Institut für Theoretische Physik
Leibniz Universität Hannover

Quantum algorithms for optimization problems

Bachelor's Thesis

Marlene Funck

10028009

Supervisor:

Prof. Dr. Tobias J. Osborne

September 27, 2022

Contents

1 Introduction

In today's modern world, optimization is a big issue. Not only to increase corporate profits, but also to protect our environment, all conceivable processes must be optimized. With the enormous technological progress of the last decades, computers are an indispensable part of this project. Especially the concept of linear programming has a wide range of applications: Consider a process consisting of a set of decisions meeting certain conditions, each having costs and rewards. As long as the decision's total cost and reward arise from a linear relationship, these decisions can be translated into a linear program. Costs and rewards can be determined by all sorts of factors such as time, money, reputation, risks or emissions, which is why this problem structure allows for a wide range of real-world applications like route planning or production processes. At the scale of today's processes, just a few percent more efficiency can save enormous amounts of money and emissions.

In many use cases, the problem actually consists of integer variables, leading to the special case of integer linear programs. For example, if one has a discrete number of options per decision, as is often the case, these options can be represented by integers.

Against this backdrop, classical optimization algorithms have been greatly refined over the past decades. Today, there are countless approaches such as Dynamic Programming, Branch-and-Bound, Heuristics and combinations like the Concorde TSP solver. However, lately there have been new developments in the field. Quantum Computers are now within reach opening a whole new area of opportunities for optimization: Quantum Computing. Although, both in theory and in practice, this field of research is still in its infancy compared to classical computation, there are already several proven algorithms that can outperform the classical ones. Grover's algorithm, for example, can find a certain element in a list in up to $\sqrt{N}$ steps, while classically N steps are required ([11], p.38). Another example is Shor's algorithm. Solving the Discrete Logarithm problem and the Integer Factorization problem in polynomial time, this algorithm could crack the encryption methods used today ([13], p.1484).

These examples suggest that Quantum Computing has a great potential to improve other optimization algorithms too - in fact, research in this area is running at full speed. Though, at present, the vast majority of the latest discoveries are comprehensible only to people with a background in physics. Thus, they are not usable to the general public. In order to change that, the theoretical results must be made accessible and usable for a wider audience.

A promising example of a quantum optimization algorithm that can solve integer linear programs was published in 2020 by Ashley Montanaro [9]. It is a quantum speedup of Branch-and-Bound algorithms that, in some cases, offers a nearly quadratic speedup over classical approaches.

In this work, Montanaro's algorithm will be explained clearly and with the help of examples to pave the way for its implementation. Therefore, first, a theoretical Background is given explaining the basics of Quantum Computation and Optimization problems. Next, the quantum Branch-and-Bound algorithm by Montanaro is introduced, followed by an explicit example of a Knapsack instance. After that, the

required subroutines, which are both standalone quantum algorithms, are separately introduced giving examples and detailed information about their implementation.

2 Theoretical Background

2.1 Quantum Computation

2.1.1 Qubits, Gates and Circuits

The basis of classical computation and information theory is the concept of bits. In a classical circuit the current either flows, or it does not which can be described by zeros and ones. Analogously, in Quantum Computing there are quantum bits or 'qubits'. Instead of current in a circuit, qubits refer to quantum states (e.g of electron spin or trapped ions). Working with qubits, the underlying concepts and mathematical approaches are those of quantum theory which are assumed to be known by the reader.

In contrast to the classical bit having only two possible states 0 and 1, a qubit can be in any superposition of these:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1}$$

where $\alpha, \beta \in \mathbb{C}$ must fulfill $|\alpha|^2 + |\beta|^2 = 1$ ([11], pp.13-17).

With several qubits, the system is mathematically represented by a tensor product. Thus, the dimension of a Hilbert space $\mathcal{H}_n = \mathbb{C}^{2^n}$ for n qubits is 2^n. For example, the Hilbert space for two qubits has dimension $2^2 = 4$ and one possible basis is

$$\{|0\rangle \otimes |0\rangle, |1\rangle \otimes |0\rangle, |0\rangle \otimes |1\rangle, |1\rangle \otimes |1\rangle\} = \{|00\rangle, |10\rangle, |01\rangle, |11\rangle\}. \tag{2}$$

In classical computers, every operation is built out of logical gates manipulating information and wires transporting the information. Such combinations are called circuits. The AND gate, for example, outputs 1, if all incoming wires have value 1 and outputs 0 otherwise, while the NOT gate swaps the state from 1 to 0 or the other way round. In general, these gates can be represented by truth tables.

In Quantum Computing there are also gates. However, the truth tables defining classical gates are not enough anymore, because superpositions must be included. Now gates have to map normalized elements of a Hilbert space to another normalized element. Thus, gates can be represented as unitary operators.

Creating a quantum NOT gate for example, the classical version defines that $|0\rangle \mapsto |1\rangle$ and $|1\rangle \mapsto |0\rangle$. Using the representation $|0\rangle \hat{=} \binom{1}{0}$ and $|1\rangle \hat{=} \binom{0}{1}$ the following unitary operator executes this map. On an arbitrary superposition $\binom{\alpha}{\beta}$ it acts as follows

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix}. \tag{3}$$

So $\frac{1}{2}|0\rangle + \frac{1}{2}|1\rangle$ for example is mapped to itself. A very important single qubit gate is the Hadamard gate.

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{4}$$

It maps $|0\rangle \mapsto \frac{1}{\sqrt{2}}\big[\,|0\rangle + |1\rangle\big]$ and $|1\rangle \mapsto \frac{1}{\sqrt{2}}\big[\,|0\rangle - |1\rangle\big]$. When applied to several qubits in the state $|0\rangle$, it creates a superposition of all possible states which is very useful in practice ([11], pp.17-19).

Another gate used on two qubits is the controlled-NOT (CNOT) gate. It flips the 'target' qubit, if the 'control' qubit is 1:

$$|00\rangle \mapsto |00\rangle\,,\ |01\rangle \mapsto |01\rangle\,,\ |10\rangle \mapsto |11\rangle\,,\ |11\rangle \mapsto |10\rangle \tag{5}$$

As a matrix this map looks as follows:

$$O_{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \text{ with } |00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, |01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, |10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{6}$$

This action can also be described as $|x, y\rangle \mapsto |x, x \oplus y\rangle$, where $\oplus$ means adding modulo two. The corresponding circuit is shown in Fig. 1.

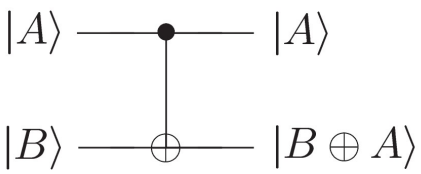

Fig. 1: CNOT gate ([11], p.21)

This concept of a control qubit can be extended for an arbitrary operation U on multiple qubits (Fig. 2). If the control qubit is in state $|0\rangle$, nothing happens and if it is in state $|1\rangle$, U is applied to the target register ([11], pp.20-24).

