the parallel computation.

collect$(A.a[\,];$ bitwise OR $|\, A.b[\,])$: This command collects the values of $A.a[\,]$ using a binary tree of bitwise OR gates, but collects info only from indices i where the flag qubit $A.b[i]$ is set to 1, similarly to as described in Subsection 6.3.2. Note that the condition $|\, A.b[\,]$ may be omitted if we want to collect everything anyway. The number of workspace qubits used is $\mathcal{O}(n * K)$, while the circuit depth is $\mathcal{O}(\log(K))$, where K is the length of the array $A.a[\,]$, and n is the bitlength of its elements.

sort$(A[\,];\ A.a[\,], A.b[\,])$: Sort the array $A[\,]$ according to the fields $A.a[\,]$ in case of equality we also compare the $A.b[\,]$ values and sort according to that. The workspace qubits of the comparators remain in use until the block ends and the array is unsorted. For the workspace an depth complexity see Subsection 6.3.1.

qRAM$(A.a[\,];\ j;\ v)$: Swaps $A.a[j] \longleftrightarrow v$, for detailed description and complexity analysis see Subsection 6.3.5.

readArray$(A.n[\,]; j)$: Works similarly to (and can be implemented easily using) **qRAM** but it only reads the array: $|A.n[\,], j, out\rangle \to |A.n[\,], j, out \oplus A.n[j]\rangle$.

uncompute$(lines)$: Uncomputes the specified *lines*. Be careful, only those computations can be undone for which the necessary input fields: array elements and variables are unchanged. The complexity is the same as the computation of *lines*.

contolledby(binary[ ]): Inside this block every operation is controlled by the values stored in the binary[ ]. Even the comparators of sorting networks, so the sort will only act on the subset where the binary[ ] has value pre-set 1. Complexity N/A.

$|i\rangle :=$**uniform**$([K])$: Generates a uniform probability distribution on $[K]$ and sets the state of the register $|i\rangle = \sqrt{1/K}\sum_{j=1}^{K}|j\rangle$. Note that it is only applicable/works correctly if before the operation $i \equiv 0$. Workspace qubits needed $\mathcal{O}(1)$, depth if the circuit $\mathcal{O}(\log(K))$ see Subsection 6.3.3.

6.4. Implementing the Element Distinctness Algorithm

The question of the Element Distinctness Problem is the following: Given a function $f : [1, 2, 3, \ldots, N] \to [1, 2, 3, \ldots, M]$ decide whether there are two identifiers $1 \leq i < j \leq N$ such that $f(i) = f(j)$ – or in other words whether f is injective.

The Element Distinctness Algorithm is based on a quantum walk on the Johnson graph. The vertices of the Johnson graph correspond to fixed size subsets of $[1, 2, 3, \ldots, N]$. Two vertices v_1, v_2 are connected to each other whenever the corresponding subsets S_1, S_2 have symmetric difference of size two ($|S_1 \triangle S_2| = 2$). To describe sets of natural numbers flawlessly we introduce some notation $[N] := [1, 2, 3, \ldots, N]$, $\binom{[N]}{K} := \{S \subseteq [N] : |S| = K\}$, $S_1 \triangle S_2 := (S_1 \cup S_2) \setminus (S_1 \cap S_2)$.

The fixed size K of the subsets of $[N]$ can be anything between 1 and N, but in terms of query complexity the optimal value is $N^{2/3}$. Here we will be concerned by the cost of non query steps as well, and will do the implementation/analysis for a general value of $K = N^q$, where $q \in (0, 1)$ is an arbitrary but fixed real number.

6.4.1. Data structure and high level description of necessary operations

To implement the Element distinctness Algorithm efficiently one needs to store the the already queried function values along with the identifiers. Our basic data structure is straightforward, we are going to store a subset of the possible identifiers $[N]$ in a sorted array of bunches of qubits. The array itself is just a whole lot of qubits which we intuitively think of as being lined up next to each other. The array is going to have K elements each of which consist of 3 segments of qubits: $(|\text{index}\rangle\, |\text{id}\rangle\, |\text{f(id)}\rangle)$

Suppose the set S consists of identifiers $a_1 < a_2 < \ldots < a_k < \ldots < a_K$ then the corresponding data array (shortly denoted by $|A^S\rangle$) is a product state of K elements: (Just the basis states; it may be in superposition so the whole state can be entangled.) $|A^S\rangle = [(|1\rangle\, |a_1\rangle\, |f(a_1)\rangle)\ (|2\rangle\, |a_2\rangle\, |f(a_2)\rangle) \ldots (|k\rangle\, |a_k\rangle\, |f(a_k)\rangle) \ldots (|K\rangle\, |a_K\rangle\, |f(a_K)\rangle)]$.
So the storage needed for this array of qubits is $\mathcal{O}(K(\log(K) + \log(N) + \log(M)) = \mathcal{O}(K\log(K + N + M))$, where M is the size of the range of $f : [N] \to [M]$.

Later in the pseudo codes we are going to use the notation $A^S[\,]$ to emphasize that it is an array. Also we are going to use the notation $A.j[\,], A.n[\,], A.v[\,]$ for the sub arrays containing respectively the index the number and the function value.

(Note that sometimes we will include the index of the elements in description and sometimes we simply uncompute it depending on our needs. Also note that all the index qubits can be calculated simultaneously e.g. with a circuit of depth 1 from 0 workspace qubits, because they depend only on the actual location of the qubits, and thus can be hard coded into circuit.)

Because we are going to work in a bipartite setting we will store a left and a right data array describing two sets S_1, S_2. So a possible base state looks like:

$$|A^{S_1}\rangle_\ell \otimes |A^{S_2}\rangle_r$$

Following the general implementation scheme of [MNRS] we basically need to implement the following three operations: (See also Subsection 5.2)
Setup: Creates the starting uniform distribution on the left vertex set from the initial zero state:

$$Setup : |0\rangle_\ell |0\rangle_r \to \sqrt{\frac{1}{\binom{N}{K}}} \sum_{S\in\binom{[N]}{K}} |A^S\rangle_\ell |0\rangle_r \tag{28}$$

Update: It covers 4 different operators. One that generates the right transition states (U_R), and its symmetric pair (U_L) together with their inverses (U_R^{-1}, U_L^{-1}).

$$U_R = |A^{S_1}\rangle_\ell |0\rangle_r \to \sqrt{\frac{1}{K(N-K)}} \sum_{S_2\in\binom{[N]}{K}}^{|S_1\triangle S_2|=2} |A^{S_1}\rangle_\ell |A^{S_2}\rangle_r$$

$$U_L = |0\rangle_\ell |A^{S_2}\rangle_r \to \sqrt{\frac{1}{K(N-K)}} \sum_{S_1\in\binom{[N]}{K}}^{|S_1\triangle S_2|=2} |A^{S_1}\rangle_\ell |A^{S_2}\rangle_r$$

Check: Checks whether the left (right) data array contains two identifiers with the same function value, and conditional on this changes the value of an additional flag qubit $|b\rangle$ used for storing this information temporarily:

$$\begin{array}{lll} C_L: & |A^{S_1}\rangle_\ell |A^{S_2}\rangle_r |b\rangle & \to & |A^{S_1}\rangle_\ell |A^{S_2}\rangle_r |b \oplus_{=}^{S_1} 1\rangle \\ C_R: & |A^{S_1}\rangle_\ell |A^{S_2}\rangle_r |b\rangle & \to & |A^{S_1}\rangle_\ell |A^{S_2}\rangle_r |b \oplus_{=}^{S_2} 1\rangle \end{array}$$

where $b \oplus_{=}^{S} 1$ is the conditional binary addition, if A^S contains two identifiers with the same value of f, it flips the value of b, otherwise keep the value of b.

For a more detailed description of these operators and their role see e.g. [MNRS].