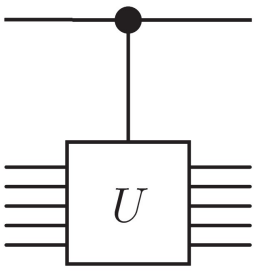

Fig. 2: Control U gate ([11], p.24)

2.1.2 Quantum Fourier Transform

The discrete Fourier Transformation is one of the known operations where quantum computers outperform classical computers. Let $\{|0\rangle, ..., |N-1\rangle\}$ be an orthonormal

basis. Then the discrete Fourier Transformation is defined by

$$|j\rangle \longmapsto \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi ijk/N} |k\rangle = \mathcal{F}(|j\rangle). \tag{7}$$

For an arbitrary state this means

$$\sum_{j=0}^{N-1} x_j |j\rangle \longmapsto \sum_{j=0}^{N-1} x_j \left[\sum_{k=0}^{N-1} e^{2\pi ijk/N} |k\rangle\right] = \sum_{k=0}^{N-1} (\sum_{j=0}^{N-1} x_j e^{2\pi ijk/N}) |k\rangle = \sum_{k=0}^{N-1} y_k |k\rangle . \tag{8}$$

We now set $N = 2^n$. The basis states $|j\rangle$ will now be denoted as $|j_1, j_2, ...j_n\rangle$, where $j_i \in \{0, 1\}$. The binary representation of j then is $j \hat{=} j_1 j_2 ... j_{n-1} \hat{=} j_1 2^{n-1} + .. + j_n 2^0$. Furthermore the notation of binary sections will be used in the following: $0.j_1 j_2 ... j_n \hat{=} \frac{1}{2} j_1 + \frac{1}{4} j_2 + ... + \frac{1}{2^n} j_n$. The above expression (7) can then be transformed to

$$\begin{aligned} &|j_1, j_2, ..., j_n\rangle \\ &\longmapsto \frac{(|0\rangle + e^{2\pi i 0.j_n} |1\rangle) \otimes (|0\rangle + e^{2\pi i 0.j_{n-1} j_n} |1\rangle) \otimes ... \otimes (|0\rangle + e^{2\pi i 0.j_1 j_2 ... j_n} |1\rangle)}{2^{n/2}}. \end{aligned} \tag{9}$$

The quantum circuit to realize this transformation only consists of Hadamard gates and one bit gates of the form

$$R_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{pmatrix}. \tag{10}$$

Applying the Hadamard gate to the initial state's first qubit, due to the fact that $e^{2\pi i 0.j_1}$ is 1 if $j_1 = 0$ and -1 if $j_1 = 1$ the state becomes

$$H \otimes \mathbb{1} \otimes ... \otimes \mathbb{1} |j_1, j_2, ..., j_n\rangle = \frac{1}{2^{1/2}} \Big[|0\rangle + e^{2\pi i 0.j_1} |1\rangle \Big] \otimes |j_2, ..., j_n\rangle . \tag{11}$$

By then applying the R_2 gate controlled by the second qubit, we have an additional factor on the first qubit's $|1\rangle$-part:

$$\begin{aligned} &R_2 \frac{\Big[|0\rangle + e^{2\pi i 0.j_1} |1\rangle \Big] \otimes |j_2, ..., j_n\rangle}{2^{1/2}} \\ &= \frac{\Big[\Big[|0\rangle + e^{2\pi i 0.j_1} |1\rangle \Big] \otimes |0\rangle + \Big[|0\rangle + e^{2\pi i 0.j_1} e^{2\pi i \frac{1}{4}} |1\rangle \Big] \otimes |1\rangle \Big] |j_3, ..., j_n\rangle}{2^{1/2}} \\ &= \frac{\Big[|0\rangle + e^{2\pi i 0.j_1 j_2} |1\rangle \Big] \otimes |j_2, ..., j_n\rangle}{2^{1/2}}. \end{aligned} \tag{12}$$

Continuing with controlled $R_3, R_4, ..., R_n$ gates on the first qubit, analogously the state becomes

$$\frac{\Big[\,|0\rangle + e^{2\pi i 0.j_1 j_2 ... j_n}\,|1\rangle\Big] \otimes |j_2, ..., j_n\rangle}{2^{1/2}}. \tag{13}$$

Repeating this procedure for all n qubits (one Hadamard gate and controlled $R_2, ..., R_{n+1-i}$ are applied to the i^{th} qubit), the output state is as follows:

$$\frac{\Big[\,|0\rangle + e^{2\pi i 0.j_1 j_2 ... j_n}\,|1\rangle\Big] \otimes \Big[\,|0\rangle + e^{2\pi i 0.j_2 ... j_n}\,|1\rangle\Big] \otimes \ ... \otimes \Big[\,|0\rangle + e^{2\pi i 0.j_n}\,|1\rangle\Big]}{2^{n/2}}. \tag{14}$$

The circuit is shown in Fig. 3. The final state equals the Fourier Transformation state in (9) up to a permutation which can be done by at most $n/2$ swapping operations, each requiring three CNOT gates. The transformation itself uses $n + (n-1) + (n-2) + ... + 1 = n(n+1)/2$ gates which leads to a total complexity of order $\mathcal{O}(n^2)$. Meanwhile, the best known classical algorithm (Fast Fourier Transformation) has a complexity of order $\mathcal{O}(n2^n)$ ([11], pp.217-220).

Note that performing QFT once is not sufficient to obtain the complete result. The probabilities of the possible states must be determined, which requires numerous repetitions

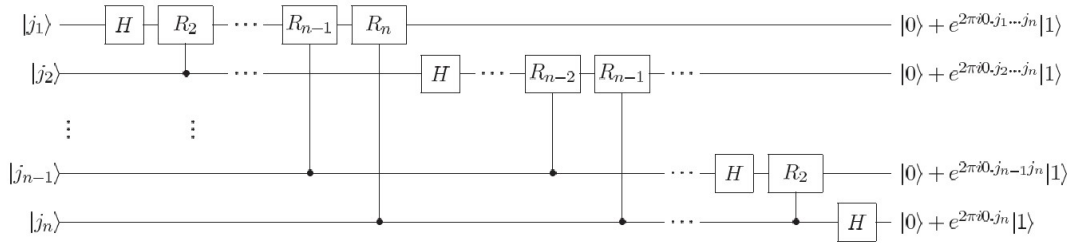

Fig. 3: Quantum Cirquit realising the Quantum Fourier Transformation ([11], p.219)

2.1.3 Phase Estimation

Quantum Phase Estimation (QPE) is another useful tool in Quantum Computing. Building up on the Quantum Fourier Transformation (QFT), the QPE provides a simple way to estimate eigenvalues of a unitary operator.

Let U be a unitary operator on n qubits (on the Hilbert space $\mathcal{H} = \mathbb{C}^{2n}$). Being unitary, U only has eigenvalues u with $|u| = 1$. Thus they may be written in the form $e^{2\pi i \phi}$, where $\phi \in [0, 1]$.

To determine ϕ, the QPE uses two registers. The first register contains t qubits starting in the state $|0\rangle$ and being put into a superposition by applying Hadamard gates to every qubit, while the second register contains n qubits prepared in the

eigenstate $|u\rangle$. In the next step, for every qubit 'i' ($i = 0, ..., t-1$) in the first register, controlled U is applied 2^i times to the second register (Fig. 4), leading to the following outcome:

$$\begin{aligned}\frac{1}{2^{t/2}}\Big[|0\rangle + |1\rangle\Big]^{\otimes t} \otimes |u\rangle \longmapsto \; & \frac{1}{2^{t/2}}\Big[|0\rangle + e^{2\pi i\phi 2^{t-1}}|1\rangle\Big] \otimes ... \otimes \Big[|0\rangle + e^{2\pi i\phi 2^{0}}|1\rangle\Big] \otimes |u\rangle \\ = \; & \frac{1}{2^{t/2}} \sum_{k=0}^{2^t-1} e^{2\pi i\phi k}|k\rangle \otimes |u\rangle \,.\end{aligned} \tag{15}$$

The second equality in (15) already looks like a Fourier Transformation. It results from multiplying out the terms and renaming the 2^n possible combinations $\{|j_1\rangle \otimes ... \otimes |j_t\rangle \,, j_i \in \{0,1\}\}$.

Using the notation of binary sections, assume that ϕ may be expressed exactly as $0.\phi_1\phi_2...\phi_t$. With this, (15) can be rewritten as

$$\frac{\Big[|0\rangle + e^{2\pi i 0.\phi_t}|1\rangle\Big] \otimes \Big[|0\rangle + e^{2\pi i 0.\phi_{t-1}\phi_t}|1\rangle\Big] \otimes ... \otimes \Big[|0\rangle + e^{2\pi i 0.\phi_1\phi_2...\phi_t}|1\rangle\Big]}{2^{n/2}}. \tag{16}$$

This is precisely the outcome of the Fourier Transformation (9) of the state $|\phi_1, ..., \phi_t\rangle$. Thus, now applying the inverse Fourier Transformation to the first register and then measuring it, will reveal the eigenvalue of U to its eigenstate $|u\rangle$. The complete circuit for the Phase Estimation is illustrated in Fig. 5.

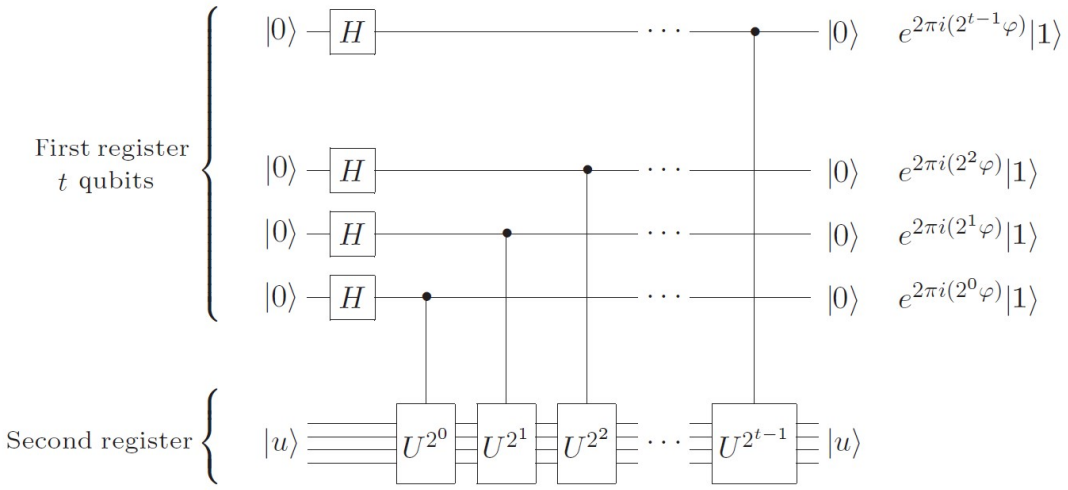

Fig. 4: First part of the Quantum Phase Estimation ([11], p.222)

It was earlier assumed that ϕ can be expressed exactly with t qubits. Since this is not necessarily the case, the QPE will only give an approximated estimation for an arbitrary ϕ: To determine ϕ with an accuracy of 2^{-d} (corresponding to d qubits) allowing a failure probability of at most ϵ, the QPE-circuit requires $t = d + \ln\big(2 + \frac{1}{2\epsilon}\big)$ qubits in the first register ([11], pp.221-225).

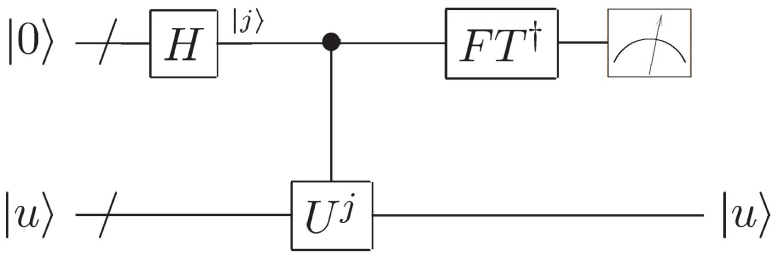

Fig. 5: Complete Quantum Phase Estimation Circuit ([11], p.223)

2.2 Integer Optimization Problems

2.2.1 Integer Linear Programming

Integer linear programming is an optimization problem, where a linear objective function is optimized over integer values. Thus, in case of a maximization the aim is to find

$$max\{c^T x \mid x \in X\}, \tag{17}$$

where $c \in \mathbf{R}^n$ and x is a vector with integer-valued components and dimension n. The set X contains all the possiblie values for x. Constraints can be added to the problem which means that x must fulfill certain conditions to be a 'valid' solution. In that case, the set X only contains those solutions that are valid. ([12], pp.528-530)

If $|X| < \infty$, such problems can always be translated into decision trees with depth n, having one layer per component of x with one branch per option. Likewise, every task consisting of a set of decisions that must fulfill conditions can be translated into such a tree and thereby into an integer linear program.