6.5. Implementation of Check

This is the simplest operation, we just sort the left (right) array based on the comparison of the stored function values $A.v[\,]$. After sorting we check each neighbour data entries of the array to see whether they are equal. We store these boolean result of the comparisons in some indicator array $b[\,]$, from which we collect the info using a binary tree of OR gates. Finally we uncompute everything except the one qubit $|o_A^S\rangle$ containing the the info whether there were any duplicates in the array $A.v[\,]$.

```
Algorithm: Implementation of Check
   Input: $|A^S[\,], o^S_A\rangle$, where $A^S[\,]$ contains the description of $S$ with a field
          $A^S.v[\,]$ containing the function values corresponding to the elements of
          $S$, and a qubit $o^S_A$ for the output.
   Result: $|o^S_A\rangle$ gets flipped iff there are indices $j \neq k$ such that $A^S.v[j] = A^S.v[k]$
 1 workspace variables: in brackets (bitlength), if array then [length of array]:
   $b(1)[K]$;
   // Assign a one bit array $\boldsymbol{b}[\,]$ of some workspace $(\equiv \mathbf{0})$ qubits to
      $\boldsymbol{A^S}[\,]$ and later use this extended array $\boldsymbol{A}[\,]$ for computation:
 2 array initialization: $A[\,] := (v[\,] = A^S.v[\,], b[\,] \equiv 0)$;
 3 sort($A[\,]$; $A.v$)    // $A[\,]$ gets sorted according to the function values
 4    e-w($k \in [K-1]$): if $A.v[k] == A.v[k+1]$ then $A.b[k] \oplus= 1$;
 5    $o^S_A \oplus=$ collect($A.b[\,]$; OR); // write the output using binary $\oplus$
 6    uncompute(4)
 7 unsort
```

Size and depth of the quantum circuit: At the initialization (line 2) we allocate K qubits of the workspace. This is not an actual operation, just an indication that we are going to use those qubits attached to our input array A^S. Therefore this step has space and time complexity 0 but reserves K workspace qubits.

The sort operation (line 3) has space complexity $\mathcal{O}(K\log(M))$ and time complexity $\mathcal{O}(\log(K)\log\log(M))$, and reserves $\mathcal{O}(K\log(K))$ permanent qubits for storing the permutation of the set.

The element-wise operations (line 4,6) have time complexity $\mathcal{O}(\log(M))$ since they may be done parallely for all elements. More precisely we may do these operations in two phases, first for even and then for the odd k-s to avoid simultaneous use of the same qubits. Because checking the equality of two $\log(M)$ bit numbers may be done with a circuit of depth $\mathcal{O}(\log\log(M))$ and using $\mathcal{O}(\log(M))$ extra workspace qubits this parallel operation has time complexity $\mathcal{O}(\log\log(M))$ and space complexity $\mathcal{O}(K\log(M))$.

The implementation of collect (line 5) has time complexity $\mathcal{O}(\log(K))$ and space complexity $\mathcal{O}(K)$.

Thus the circuit depth needed to implement this algorithm is $\mathcal{O}(\log(K)\log\log(M))$ and to achieve this we need to use $\mathcal{O}(K\log(M))$ extra workspace qubits. The query complexity is 0 since we did not use any oracle call.

6.6. Implementation of Update

Suppose we store the set $S_1 = \{a_1, a_2 \dots a_K\}$ described in our data array $|A^{S_1}\rangle$. We need to generate uniformly at random a new set S_2 such that $|S_1 \triangle S_2| = 2, |S_1| = |S_2|$. We proceed by generating a random index $i \in [K]$ and also a "co-index" $c \in [N-K]$. Then we define $S_2 = S_1 \setminus \{a_i\} \cup \{b_c\}$, where b_c is the c-th largest element of $N \setminus S_1$. To proceed further we need to replace the transition description $|i,c\rangle$ with a proper one $|i,j,b_c,f(b_c)\rangle$, where j is the index of the largest element of S_1 being less than b_c. This part of the process is described in the uniformComplementElements procedure. While the whole operation we want to implement is:

$$|A^{S_1}[\,]\rangle_\ell |0\rangle_r |0\rangle_w \to \sqrt{\frac{1}{K(N-K)}} \sum_{S_2 \in \binom{[N]}{K}}^{|S_1 \triangle S_2|=2} |A^{S_1}[\,]\rangle_\ell |A^{S_2}[\,]\rangle_r |0\rangle_w \tag{29}$$

Procedure: uniformComplementElements

```
Input: $|A^S[\,]\rangle$ containing the description of $S$ with fields $A^S.n[\,]$, $A^S.v[\,]$, such
       that the values in $A.n[\,]$ are stored in increasing order.
Result: A uniform distribution over the elements of the complementer set
        $[N] \setminus S$ : represented as $|A^S[\,]\rangle\, 1/\sqrt{N-K} \sum_{b_c \in [N]\setminus S} |j, b_c, v_{b_c} = f(b_c)\rangle$,
        where $j$ is the index of the largest element of $A.n[\,]$ being less than $b_c$.
1  workspace variables: in brackets (bitlength), if array then [length of array]:
   $c(\log(N)),\ b_c(\log(N)),\ j(\log(K)),\ v_{b_c}(\log(M)),\ j(\log(K))[K],\ b(1)[K]$;
   // First generate the uniform probability distribution over $[N-K]$
      so later we only need to use reversible logical operations:
2  probability distribution initialization: $|c\rangle$ :=**uniform**$([N-K])$;
   // Assign some arrays of workspace $(\equiv 0)$ qubits to $A^S[\,]$ and later
      use this extended array $A[\,]$ for computation:
3  array initialization: $A[\,] := (n[\,] = A^S.n[\,],\ v[\,] = A^S.v[\,],\ j[\,] \equiv 0,\ b[\,] \equiv 0)$;
4  **e-w**$(k \in [K])$: $A.j[k] \oplus= k$; // Generating the indices for later use
5  **e-w**$(k \in [K-1])$: **distribute**$(c)$
     // For simplicity we write $k$ instead of $A.j[k]$ here:
6    **if** $A.n[k] - k - c < 0$ & $A.n[k+1] - (k+1) - c \geq 0$ **then** $A.b[k] \oplus= 1$;
7  $j \oplus=$ **collect**$(A.j[\,]$; bitwise OR $|\, A.b[\,])$;
8  **uncompute**$(4$ - $6)$;
   // If $j=0$ then we want to replace it with $K$. Since $j=K$ can not
      hold (note the range of 5) we proceed reversibly as follows:
9  flag$\equiv 0$;// Use a workspace qubit originally in 0 state
10 flag$\oplus= (j == 0)$;
11 **if** flag $== 1$ **then** $j \oplus= K$;
12 flag$\oplus(j == K)$;// Note that again flag$\equiv 0$ since before line 10 $j \neq K$
13 $b_c \oplus= (c + j)$; // Now we know the $c$-th element of $[N] \setminus S$
14 $v_{b_c} \oplus= (f(b_c))$; // One oracle call to $f$
15 $c \oplus= (b_c - j)$; // We uncomputed $c$, now $c \equiv 0$
```

First let us discuss the correctness of the procedure described in uniformComplementElements. This we will call phase i-ii:

i) Generate two numbers $i \in [K]$ and $c \in [N-K]$ uniformly:
$|A^{S_1}[\,]\rangle_\ell |0\rangle_r (|0\rangle)_w \to |A^{S_1}[\,]\rangle_\ell |0\rangle_r \left(\sqrt{\frac{1}{K(N-K)}}\, |i\rangle\, |c\rangle\, |0\rangle\right)_w$
We may implement this step as described in 6.3.3.

ii) Calculate the proper description of the element to be added to S_1 originally described by "co-index" c. The proper description includes the actual identifier b_c to be added to S_1 and also the index j of the element after which we would insert b_c. Let $S_1 = \{a_1 < a_2 < \ldots < a_K\}$ then formally j is the index for which $a_j < b_c$ and $b_c < a_{j+1}$ or if $a_K < b_c$ then $j = K$. Formally we want to replace the old description $(|i\rangle\,|c\rangle\,|0\rangle)_w$ by $(|i\rangle\,|j\rangle\,|b_c\rangle\,|f(b_c)\rangle\,|0\rangle)_w$.

To extract the value j from $|A^{S_1}[\,]\rangle$ we need some calculations. First observe that $(a_k - k)$ is the number of elements preceding a_k not in S_1 i.e. $(a_k - k) = |([N]\setminus S_1)\cap[a_k]|$. Thus $a_k < b_c$ iff $|([N]\setminus S_1)\cap[a_k]| < c$ which we may rewrite as $a_k - k - c < 0$. So j is the largest index for which this holds. For now let us interpret $(a \mod N)$ to be the representant in $[0, 1, \ldots, N-1]$ rather than the whole congruence class. Using this notation $j = \arg\max(a_k - k - c \mod N)$ since $0 \leq a_k - k < N$.