2.2.2 Knapsack

The Knapsack problem is a special case of an integer linear optimization problem. Given a set of n items each associated with a cost c_i and a reward r_i. The task is to find a subset that maximizes the total reward

$$R = \sum_{i=1}^{n} x_i r_i \tag{18}$$

while keeping the total cost C below a certain maximum C_{max}:

$$C = \sum_{i=1}^{n} x_i c_i \leqslant C_{max}. \tag{19}$$

Here each $x_i \in \{0, 1\}$ denotes if the element is chosen to be in the 'knapsack' or not.

Of course, there are several variants of this problem. One example could be to minimize the cost while keeping the reward higher than a certain minimum ([6], p.24).

2.3 Classical Branch-and-Bound

Branch-and-Bound is an algorithm design for optimization problems applicable to integer linear programming problems that can be translated into a rooted tree. Every node of the tree represents a certain subset of the candidate solutions to the problem: The root would be the whole set and every leaf represents one candidate solution.

The general idea is to explore branches of the tree checking for upper or lower bounds of leaf's costs. If a branch is bounded to solutions with costs worse than the best one found so far, it will be cut off the tree. This way, it is likely that the algorithm does not explore the whole tree, decreasing its total running time.

Put differently, in every step the algorithm splits the total set of candidate solutions into subsets. This is called 'branching'. Then it checks for upper and lower bounds and deletes the subsets that do not contain any improved solutions. Let $cost(v)$ be the function that calculates the lower bound of the solutions corresponding to a certain subtree. For a minimization it must fulfill the condition $cost(v) \leqslant cost(w)$ if w is a child of v. Otherwise its outputs would not be reasonable bounds.

Obviously this method requires efficient estimates of the bounds. If that is not the case, this algorithm degenerates into an exhaustive search ([4], pp.205-219).

2.3.1 Example

As an example consider the following scenario:

Given a list of $n = 7$ items that are each assigned a cost c_i and a reward r_i, the task is to find a subset A of these items that has minimal cost and fulfills the condition

$$R = \sum_{A} r_i \geq 20. \tag{20}$$

items i	1	2	3	4	5	6	7
cost c_i	8	2	3	9	4	3	2
reward r_i	6	3	9	2	5	1	4
r_i/c_i	0.75	1.5	3	0.22	1.25	0.33	2

Tab. 1: Knapsack Example

For each item it needs to be decided, whether it is picked or not. Hence, this problem can be easily translated into a binary tree. Each layer represents one of the items, each node represents a subset of these items and each leaf stands for one candidate solution. The tree in this example has $\sum_{i=0}^{n} 2^i = 255$ nodes and there are $2^7 = 128$ candidate solutions.

The bounds on the cost can be calculated making use of the reward per cost (tab. 1) each item gives: For the upper bound, the items are picked such that the lowest reward per cost are collected first. The last item, picked before the total reward reaches 20, is chosen such that the cost is as high as possible. The lower bound is determined by picking the items in reverse order allowing to pick a 'partial' item, such that the total reward reaches exactly 20. At the end, the total cost is rounded off. In this example the items 1, 2, 3, 4, 5 and 6 sum up to a total reward of 26 and an absolute maximum cost of $c_{max} = 29$. The lower bound is obtained by picking the items 2, 3, 7 and 4/5 of item 5. This yields a minimum cost of $c_{min} = 10$. Analogously, the bounds can be calculated, if a subset of items is already fixed. Thereby, having a set of items that already fulfills $R \geq 20$, adding more items is not allowed (the tree is cut at that node).

To take advantage of the Branch-and-Bound method, at first a valid solution with a certain cost needs to be determined. Thus, the first step is to go through the tree and find one. Thereby it often makes sense, not to randomly browse through any part of the tree, but to take a path that has high potential of delivering a low cost solution. In this example, the first path taken will be the one with the lowest possible lower cost bounds (see Fig. 6).

If the first element is chosen, the lower bound of the cost becomes $c_{low} = 13$. Otherwise, by not picking the first item, the lower bound stays at $c_{low} = c_{min} = 10$. So the first step in the tree goes to the left ('No'). Continuing like that, the next step goes to the right (the second item is picked). Deciding about the third item, something special happens: With the path taken so far, not selecting the third item is not an option anymore, because the condition $R \geq 20$ would not be fulfilled. Therefore, this part of the tree can be cut away already. Proceeding analogously results in the first solution $A = \{2, 3, 5, 7\}$ with a cost of $c_A = 11$. Considering the fact that all lower bounds of the unexplored paths are higher than 11, they can all be cut away. As it turns out, the first solution already is the optimal solution for this optimization problem. Hence, the rest of the tree does not have to be further explored.

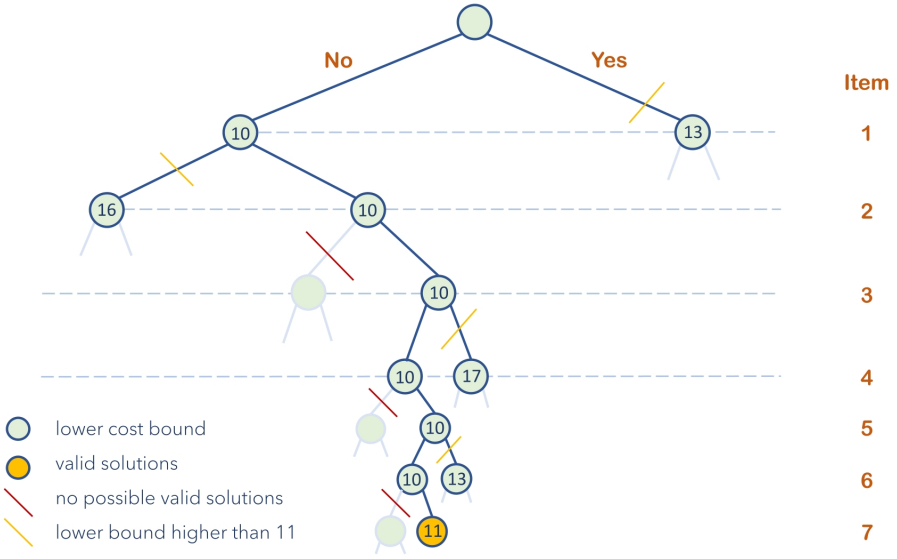

Fig. 6: Example for the classical Branch-and-Bound method

3 Quantum Branch-and-Bound by Montanaro

In his 2020 work Montanaro combined the tree size estimation algorithm [1] and the quantum search algorithm [10] to produce a quantum algorithm with a speedup for general Branch-and-Bound algorithms [9].