The above observations enable us to extract $|j\rangle$ from $|A^{S_1}[\,]\rangle_\ell$. We first modify identifiers of each element in $|A^{S_1}[\,]\rangle_\ell$: $|k\rangle\,|a_k\rangle\,|f(a_k)\rangle \to |k\rangle\,|a_k - k - c \mod N\rangle\,|f(a_k)\rangle$ then we sort the array using the modified identifiers resulting in a modified data array denoted by $|mA_c^{S_1}[\,]\rangle_\ell$. According to the previous observations the last element of $|mA_c^{S_1}[\,]\rangle_\ell$ is $|j\rangle\,|a_j - j - c \mod N\rangle\,|f(a_j)\rangle$ thus we get access to $|j\rangle$.

Observing that $j = |S_1 \cap [b_c]|$ we can conclude that $b_c = c + j$. As we have access to $|j\rangle$ now it is straightforward to implement $\underbrace{(\ldots|j\rangle\ldots)}_{|mA_c^{S_1}[\,]\rangle_\ell|0\rangle_r}(|i\rangle\,|c\rangle\,|0\rangle)_w \to$ $(\ldots|j\rangle\ldots)(|i\rangle\,|j\rangle\,|b_c\rangle\,|f(b_c)\rangle\,|c\rangle\,|0\rangle)_w$ with one query to f.

Finally we can put the whole process together:

$$|A^{S_1}[\,]\rangle_\ell\,|0\rangle_r\,(|i\rangle\,|c\rangle\,|0\rangle)_w \to |mA_c^{S_1}[\,]\rangle_\ell\,|0\rangle_r\,(|i\rangle\,|c\rangle\,|0\rangle)_w \to |mA_c^{S_1}[\,]\rangle_\ell\,|0\rangle_r\,(|i\rangle\,|j\rangle\,|b_c\rangle\,|f(b_c)\rangle\,|c\rangle\,|0\rangle)_w$$
$$\to |A^{S_1}[\,]\rangle_\ell\,|0\rangle_r\,(|i\rangle\,|j\rangle\,|b_c\rangle\,|f(b_c)\rangle\,|c\rangle\,|0\rangle)_w \to |A^{S_1}[\,]\rangle_\ell\,|0\rangle_r\,(|i\rangle\,|j\rangle\,|b_c\rangle\,|f(b_c)\rangle\,\underbrace{|c\oplus(b_c - j)\rangle\,|0\rangle}_{|0\rangle})_w$$

The last step shows how to uncompute c using the observation that $c = b_c - j$.

```
Algorithm: Implementation of Update
    Input: $|A^{S_1}[\,]\rangle$ containing the description of $S_1$ with fields $A^{S_1}.n[\,]$, $A^{S_1}.v[\,]$,
           such that the values in $A.n[\,]$ are stored in increasing order.
    Result: A uniform distribution over the neighbour sets as described in (29).
 1  workspace variables: in brackets (bitlength), if array then [length of array]:
    $i(\log(K)),\ j(\log(K)),\ b_c(\log(N)),\ v_{b_c}(\log(M)),\ \mathrm{flag}(1),\ j(\log(K))[K],\ b(1)[K]$;
    // First generate the uniform distribution for the transitions so
       later we only need to use reversible logical operations.
 2  probability distribution initialization: $|i\rangle$ :=uniform([$K$]);
 3  $|j, b_c, f(b_c)\rangle$ :=uniformComplementElements($|A^{S_1}[\,]\rangle$); // Alg. above
    // Assign some arrays of workspace $(\equiv 0)$ qubits to $A^{S_1}[\,]$ and later
       use this extended array $A[\,]$ for computation.
 4  array initialization 1: $A[\,] := (n[\,] = A^{S_1}.n[\,],\ v[\,] = A^{S_1}.v[\,],\ j[\,] \equiv 0,\ b[\,] \equiv 0)$;
    // Allocate some workspace $(\equiv 0)$ qubits for a second array $B[\,]$.
 5  array initialization 2: $B[\,] := \{n[\,] \equiv 0,\ v[\,] \equiv 0,\ j[\,] \equiv 0,\ b[\,] \equiv 0\}$;
    // Note that this array $B[\,]$ is going to represent $A^{S_2}[\,]$
 6  e-w($k \in [K]$): $A.j[k] \oplus= k$, $B.j[k] \oplus= k$; // Generating the indices
 7  e-w($k \in [K]$): distribute($i, j$)
 8      switch $k$ do
 9          case $(k == i)$ ; // Nothing hapens, this is the element we omit
10          case $(i < k \leq j)$ : $B.n[k-1] \oplus= A.n[k]$, $B.v[k-1] \oplus= A.v[k]$;
11          case $(j < k \leq i)$ : $B.n[k+1] \oplus= A.n[k]$, $B.v[k+1] \oplus= A.v[k]$;
            // $k >$max$(i, j)$ or $(i \neq) k \leq$min$(i, j)$, indices agree in $S_1, S_2$:
12          otherwise : $B.n[k] \oplus= A.n[k]$, $B.v[k] \oplus= A.v[k]$;
13      endsw
14  end e-w
15  flag$\equiv 0$, flag$\oplus= (j < i)$; // Flip the value of a workspace qubit if $j < i$
16  if flag == 1 then $j$+= 1;  // This is not binary, but normal addition!
17  if readArray($A.n[\,]; j$) $\geq b_c$ then flag $\oplus= 1$;          // Now flag $\equiv 0$
18  qRAM($B.n[\,]; j; b_c$); // $b_c$ was replaced by $A.n[j] \equiv 0$
19  qRAM($B.v[\,]; j; f(b_c)$); // $f(b_c)$ was replaced by $A.v[j] \equiv 0$
20  e-w($k \in [K]$):   // If $k - 1 = 0$ or $k + 1 = K + 1$ consider term true:
21      if $A.n[k] \neq B.n[k-1]$ & $A.n[k] \neq B.n[k]$ & $A.n[k] \neq B.n[k+1]$ & then
22          $A.b[k] \oplus= 1$; // $A.n[k]$ is not present in $B.n[\,]$
23      if $B.n[k] \neq A.n[k-1]$ & $B.n[k] \neq A.n[k]$ & $B.n[k] \neq A.n[k+1]$ & then
24          $B.b[k] \oplus= 1$; // $B.n[k]$ is not present in $A.n[\,]$
25  $i \oplus=$ collect($A.j[\,]$; bitwise OR | $A.b[\,]$); // Now $i \equiv 0$
26  $j \oplus=$ collect($B.j[\,]$; bitwise OR | $B.b[\,]$); // Now $j \equiv 0$
27  uncompute(20 - 24); // From now on $A.b[\,] \equiv 0, B.b[\,] \equiv 0$
28  uncompute(6); // From now on $A.j[\,] \equiv 0, B.j[\,] \equiv 0$
```

iii) The detailed description of Implementation of Update. Note that some step are discussed in more detail than described in the pseudo code, e.g. step 0.)-1.) is not divided to two parts in the pseudo code.
Here comes the rest: generating the set S_2 from S_1: $|A^{S_1}[\,]\rangle_\ell\,|0\rangle_r\,(|i\rangle\,|j\rangle\,|b_c\rangle\,|f(b_c)\rangle\,|0\rangle)_w \to |A^{S_1}[\,]\rangle_\ell\,|A^{S_2}[\,]\rangle_r\,(|0\rangle)_w$ The difference of S_1 and S_2 is fully described by the information $|i\rangle\,|j\rangle\,|b_c\rangle\,|f(b_c)\rangle$ stored on the workspace, so we can reversibly transform this description to an actual array representation of S_2. Technically we proceed by first generating $|A^{S_2}[\,]\rangle$ from $|A^{S_1}[\,]\rangle$ using the description $|i\rangle\,|j\rangle\,|b_c\rangle\,|f(b_c)\rangle$ then we uncompute these workspace qubits by calculating the symmetric difference of S_1 and S_2. We divide this phase into 4 steps:

$$\begin{array}{lclcl}
|11\rangle\,|1\rangle\,|a_1\rangle\,|f(a_1)\rangle & & |11\rangle\,|1\rangle\,|a_1\rangle\,|f(a_1)\rangle & & |00\rangle\,|1\rangle\,|a_1\rangle\,|f(a_1)\rangle \\
\quad\;\; |0\rangle\,|0\rangle\,|0\rangle & & \quad\;\; |0\rangle\,|a_1\rangle\,|f(a_1)\rangle & & \quad\;\; |1\rangle\,|a_1\rangle\,|f(a_1)\rangle \\
|11\rangle\,|2\rangle\,|a_2\rangle\,|f(a_2)\rangle & & |11\rangle\,|2\rangle\,|a_2\rangle\,|f(a_2)\rangle & & |00\rangle\,|2\rangle\,|a_2\rangle\,|f(a_2)\rangle \\
\quad\;\; |0\rangle\,|0\rangle\,|0\rangle & & \quad\;\; |0\rangle\,|a_2\rangle\,|f(a_2)\rangle & & \quad\;\; |2\rangle\,|a_2\rangle\,|f(a_2)\rangle \\
\vdots & & \vdots & & \vdots \\
|11\rangle\,|i-1\rangle\,|a_{i-1}\rangle\,|f(a_{i-1})\rangle & & |11\rangle\,|i-1\rangle\,|a_{i-1}\rangle\,|f(a_{i-1})\rangle & & |00\rangle\,|i-1\rangle\,|a_{i-1}\rangle\,|f(a_{i-1})\rangle \\
\quad\;\; |0\rangle\,|0\rangle\,|0\rangle & & \quad\;\; |0\rangle\,|a_{i-1}\rangle\,|f(a_{i-1})\rangle & & \quad\;\; |i-1\rangle\,|a_{i-1}\rangle\,|f(a_{i-1})\rangle \\
|00\rangle\,|i\rangle\,|a_i\rangle\,|f(a_i)\rangle & & |00\rangle\,|i\rangle\,|a_i\rangle\,|f(a_i)\rangle & & |00\rangle\,|i\rangle\,|a_i\rangle\,|f(a_i)\rangle \\
\quad\;\; |0\rangle\,|0\rangle\,|0\rangle & & \quad\;\; |0\rangle\,|a_{i+1}\rangle\,|f(a_{i+1})\rangle & & \quad\;\; |i\rangle\,|a_{i+1}\rangle\,|f(a_{i+1})\rangle \\
|01\rangle\,|i+1\rangle\,|a_{i+1}\rangle\,|f(a_{i+1})\rangle & & |01\rangle\,|i+1\rangle\,|a_{i+1}\rangle\,|f(a_{i+1})\rangle & & |00\rangle\,|i+1\rangle\,|a_{i+1}\rangle\,|f(a_{i+1})\rangle \\
\quad\;\; |0\rangle\,|0\rangle\,|0\rangle & & \quad\;\; |0\rangle\,|a_{i+2}\rangle\,|f(a_{i+2})\rangle & & \quad\;\; |i+1\rangle\,|a_{i+2}\rangle\,|f(a_{i+2})\rangle \\
\vdots & \overset{\mathbf{1}}{\to} & \vdots & \overset{\mathbf{2}}{\to} & \vdots \\
|01\rangle\,|j-1\rangle\,|a_{j-1}\rangle\,|f(a_{j-1})\rangle & & |01\rangle\,|j-1\rangle\,|a_{j-1}\rangle\,|f(a_{j-1})\rangle & & |00\rangle\,|j-1\rangle\,|a_{j-1}\rangle\,|f(a_{j-1})\rangle \\
\quad\;\; |0\rangle\,|0\rangle\,|0\rangle & & \quad\;\; |0\rangle\,|a_j\rangle\,|f(a_j)\rangle & & \quad\;\; |j-1\rangle\,|a_j\rangle\,|f(a_j)\rangle \\
|01\rangle\,|j\rangle\,|a_j\rangle\,|f(a_j)\rangle & & |01\rangle\,|j\rangle\,|a_j\rangle\,|f(a_j)\rangle & & |00\rangle\,|j\rangle\,|a_j\rangle\,|f(a_j)\rangle \\
\quad\;\; |0\rangle\,|0\rangle\,|0\rangle & & \quad\;\; |0\rangle\,|0\rangle\,|0\rangle & & \quad\;\; |j\rangle\,|b_c\rangle\,|f(b_c)\rangle \\
|11\rangle\,|j+1\rangle\,|a_{j+1}\rangle\,|f(a_{j+1})\rangle & & |11\rangle\,|j+1\rangle\,|a_{j+1}\rangle\,|f(a_{j+1})\rangle & & |00\rangle\,|j+1\rangle\,|a_{j+1}\rangle\,|f(a_{j+1})\rangle \\
\quad\;\; |0\rangle\,|0\rangle\,|0\rangle & & \quad\;\; |0\rangle\,|a_{j+1}\rangle\,|f(a_{j+1})\rangle & & \quad\;\; |j+1\rangle\,|a_{j+1}\rangle\,|f(a_{j+1})\rangle \\
\vdots & & \vdots & & \vdots \\
|11\rangle\,|K\rangle\,|a_K\rangle\,|f(a_K)\rangle & & |11\rangle\,|K\rangle\,|a_K\rangle\,|f(a_K)\rangle & & |00\rangle\,|K\rangle\,|a_K\rangle\,|f(a_K)\rangle \\
\quad\;\; |0\rangle\,|0\rangle\,|0\rangle & & \quad\;\; |0\rangle\,|a_K\rangle\,|f(a_K)\rangle & & \quad\;\; |K\rangle\,|a_K\rangle\,|f(a_K)\rangle \\
 & & & & \\
(|i\rangle\,|j\rangle\,|b_c\rangle\,|f(b_c)\rangle\,|0\rangle)_w & & (|i\rangle\,|j\rangle\,|b_c\rangle\,|f(b_c)\rangle\,|0\rangle)_w & & (|i\rangle\,|j'\rangle\,|0\rangle\,|0\rangle\,|0\rangle)_w
\end{array}$$

Figure 4: Illustration of the process described in phase iii). The entries of the left and right arrays are interleaved together. The three columns represent the state after the 0th, 1st and 2nd steps accordingly. For better readability here we postponed the uncomputation of the flags until the 2nd step.

0.) At the beginning for each element of the data-array $A^{S_1}[\,]$ we assign two flag qubits. For the k-th element we set the value according to the four distinct possibilities:

(00) $k = i$ – the element is omitted from S_2

(01) $i < k \leq j$ – the index of the element is going to be smaller by 1 in S_2 then in S_1

(10) $j < k < i$ – the index of the element is going to be greater by 1 in S_2 then in S_1

(11) $k > \max(i, j)$ or $(i \neq)k \leq \min(i, j)$ – the indices are going to be the same in S_1 and S_2

Note that both (01) and (10) values can not be present in the same array as they correspond to distinct cases $i < j$ and $i > j$.

1.) Based on these flag qubits we do three controlled copy (bitwise addition) operators, to write the k-th element of the left array to the $(k-1)$-st, k-th or $(k+1)$-st place of the right array conditional on the flag values (01), (11) or (10). Here we copy only the identifiers and the corresponding function values $|a_k\rangle\,|f(a_k)\rangle$, but not the possibly wrong indices. After the controlled copy we uncompute the flags.

2.) Now we calculate the position j' where to insert the new element. If $j < i$ then $j' = j + 1$ otherwise $j' = j$. Let us denote with $c_{j<i} - NOT$ the conditional flip operation conditioned on whether $j < i$, then we may implement this calculation: $|i\rangle\,|j\rangle\,|0\rangle \overset{c_{j<i}-NOT}{\rightarrow} |i\rangle\,|j\rangle\,|f\rangle \rightarrow |i\rangle\,|j' = j + f\rangle\,|f\rangle \overset{c_{b_c<a_{j'}}-NOT}{\rightarrow} |i\rangle\,|j\rangle\,|0\rangle$. Note that in the last step we can not use $c_{j<i} - NOT$ because we have access to j' only, which is not enough. By looking only at the value j' we can not distinguish the two cases $j = i$ and $j = i - 1$. However, we may recover whether $j = j'$ by checking $b_c < a_{j'}$, because j was defined as the largest index for which $a_j < b_c$ hold. To check whether $b_c < a_{j'}$ we may use our data array $|A^{S_1}[\,]\rangle$ as described in our qRAM implementation 6.3.5.