3.1 Setting and Requirements

In the following sections the problem discussed will be a minimization. The aim is to find a valid solution ('marked leaf') with minimal cost. In this case, the number of possible solutions increases, if the cost bound at which the tree is cut is increased. Here 'cutting a tree at a cost c' means cutting away all the parts whose lower bounds are higher than c. The size (number of nodes) of this truncated tree will be denoted as T_c. A subset of solutions corresponds to a node in the decision tree. The lower cost bound c on all of the valid solutions in that subtree will be referred to as the node's 'label'.

To make use of the algorithm, one does not need to have access to the whole tree, but to two oracles. Firstly, the oracle $children(v)$ takes a node v as an input and returns the node's children. The second one - $cost(v)$ - returns the lower cost bound of the subtree corresponding to the given node.

The second crucial ingredient of the algorithm are two quantum subroutines named $search_c(\epsilon)$ and $count_c(T_0, \epsilon, \delta)$. They were separately introduced in earlier papers by Montanaro [10] himself in 2015 and by Ambainis and Kokainis [1] in 2017. In Chapters 4 and 5 they will be explained in more detail. For now the following definitions are sufficient:

The fist required subroutine determines, whether there is a valid solution in a tree. It takes as an input the failure probability ϵ and needs access to the oracles of the subtree with a given maximum cost c. With failure probability at most ϵ it outputs

$$search_c(\epsilon) = \begin{cases} \textbf{label of a marked leaf}, \text{ if there is at least one valid solution} \\ \textbf{'not found'}, \text{ if there is no valid solution} \end{cases} . \tag{21}$$

It uses $\mathcal{O}(\sqrt{T}n^{3/2} \ln n \ln 1/\epsilon)$ queries to the oracles and $\mathcal{O}(1)$ other operations, where n denotes the depth of the tree and T is the tee size.

The second subroutine estimates the size of a tree. It takes an estimate of the tree size T_0, a failure probability ϵ and an accuracy δ as an input. It also needs access to the oracle of the subtree with a given cost bound c. Using $\mathcal{O}(\frac{\sqrt{T_0 n}}{\delta^{1/2}} \ln^2 1/\epsilon)$ queries to the oracles and $\mathcal{O}(\ln T_0)$ other operations, with failure probability at most ϵ it gives an output as follows:

$$count_c(T_0, \epsilon, \delta) = \begin{cases} \tilde{\boldsymbol{T}} \textbf{ (such that } |\tilde{\boldsymbol{T}} - \boldsymbol{T}| \leqslant \boldsymbol{\delta T}\textbf{)}, \text{ if } T \leqslant \frac{T_0}{1+\delta} \\ \textbf{'contains more than } \boldsymbol{T_0} \textbf{ nodes'}, \text{ if } T > (1+\delta)T_0 \end{cases} . \tag{22}$$

Here $\tilde{T}$ is the estimated tree size that with failure probability of at most ϵ fulfills the inequation $(1-\delta)T \leq \tilde{T} \leq (1+\delta)T$.

To explain it in more vivid words: $search_c(\epsilon)$ returns the cost of any valid solution in a given subtree, if there is at least one. $count_c(T_0, \epsilon, \delta)$ returns the tree size of a subtree. If the subtree is 'too big' (bigger than T_0) it will stop counting and return 'contains more than T_0 nodes' ([9], pp.1-3).

3.2 Algorithm

The vivid idea of the algorithm is as follows: In order to keep under control the size of the subtree the *search* function has to explore (thereby decreasing the total running time), in each iteration step the algorithm sets a maximum allowed tree size T. Starting with $T = 1$, for each maximum tree size it searches for the highest possible cost bound c_{new} in the interval $[0, c_{max}]$, such that the corresponding truncated tree is 'small enough'. If there is such a subtree, the *search*-function checks whether there is a valid solution in it, and concludes the result with a binary search for the (minimum) cost. If there is no 'small enough' subtree, the allowed tree size is doubled and the next round begins: Fig. 7.

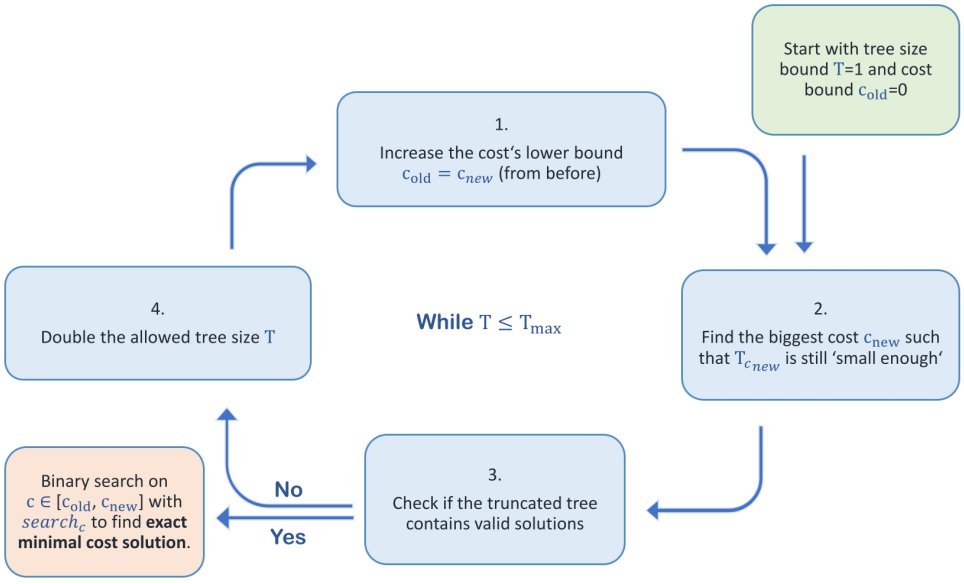

Fig. 7: Schematic illustration of the Quantum Branch-and-Bound Algorithm

The exact algorithm is illustrated in Fig. 8. It takes as an input the upper bound c_{max} on all valid solution's costs which is assumed to be a power of 2, an upper bound T_{max} on the size of the whole tree and a failure probability ϵ. Moreover, it needs access to the oracles and subroutines discussed above.

As it is also marked in Fig. 7 and 8, besides the initialization, the algorithm basically consists of four steps:

Initialization: The lower bound of the first observed cost interval is set to be $c_{old} = 0$. The first allowed tree size is $T = 1$. Basically the smallest possible values are chosen, so that the algorithm works for a general scenario.

1. If this is the 'last round' of the loop (i.e. $T > T_{max}/2$), the upper limit of the cost interval to be observed in the third step will be the maximal cost: $c_{new} = c_{max}$. If not, the upper limit is calculated in step 2.

2. To make this step easier, we assumed the maximum cost c_{max} to be a power of 2. Starting with the interval $[0, c_{max}]$, in every iteration step the size of the tree truncated at the cost in the middle of the interval is checked. If this tree is small enough, the upper half of the current interval will be investigated in the next step (the possible costs and thereby the tree sizes increase). If it is too big, the lower half is checked (the possible costs decrease). This way, the size of the interval of costs that are considered halves with every iteration, leading to the biggest possible cost whose corresponding tree is still 'small enough'. This is fulfilled, if $count_c(T, \epsilon, \delta)$ does not return 'contains more than T nodes'. See Fig. 9.

3. In the third step it is checked, if the tree truncated at the found cost c_{new} contains solutions by running $search_{c_{new}}(\epsilon')$. If it does contain a solution, the function returns its label. A binary search of costs $c \in [c_{old}, c_{max}]$ over $search_c(\epsilon')$ then reveals the final result (the algorithm stops).

4. Preparing the next iteration, the allowed tree size is increased to $T = 2T$ and the lower limit of the cost interval is increased to $c_{old} = c_{new}$ ([9], pp.1-3).

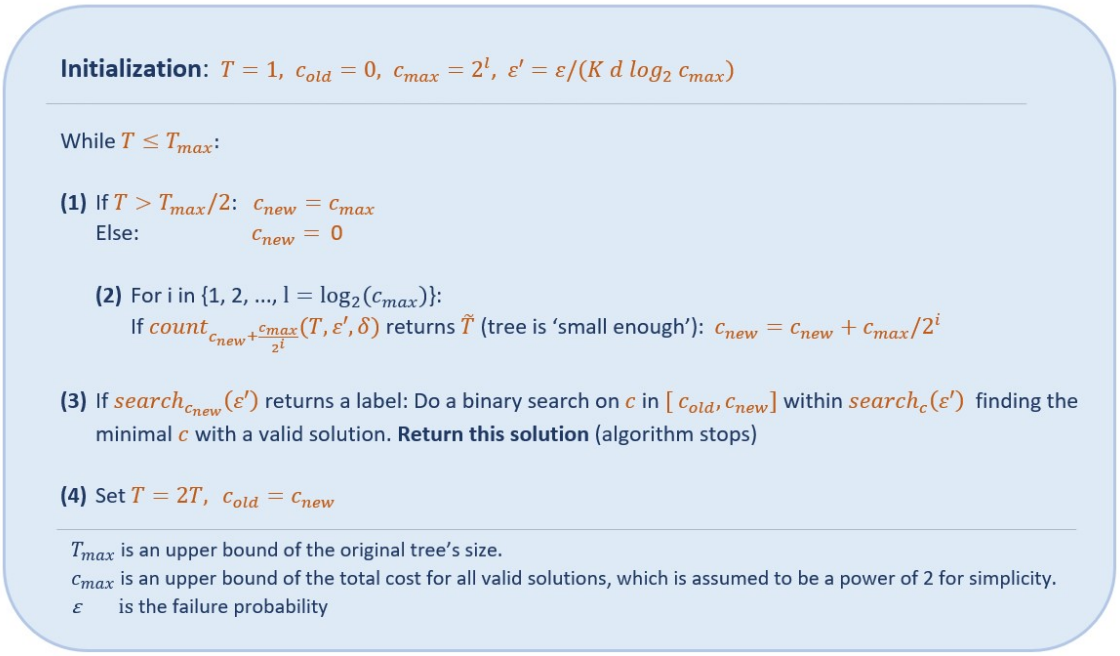

Fig. 8: Quantum Branch-and-Bound Algorithm ([9], p.3)

3.3 Example

Consider the knapsack example, given in Chapter 2.3.1. The maximum cost $c_{max} = 29$ is now increased, such that it is a power of two: $c_{max} = 32 = 2^5$. Thus, the iteration

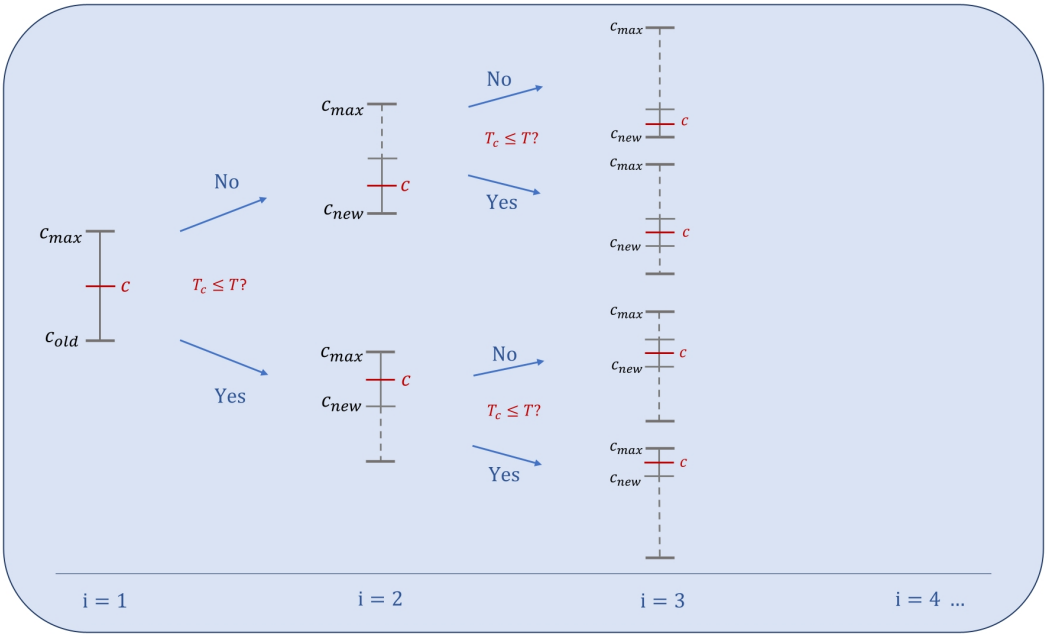

Fig. 9: Iteration to reduce the interval size

in step 2 of the algorithm will go from $i = 1$ to $i = 5$. The relevant parts of the tree for this algorithm is shown in Fig. 10. Here the nodes marked with a lower bound '∞', are the ones that correspond to invalid solutions ($R \geq 20$ is not achievable).