Now we generate all the indices of the right data array. This may be done with a circuit of depth one, since the index of an element is directly related to its actual (physical) location. Thus we may simply put a (NOT) gate where the indices have binary value 1, and not do anything where this binary value is 0.

We are ready to swap $|b_c\rangle\,|f(b_c)\rangle)$ to the j'-th position – again we may proceed as described in our qRAM implementation 6.3.5.

3.) Finally based on the symmetric difference of the right and left arrays we can find the indices i, j' where the two sets differ. Again we may calculate an indicator flag showing whether the k-th element of one data array does not equal neither the $(k-1)$-st or the k-th or $(k+1)$-st element of the other data array. This will only mark the i-th element of S_1 and the j'-th element of S_2. Based on these flag qubits we can sort the data arrays so that i and j' are going to be the last elements of the two data arrays.

Then we can erase the first two remaining registers for example by bitwise addition (CNOT). I.e. we can implement the final step $|A^{S_1}[\,]\rangle_\ell\,|A^{S_2}[\,]\rangle_r\,(|i\rangle\,|j'\rangle\,|0\rangle)_w \rightarrow |A^{S_1}[\,]\rangle_\ell\,|A^{S_2}[\,]\rangle_r\,(|0\rangle)_w$

The phases i-ii-iii together implement the Update transformation using $\mathcal{O}(N\log(N))$ extra workspace initially in the zero state and with $\mathcal{O}(log(N))$ circuit depth using 1 query:

$$U_R^w : |A^{S_1}[\,]\rangle_\ell |0\rangle_r (|0\rangle)_w \to \sqrt{\frac{1}{K(N-K)}} \sum_{S_2 \in \binom{[N]}{K}}^{|S_1 \triangle S_2|=2} |A^{S_1}[\,]\rangle_\ell |A^{S_2}[\,]\rangle_r (|0\rangle)_w$$

6.7. Implementation of Setup

With the Setup (28) operation we aim to generate a uniform probability distribution over all the $\binom{[N]}{K}$ subsets. Also we would like to represent or store the subsets as data arrays described in Section 6.4.1.

Classically this is not a hard task. Following a usual approach we could approximate the uniform distribution on the cumbersome $\binom{[N]}{K}$ set of subsets by reducing a larger but simpler distribution over the integer sequences. Instead of generating K distinct integers we could first generate a uniform sequence of L numbers which may contain some repetitions. Then we could seek for the first K distinct numbers and move (sort) them to the beginning of the sequence. We may not find K distinct ones but if do, then we a get a uniform sequence of K distinct numbers, which we could finally sort in order to get our increasing sequence representation of sets.

Classically the only question is the length of the sequence needed to get K distinct numbers with high probability. This is the well known coupon collector's problem. Let us denote the position of the first occurrence of the K-th distinct number in the sequence with W_K^N, and let μ_N, σ_N^2 be the mean and variance of this random variable. If both $K(N)$ and $N - K(N)$ tends to infinity as N tends to infinity then a sort of central limit theorem holds [BB]: $(W_K^N - \mu_N)/\sigma_N$ tends to $\mathcal{N}(0,1)$ in distribution, where $\mu_N = \sum_{i=1}^{K(N)} \frac{N}{N-i+1}$ and $\sigma_N^2 = \sum_{i=1}^{K} \frac{(i-1)N}{(N-i+1)^2}$. If $K < N/2$ then $\mu_N < 2K$ and $\sigma_N^2 < 2K^2/N < K$ showing that a sequence of length $c \cdot K$ $(c > 2)$ fails to contain K distinct elements with exponentially small probability in $\sqrt{K}$. Note that this may be seen directly using the Chernoff bound.

Even in the worst case when $K = N$ a sequence of length $(\log(N) + c)N$ fails to contain all numbers from $[N]$ with only exponentially small probability in c for large enough N [ER]. But from the point of this algorithm the case when $K = \Omega(N)$ is irrelevant, so from now on we assume that $K < N/2$ thus a sequence of length $3K$ contains K disjoint numbers with high probability.

However this technique does not seem to work in the quantum setup. Since we carry out reversible computational steps the unused part of the array containing the random sequence is going to be still present on the workspace yielding unwanted correlations. These correlations introduce entanglement that may prevent some interference during the quantum algorithm as explained Section 6.1. If we do nothing else just move the first instances of the first K distinct numbers to the beginning of the array then some correlations are obviously present between the two parts of the array. For example, the new $(K+1)$-st element will be more probably a number contained in the first K element of the array then one not contained there.

```
Algorithm: Implementation of Setup
   Input: A bunch of workspace qubits originally in zero state
   Result: A state very colse to the uniform distribution over $\binom{[N]}{K}$.
 1 workspace variables: in brackets (bitlength), if array then [length of array]:
   $k(\log(3K)), n(\log(N))[3K], v(\log(M))[K], j(\log(3K))[3K], b_1(1)[3K], b_2(1)[3K]$;
   // Allocate some workspace qubits for an array having length $3K$
 2 : array initialization 1: $A[\,] := (n[\,] \equiv 0,\ v[\,] \equiv 0,\ j[\,] \equiv 0,\ b_1[\,] \equiv 0,\ b_2[\,] \equiv 0)$;
   // First generate a uniform distribution over the sequences $[N]^{3K}$:
 3 e-w($i \in [3K]$): $|A.n[i]\rangle$ :=uniform($[N]$);
   /* The rest is plain reversible logic, thus we ignore $|.\rangle$ notation */
   /* Lines 4-13: using the binary $A.b_1[\,]$ we mark all the elements
      before the first occurrence of the $K$-th disjoint number.        */
 4 e-w($i \in [K]$): $A.j[i] \oplus= i$; // Generating the indices for later use
 5 sort($A[\,]; A.n[\,], A.j[\,]$)          // Sort with respect to (number, index)
 6     e-w($i \in [3K]$): if $A.n[i-1] == A.n[i]$ then $A.b_2[i] \oplus= 1$;   // Duplicates
 7     sort($A[\,]; A.b_2[\,], A.j[\,]$)          // Duplicates to the end of the list
 8         $k \oplus= A.j[K]$; //Index of the first occurrence of the $K$-th value
           e-w($i \in [3K]$): distribute($k$): if $A.j[i] \le k$ then $A.b_1[i] \oplus= 1$;
 9         $k \oplus= A.j[K]$;// $k$ uncomputed
10     unsort
11     uncompute(6);
12 unsort
13 uncompute(4);// $A.j[] \equiv 0$ again.
   // From now on only do computation on elements of $A[\,]$ before $k$:
14 controlledby($A.b_1[\,]$)
15     sort($A[\,]; A.n[\,]$)                  // Sort with respect to the numbers
16         e-w($i \in [3K]$) :
17             if $A.n[i-1] == A.n[i]$ then $A.b_2[i] \oplus= 1$;   // Duplicates marked
18             if $A.b_2[i] == 0$ then        // We work only with non duplicates
19                 foreach $j$ in $[c]$ do              // For $3K \le N^{2/3}$ $c := 5$
20                     if $A.n[i+j] == A.n[i]$ then $A.j[i+j] \oplus= i$;
21                     if $A.j[i+j] == i$ then $A.n[i+j] \oplus= A.n[i]$;
22                 end
23             end
24         end e-w
25         sort($A[\,]; A.b_2[\,], A.n[\,]$)       // Sort with respect to duplicates
26             e-w($i \in [K]$): $A.v[i] \oplus= f(A.n[i])$; // Oracle calls
```

We show that using some tricks we may eliminate these unwanted correlations. The main idea is to "divide and conquer", i.e. we are going to split the set of all possible sequences into small groups within each we transform the sequences such that we can eliminate the correlations.

Since we are going to work a lot with subsequences we introduce some handful notations. Let $n = (n_1, n_2, \dots, n_{3K}) \in [N]^{3K}$ be a sequence of $3K$ numbers from $[N]$. We denote the subsequence $(n_1, n_2, \dots, n_j)$ with $n(j)$. The set consisting of the elements of a (sub)sequnce $n(j)$ will be denoted by $\{n(j)\}$ and for the size of this set, i.e. the number of distinct integers in $n(j)$ we are going to use the notation $|\{n(j)\}|$.