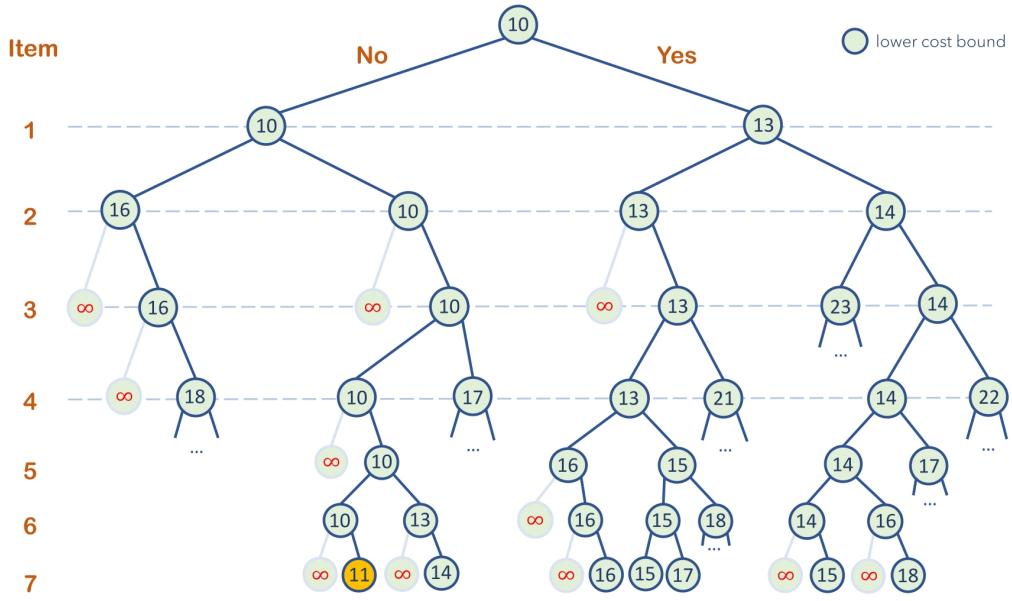

Fig. 10: Example tree for a Knapsack problem

The algorithm starts with $T = 1$ and $c_{old} = 0$. In step 1 c_{new} is set to be 0. Now by using the procedure decribed in Fig. 9, the for-loop of step 2 runs the $count_c$ function for several costs c in order to find the maximum c such that the tree truncated at this cost is still smaller than T. Assuming that the $count_c$ function works perfectly,

the following happens:

$$
\begin{aligned}
& & i=1: \quad & T_{c_{new}+\frac{c_{max}}{2^i}} = T_{16} = 29 > 1 & & \\
& & i=2: \quad & T_{c_{new}+\frac{c_{max}}{2^i}} = T_{8} \;\; = \;\; 0 \leq 1 & \Rightarrow \quad & c_{new} = 8 \\
& T=1: & i=3: \quad & T_{c_{new}+\frac{c_{max}}{2^i}} = T_{12} = \;\; 8 > 1 & & \\
& & i=4: \quad & T_{c_{new}+\frac{c_{max}}{2^i}} = T_{10} = \;\; 7 > 1 & & \\
& & i=5: \quad & T_{c_{new}+\frac{c_{max}}{2^i}} = T_{9} \;\; = \;\; 0 \leq 1 & \Rightarrow \quad & c_{new} = 9
\end{aligned}
\tag{23}
$$

Hence, $c_{new} = 9$ should be the biggest cost, such that the truncated tree has one or less nodes. This can be easily verified, by looking at Fig. 10: For $c = 10$ there are several nodes with a lower bound 10, whereas there are no label showing 9 or smaller. The sizes of the truncated trees T_{16}, T_8, ... can be determined as it is shown in Fig. 11, 12, 13, 14 and 15.

Next, step 3 of the algorithm checks, whether the tree truncated at cost $c_{new} = 9$ contains a valid solution. It does not. Therefore, in the last step T is doubled and c_{old} is updated to be $c_{old} = c_{new} = 9$.

For $T = 2$ and $T = 4$ the same happens as it does for $T = 1$: Because $T_c = 0$ for all cost bounds $c < 10$ and $T_c = 7$ for $c = 10$, the iteration in step 2 will give the same result as before: $c_{new} = 9$. The *search* in step 4 will therefore not find a solution within the truncated tree resulting solely in a doubling of T.

The first interesting thing happens for T=8. With c_{old}=9 and $c_{new} = 0$, the iteration in step 2 leads to the following:

$$
\begin{aligned}
& & i=1: \quad & T_{c_{new}+\frac{c_{max}}{2^i}} = T_{16} = 29 > 8 & & \\
& & i=2: \quad & T_{c_{new}+\frac{c_{max}}{2^i}} = T_{8} \;\; = \;\; 0 \leq 8 & \Rightarrow \quad & c_{new} = 8 \\
& T=8: & i=3: \quad & T_{c_{new}+\frac{c_{max}}{2^i}} = T_{12} = \;\; 8 \leq 8 & \Rightarrow \quad & c_{new} = 12 \\
& & i=4: \quad & T_{c_{new}+\frac{c_{max}}{2^i}} = T_{14} = 19 > 8 & & \\
& & i=5: \quad & T_{c_{new}+\frac{c_{max}}{2^i}} = T_{13} = 13 > 8 & &
\end{aligned}
\tag{24}
$$

For $T = 8$, the cost bound is $c_{new} = 12$. Thus, in step 3, the $search_{c=12}$ function runs on the subtree shown in Fig. 14 which actually contains a valid solution. Hence, the last thing to do is a binary search on the cost $c \in [9, 12]$ over $search_c$:

$$
\begin{aligned}
& search_{11}: \text{ valid node } \Rightarrow \quad c_{next} = 9 + \frac{11-9}{2} = 10 \\
& search_{10}: \text{ no valid node}
\end{aligned}
\tag{25}
$$

This yields the same result as the classical algorithm found in Chapter 2.3.1. The optimal (minimum) cost achievable in this scenario is $c = 11$.