Now we describe the correctness of the Implementation of Setup, but we not follow the it word-by-word. In particular at the end of the algorithm we proceed slightly differently to make easier the analysis of the process. Then the computationally preferred pseudo code algorithm may be analysed along similar lines. See Remark 24

In the quantum setting we also start with first uniformly generating all sequence of length $3K$ as described in Subsection 6.3.3. The difference is that now we work with superpositions representing the uniform probability distribution. Thus we generate the state:

$$\frac{1}{\sqrt{N^{3K}}} \sum_{[N]^{3K}} |n_1, n_2, \dots, n_{3K}\rangle$$

The second step is that we mark all elements (even possible duplicates) of the sequence $(n_1, n_2, \dots, n_{3K})$ up to the first occurrence of the K-th different element (having index say k) with a flag qubit set to $|b^1 = 1\rangle$. Then formally k is the unique index for which $|\{n(j-1)\}| = K - 1$ and $n_j \notin \{n(j-1)\}$. It is possible that such j does not exists, but only in the case when $\{n_1, n_2, \dots, n_{3K}\} < K$ which has low probability as we discussed in the classical setup. Now we group our sequences according to the valuee of k they have. The good thing is, that knowing the value k the subsequence before and after the k-th element remain independent, i.e. we may write our state as

$$\frac{1}{\sqrt{N^{3K}}} \sum_{k=K}^{3K} \left(\sum_{n_k \in N} \sum_{n(k-1) \in ([N] \setminus n_k)^{k-1}}^{|\{n(k-1)\}|=K-1} \underbrace{|n_1, \dots, n_{k-1}\rangle |n_k\rangle}_{|b^1=1\rangle} \right) \left(\sum_{(n_{k+1}, \dots, n_{3K})}^{[N]^{3K-k}} \underbrace{|n_{k+1}, \dots, n_{3K}\rangle}_{|b^1=0\rangle} \right)$$
$$+ \frac{1}{\sqrt{N^{3K}}} \left(\sum_{|(n_1, \dots, n_{3K})|<K}^{[N]^{k-1}} |n_1, \dots, n_{3K}\rangle \right)$$

We do not care what happens with wrong sequences appearing in the last sum since they represent only a small portion of the whole state. We would like to decompose the state consisting the good sequences into uncorrelated (or unentangled) parts. Since the elements marked with flag $|0\rangle$ are already in a product state within the state of good sequences we simply leave them unchanged i.e. from a computational point of view all the operations from now on will be in fact controlled unitary operations controlled by $|b_j^1\rangle$. What remains is to transform the first part of the above state into a nicer form. We may split the above sum into several other summations:

$$|\psi_k\rangle := \sum_{n_k \in N} \sum_{n(k-1) \in ([N] \setminus n_k)^{k-1}}^{|\{n(k-1)\}|=K-1} |n_1, \dots, n_{k-1}\rangle |n_k\rangle = \sum_{n_k \in N} \sum_{S \subset \binom{[N] \setminus n_k}{K-1}} \sum_{n(k-1) \in S^{k-1}}^{|\{n(k-1)\}|=K-1} |n_1, \dots, n_{k-1}\rangle |n_k\rangle$$

Now we mark the elements of the array state $|\psi_k\rangle$ when they are duplicates, i.e. we assign a new second flag qubit $|b^2\rangle$ to each element of the whole array. Formally speaking we set flag qubit $|b^2_j = 1\rangle$ of the j-th element iff $n_j \in n(j-1)$ (and the first flag qubit is set to $|1\rangle$). We also assign a register i_j (originally in zero state) for each element of the array which will be used for describing duplicates. If the j-th element is a duplicate then $i_j := \min\{\ell | s_l = s_j\}$. We store this value attached to the elements of the array similarly to our data-array structure 6.4.1. So one element will consist of four parts:
$|n_j\rangle\, |b^1_j \text{ indicating } j \leq k\rangle\, |b^2_j \text{ indicating duplicates}\rangle\, |i_j \text{ index of first occurence if duplicate}\rangle$
Now we also erase the duplicates by subtracting the value of the i-th element of the sequence.

$$\begin{array}{rlclclcl}
1: & |3\rangle|1\rangle|0\rangle|0\rangle & & |3\rangle|1\rangle|0\rangle|0\rangle & & |3\rangle|1\rangle|0\rangle|0\rangle & & |3\rangle|1\rangle|0\rangle|0\rangle \\
2: & |3\rangle|1\rangle|0\rangle|0\rangle & & |3\rangle|1\rangle|1\rangle|1\rangle & & |0\rangle|1\rangle|1\rangle|1\rangle & & |5\rangle|1\rangle|0\rangle|0\rangle \\
3: & |5\rangle|1\rangle|0\rangle|0\rangle & & |5\rangle|1\rangle|0\rangle|0\rangle & & |5\rangle|1\rangle|0\rangle|0\rangle & & |2\rangle|1\rangle|0\rangle|0\rangle \\
4: & |3\rangle|1\rangle|0\rangle|0\rangle & & |3\rangle|1\rangle|1\rangle|1\rangle & & |0\rangle|1\rangle|1\rangle|1\rangle & & |0\rangle|1\rangle|1\rangle|1\rangle \\
5: & |5\rangle|1\rangle|0\rangle|0\rangle & \rightarrow & |5\rangle|1\rangle|1\rangle|3\rangle & \rightarrow & |0\rangle|1\rangle|1\rangle|3\rangle & \rightarrow & |0\rangle|1\rangle|1\rangle|1\rangle \\
6: & |3\rangle|1\rangle|0\rangle|0\rangle & & |3\rangle|1\rangle|1\rangle|1\rangle & & |0\rangle|1\rangle|1\rangle|1\rangle & & |0\rangle|1\rangle|1\rangle|3\rangle \\
7: & |2\rangle|1\rangle|0\rangle|0\rangle & & |2\rangle|1\rangle|0\rangle|0\rangle & & |2\rangle|1\rangle|0\rangle|0\rangle & & |0\rangle|1\rangle|1\rangle|1\rangle \\
8: & |3\rangle|0\rangle|0\rangle|0\rangle & & |3\rangle|0\rangle|0\rangle|0\rangle & & |3\rangle|0\rangle|0\rangle|0\rangle & & |3\rangle|0\rangle|0\rangle|0\rangle \\
9: & |6\rangle|0\rangle|0\rangle|0\rangle & & |6\rangle|0\rangle|0\rangle|0\rangle & & |6\rangle\underbrace{|0\rangle|0\rangle|0\rangle}_{\text{pattern}} & & |6\rangle|0\rangle|0\rangle|0\rangle \\
 & & & & & & & \\
w: & |0\rangle & & |0\rangle & & |0\rangle & & |\text{w(p)}\rangle
\end{array}$$

Figure 5: Illustration of the data array $|n_j\rangle\, |b^1_j\rangle\, |b^2_j\rangle\, |i_j\rangle$ and the elimination of duplicates. The parameters are $K = 3$, $N = 6$ and $k = 7$. The final step shows the result of the subsequence $\rightarrow$ unsorted set + pattern transformation.

We are going to call the arrangement of the location of duplicates a pattern which is now uniquely determined by the set of registers $\{|i_j\rangle : 1 \leq j \leq k\}$. Now we may sort the array according the $|b^2\rangle$ flags to move the elements of the set $S = \{n(k)\}$ to the first K positions of the array, but not yet changing its permutation σ_S described by the order of their first appearance in $n(k)$. The reversible sorting changes the value of some workspace qubits represented in some workspace pattern $|w(p)\rangle$. Since we sort according to the $|b^2\rangle$ labels which are fully described by the previously defined duplicate pattern we may treat $|w(p)\rangle$ as a function of this duplicate pattern as indicated by the notation. Each subsequence $n(k)$ of length k with $|\{n(k)\}| = K$ is uniquely determined by the pattern p the set S and its permutation σ_S. So we may write the by now transformed $|\psi_k\rangle$ state in the following form:

$$|\psi'_k\rangle = \sum_{S\subset\binom{[N]}{K}} \sum_{\sigma_S\in[N!]} \sum_{p}^{\text{patterns}} \underbrace{|s_{\sigma_1},\ldots,s_{\sigma_K}\rangle}_{|b^2=0\rangle} \overbrace{\underbrace{|0,\ldots,0\rangle}_{|b^2=1\rangle}}^{k-K} |\text{p}\rangle\, |w(\text{p})\rangle$$

The key observation here is that each valid pattern, set and permutation describe a valid $n(k)$ subsequence so the above three sums are independent of each other. We may finally sort the first K elements of the array according to the values, again introducing some workspace qubit pattern $|w(\sigma_S)\rangle$ which depends on σ_S but is otherwise independent of S. So using the independence of p, S and σ we may write the new sorted state as:

$$|\psi''_k\rangle = \left(\sum_{S\subset\binom{[N]}{K}} \underbrace{|s_1,\ldots,s_K\rangle}_{|b^2=0\rangle}\right) \underbrace{\overset{k-K}{|0,\ldots,0\rangle}}_{|b^2=1\rangle} \left(\sum_{\sigma\in[N!]} |w(\sigma)\rangle\right) \left(\sum_{p}^{\text{patterns}} |\mathrm{p}\rangle\, |w(\mathrm{p})\rangle\right)$$

Observing that S and σ are independent of k we may write the full state coming from good sequences (for which $|\{n\}| = K$) as:

$$\frac{1}{\sqrt{N^{3K}}}\left(\sum_{S\subset\binom{[N]}{K}} \underbrace{|s_1,\ldots,s_K\rangle}_{|b^1=1\rangle|b^2=0\rangle}\right) \otimes \left(\sum_{\sigma\in[N!]} |w(\sigma)\rangle\right)$$

$$\otimes \left(\sum_{k=K}^{3K}\left(\sum_{p}^{\text{patterns(k)}} |\mathrm{p}\rangle\, |w(\mathrm{p})\rangle\right) \underbrace{\overset{k-K}{|0,\ldots,0\rangle}}_{|b^1=1\rangle|b^2\rangle=1} \sum_{(n_{k+1},\ldots,n_{3K})}^{[N]^{3K-k}} \underbrace{|n_{k+1},\ldots,n_{3K}\rangle}_{|b^1=0\rangle|b^2=0\rangle}\right)$$

Remark 24

At the end of the Implementation of Setup we use an approximation. We only check for some fixed amount of possible duplicates. We show that this is going to be good with high probability. Let us assume, that for some $0 < l < 1 : 3K \leq N^l \leq N/2$. Then for a fixed $k \in [N]$ $P(k \text{ is repeated more than } c \text{ times}) = \sum_{i=1}^{3K-c}\binom{3K}{c+i}(\frac{1}{N})^{c+i}(1-\frac{1}{N})^{3K-(c+i)} \leq \sum_{i=1}^{3K-c}(3K)^{c+i}(\frac{1}{N})^{c+i} \leq (\frac{3K}{N})^{c+1}\sum_{i=0}^{\infty}(\frac{3K}{N})^i \leq 2(\frac{3K}{N})^{c+1} \leq 2N^{(l-1)(c+1)}$. So by the union bound we also have $P(\exists k \in [N] : k \text{ is repeated more than } c \text{ times}) \leq N2N^{(l-1)(c+1)}$. It is enough for us if we bound this by say $\frac{2}{N}$. Now $N2N^{(l-1)(c+1)} \leq \frac{2}{N} \Leftrightarrow$
$(l-1)(c+1) \leq -2 \Leftrightarrow c \geq \frac{2}{1-l} - 1$. Now by substituting $l = \frac{2}{3}$, we get $c \geq 5$. So the probability some $k \in [N]$ is repeated more than 5 times is smaller $\frac{2}{N}$ if $3K \leq N^{2/3}$. ■

Note that we may be able to implement the corresponding step exactly and efficiently using a parallel algorithm similar to the well known Parallel Rank Computing.

Finally note that we may eliminate the bad part of the sate coming from bad sequences by the exact version of Amplitude Amplification 3, or calculate and indicator based on "goodness" and measure it repeating the process until we get the result 1.

Finally note that a perfect Setup operation would clean up the workspace, but here we do not need to do that, since we only apply this operation once at the beginning. Also the data-array $\sum_{S\subset\binom{[N]}{K}}|s_1,\ldots,s_K\rangle$ is in a product state with the workspace qubits, thus we may measure all the workspace qubits to free up space – it will not ruin the data-array because of the lack of entanglement.

6.8. Summary

Looking at the pseudo code of the algorithms we see that the heaviest operations are sorting and using our qRAM implementation. Note that trivially $K < N < M$ and so the arrays largest component is the one stores the function values. Thus we see that our implementation of Update, Check and Setup requires an extra workspace of size $\mathcal{O}(K \log(M))$. But we already need that order of space to store the function values for the set of size K, so it does not change the magnitude of the number of qubits needed.

Setting $K = N^{2/3}$ and again use that the heaviest operation is that of Check, where we need to sort according the function values we get that the overall circuit depth is $\mathcal{O}(\log(N) \log\log(M))$. Finally, assuming that $M < N^{100}$ we get that the total circuit depth is $\mathcal{O}(\log(N) \log\log(N))$.

Also note that if we allow simultaneous calls to the oracle then our implementation of Setup becomes fully parallel.

Finally, note that this implementation is close to optimal, as a straightforward lower bound shows. We store data of size at least $N^{2/3}$ and as we need to access larger than an exponentially small proportion of this data, one walk step should have circuit depth $> 1/2\log(N)$ if we use bounded in degree gates. As we need to carry out $\sim N^{2/3}$ walk steps this results in a lower bound of $\sim N^{2/3}\log(N)$ in the circuit depth.

7. The MNRS search

The authors of the MNRS paper redefined the transition states of the quantised walk W_P. They used the time reversed walk for the second phase:

$$\forall y \in X : \ \ |l_y\rangle = \textstyle\sum_{x \in X} \sqrt{p^*_{yx}}\, |x\rangle\, |y\rangle \tag{30}$$

This change makes it also necessary to adjust the initial state in terms of the stationary distribution τ of the chain P:

$$|\phi_0\rangle := \sqrt{\tau_x} \sum_{x \in X} |r_x\rangle = \sqrt{\tau_y} \sum_{y \in X} |l_y\rangle$$

The second equality follows from the definition of the time reversed chain, and shows that $|\phi_0\rangle$ is an eigenvector of W_P. Note that these definitions coincide with the definitions of Szegedy in the case when P is symmetric, but they allow us to extend several results for reversible chains.

The discriminant matrix takes for example a nice form. $D(P)_{xy} = \sqrt{p_{xy}}\sqrt{p^*_{yx}}$, using the definition $p^*_{yx} = \tau_x/\tau_y p_{xy}$ we get that $D(P)_{xy} = \sqrt{\tau_x} p_{xy} 1/\sqrt{\tau_y}$. We can write it in a compact form $D(P) = \text{Diag}(\tau) P \text{Diag}(\tau)^{-1}$, which shows that P and $D(P)$ are similar matrices. Further in the case when the Markov chain is reversible then $D(P) = D(P)^T$. It implies that the singular values of $D(P)$ and the eigenvalues of P are the same up to sign in the reversible case.

The other novelty in their approach is that they use W_P directly instead of the leaking walk operator. Basically they follow the logic of Amplitude Amplification, but implement the reflection through the line of the starting state $|\phi_0\rangle$ differently.

They use the phase estimation algorithm to separate the eigenvectors of W_P. If the spectral gap of $D(P)$ is δ then using $\mathcal{O}(1/\sqrt{\delta})$ calls to W_P the phase estimation algorithm can separate the eigenvectors having non-zero imaginary part with high accuracy. After the separation is done the algorithm can flip the amplitude of those eigenvectors, i.e. flip the orthogonal complement of ϕ_0 on the busy subspace. Finally uncomputing the phase estimation does simulate the effect of ref_{ϕ_0} restricted to the busy subspace with high accuracy as well. But the plane in which we would like to simulate the rotation lies in this subspace, so it is enough for us.

$\mathcal{O}(1/\sqrt{\epsilon})$ repetition of the process provides a marked element with high probability after measurement - just as in the case of Amplitude Amplification. Note that ϵ here is the probability that an element is marked according to the stationary distribution τ.

If the Markov chain is reversible then the spectral gap of $D(p)$ is the same as the eigenvalue gap of P. So this algorithm extends the scope of our previous $\sqrt{\delta\epsilon}$ rule significantly.

Theorem 25 (MNRS search) *Let P be a reversible Markov chain, and δ be the eigenvalue gap of P. If for all $M \in \mathcal{M}$ the probability that an element is marked greater then ϵ according to the stationary distribution τ of P then we can decide the* *detection problem with high probability with cost* $\mathcal{O}(S + 1/\sqrt{\delta}(U + C/\sqrt{\epsilon}))$ *using the MNRS search. If* M *is not empty we also find a marked element with high probability.*

This theorem extends Szegedy's results in many ways. It is applicable to a wider class of Markov chains and it can also find a marked element with high probability. Additionally note that the cost structure provided by Szegedy's approach was similar, but slightly worse: $\mathcal{O}(S + 1/\sqrt{\delta\epsilon}(U + C))$.

Although this result extends the scope of Szegedy's $\sqrt{\delta\epsilon}$ rule, it fails preserving the advance that Szegedy's quantum random walk operated in time proportional to the square root of the classical hitting time. This result only relates the running time to the eigenvalue gap of the Markov chain, which can result in possibly larger running time, as we have seen in the case of the hypercube.

Finally Krovi, Magniez, Ozols and Roland showed in 2010 [KMOR] how to find a marked element in the presence of multiple marked elements quadratically faster then the classical hitting time for any reversible Markov chain. They used an interpolation between the leaking walk matrix P_L and the unperturbed chain P to construct their quantum walk.

8. Several applications

First we show how the previous results imply that there is a spatial version of Grover search for several graphs, which is essentially not disturbed by the spatial constraints. This was observed by Szegedy in [Sz1].

Definition ***(Expander graph)*** A regular graph $G = (V, E)$ is called expander with expansion constant $c \in \mathbb{R}^+$ if it has the following property: for every $S \subset V$ with $|S| \leq |V|/2$ it has $|\partial S \setminus S| \geq c|S|$. (Here ∂S denotes the vertices which share a common edge with some $v \in S$)

Expander graphs are easy to traverse, which yields that they have a large eigenvalue gap for the uniform walk. The following theorem of [Lov2] formulates this fact: (Note that in this theorem eigenvalues are sorted without taking the absolute values.)

Theorem 26 (Eigenvalue gap of expander graphs) *If G is a d-regular expander with expansion constant c, then the second largest eigenvalue for the uniform walk on G is at most $1 - c^2/5d$.*

Note that this theorem is useful only in the case when d is small, science c is bounded by 2 due to its definition.

Applying this theorem and using MNRS search with the modified walk $(P + I)/2$ gives the following corollary:

Corollary 27 (Local Grover Search) *Let $d_0, c_0 \in \mathbb{R}^+$ fixed. For d-regular expander graphs with expansion constant c, where $d_0 > d$ and $c > c_0$ we have a local version of Grover Search which finds a marked element in time $\mathcal{O}(1/\sqrt{\epsilon})$ if the proportion of marked elements is at lest ϵ.*

We finally list several problems which can be solved efficiently using quantum walk based algorithms. All the complexities are discussed in the query model, and the input will always be a quantum black box which implements a unitary operation corresponding to the problem.

We have already discussed element distinctness, but now we also know how to find marked elements:

Element Distinctness:

- Black box: Computes f on inputs corresponding to elements of $[n]$
- Question: Are there any $i \neq j \in [n] \times [n]$ such that $f(i) = f(j)$?
- Output: An (i, j) pair with $i \neq j$ and $f(i) = f(j)$ if there is any, otherwise reject.
- Query complexity: Can be solved with high probability using $\mathcal{O}(n^{2/3})$ queries.

The following problem is about to find a triangle i.e. a K_3 in a graph $G = (V, E)$. The algorithm which solves it uses element distinctness as a subroutine.

Triangle Finding:

- Black box: For any pair $u, v \in V \times V$ tells whether there is an edge uv

- Question: Is there any triangle in G?
- Output: A triangle if there is any, otherwise reject.
- Query complexity: Can be solved with high probability using $\mathcal{O}(n^{13/10})$ queries.

Matrix Product Verification:

- Black box: Tells any entry of the $n \times n$ matrices A, B or C.
- Question: Does $AB = C$ hold?
- Output: If not then give i, j indices s.t. $(AB)_{ij} \neq C_{ij}$, otherwise accept.
- Query complexity: Can be solved with high probability using $\mathcal{O}(n^{5/3})$ queries.

References

[AKSz] M. Ajtai, J. Komlós, E. Szemerédi: An O(n log n) sorting network, Proceedings of the 15th Annual ACM Symposium on Theory of Computing, pages 1-9 (1983)

[Amb1] A. Ambainis: Quantum walks and their algorithmic applications, International Journal of Quantum Information, pages 507-518. (2003)

[Amb2] A. Ambainis: Quantum search algorithms, SIGACT News Complexity Column, 35 (2):22-35. (2004)

[Amb3] A. Ambainis: Quantum walk algorithm for element distinctness, Proceedings of the 45th IEEE Symposium on Foundations of Computer Science, pages 22-31 (2004)

[ASh] S. Aaronson, Y. Shi: Quantum lower bounds for the collision and the element distinctness problems, Proc. of the 43th IEEE Symposium on Foundations of CS, pg 513 - 519 (2002)

[BB] L.E. Baum, P. Billingsley: Asymptotic distributions for the coupon collector's problem, Ann. Math. Statist. 36, 1835-1839 (1965)

[Bel] A. Belovs: Applications of the Adversary Method in Quantum Query Algorithms, PhD Thesis, University of Latvia (2013)

[Buh] H. Buhrman, C. Dürr, M. Heiligman, P. Høyer, F. Magniez, M. Santha, R. de Wolf: Quantum algorithms for element distinctness, Proceedings of the 16th Annual IEEE Conference on Computational Complexity, pp. 131-137. (2001)

[ER] P. Erdős, A. Rényi: On a classical problem of probability theory, Magyar Tud. Akad. Mat. Kutató Int. Közl. 6, 215-220 (1961)

[GLM] V. Giovannetti, S. Lloyd, L. Maccone: Quantum random access memory, Phys. Rev. Lett. 100, 160501 (2008)

[GR] Lov Grover, Terry Rudolph: Creating superpositions that correspond to efficiently integrable probability distributions, quant-ph/0208112, arXiv (2002)

[Jor] C. Jordan: Essai sur la géométrie à n dimensions, Bulletin de la Société Mathématique de France, 3:103–174 (1875)

[Jozs] R. Józsa: Quantum Computation Lecture notes, University of Cambridge (2014)

[Kem] J. Kempe: Quantum random walks - an introductory overview, Contemporary Physics, Vol. 44 (4), p.307-327. (2003)

[KMOR] H. Krovi, F. Magniez, M. Ozols and J. Roland: Finding is as easy as detecting for quantum walks, ICALP'10, LNCS vol. 6198, pp. 540-551 (2010)

[Knuth] D. E. Knuth: Combinatorial matrices, Selected Papers on Discrete Mathematics, volume 106 of CSLI Lecture Notes, Stanford University (2003)

[Lov1] L. Lovász: Random Walks on Graphs: A Survey , Combinatorics, Paul Erdős Eighty (Volume 2), Keszthely (Hungary), pp. 1–46. (1993)

[Lov2] L. Lovász: Eigenvalues of graphs, `www.cs.elte.hu/~lovasz/eigenvals-x.pdf` (2007)

[Mey] C. Meyer: Matrix Analysis and Applied Linear Algebra Book and Solutions Manual, North Carolina State University (2000)

[MNRS] F. Magniez, A. Nayak, J. Roland, M. Sántha: Search via quantum walk, Proc. 39th STOC, ACM Press 575-584 (2007)

[NC] M. Nielsen, I. Chuang: Quantum Computation and Quantum Information, Cambridge University Press. (2000)

[Sch] U. Schöning: A probabilistic algorithm for k-SAT and constraint satisfaction problems, Proc. of the 40th IEEE Symposium on Foundations of Computer Science, pages 17-19. (1999)

[SKW] N. Shenvi, J. Kempe, K. Whaley: Quantum random-walk search algorithm, Physical Review A, 67:052307 (2003)

[Sz1] M. Szegedy: Spectra of Quantized Walks and a $\sqrt{\delta\epsilon}$ rule, quant-ph/0401053, arXiv (2004)

[Sz2] M. Szegedy: Quantum Speed-Up of Markov Chain Based Algorithms, Proceedings of the 45th IEEE Symposium on Foundations of Computer Science, pages 32-41 (2004)