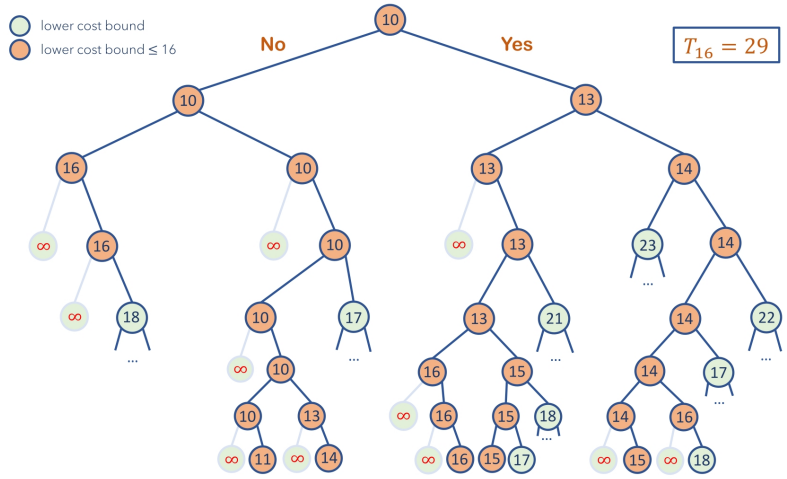

Fig. 11: Determining T_{16}

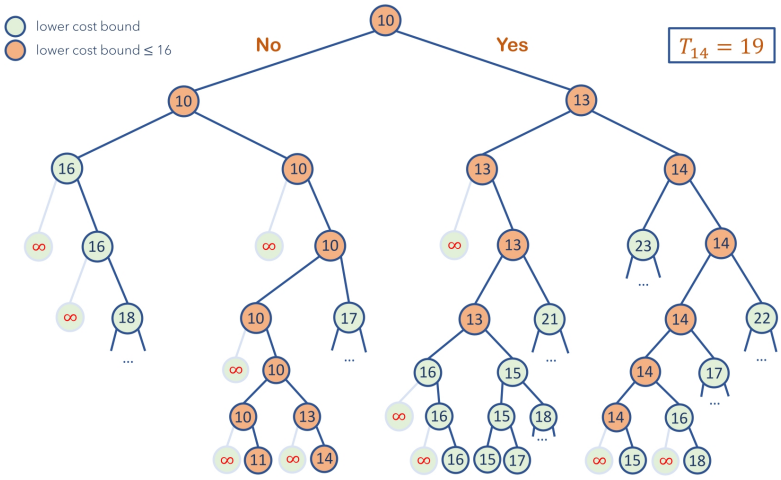

Fig. 12: Determining T_{14}

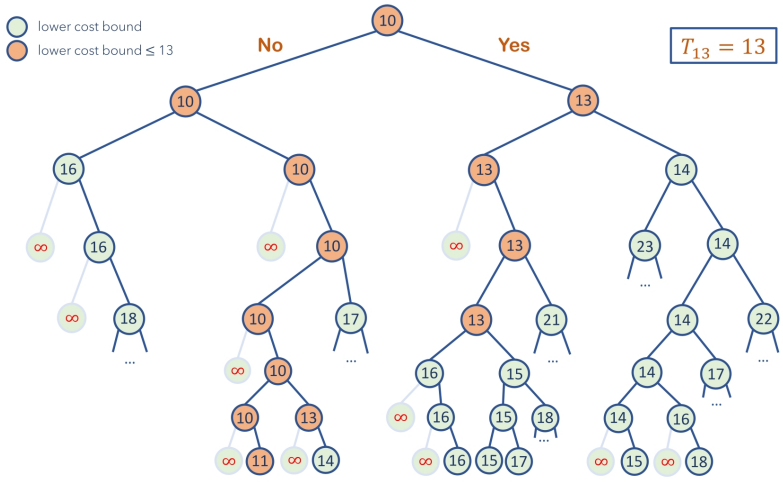

Fig. 13: Determining T_{13}

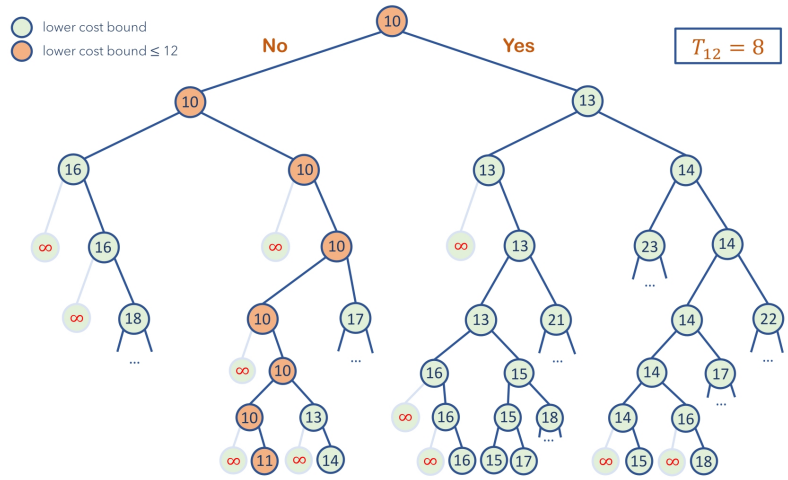

Fig. 14: Determining T_{12}

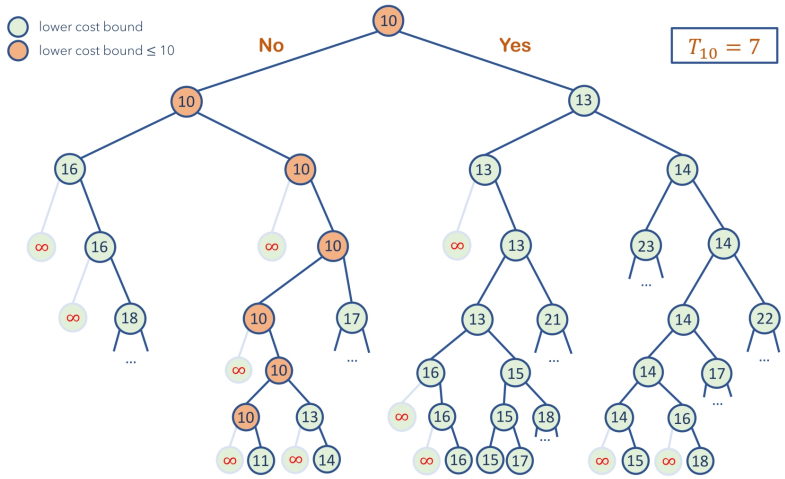

Fig. 15: Determining T_{10}

3.4 Speedup

The time complexity of the Quantum Branch-and-Bound algorithm results from those of the used subroutines:

The Backtracking algorithm behind the function $search_c(\epsilon)$ uses $\mathcal{O}(\sqrt{T}n^{3/2} \ln n \ln 1/\epsilon)$ queries to the oracles and $\mathcal{O}(1)$ other operations, where n denotes the depth of the tree, ϵ is the failure probability and T is the tree size.

The Quantum tree size estimation acting as $count_c(T_0, \epsilon, \delta)$ needs $\mathcal{O}(\frac{\sqrt{T_0 n}}{\delta^{1/2}} \ln^2 1/\epsilon)$ queries and $\mathcal{O}(\ln T_0)$ other operations ([9], p.3).

With these subroutines and the stated algorithm, it is shown that the overall number of queries is $\mathcal{O}\left(\sqrt{T_{c_{min}} n} \ln(c_{max}) \ln \frac{n \ln(c_{max})}{\epsilon} \times \left[\ln \frac{n \ln(c_{max})}{\epsilon} + n \ln n\right]\right)$.

Montanaro shows a more concrete acceleration using the Sherrington-Kirkpatrick model as an example. This model contains Hamiltonians of a certain form and the aim

is to find their ground state energies. Solving this problem with a classical Branch-and-Bound algorithm, for the large majority of trees with depth n corresponding to the size of the problem, the running time is $\mathcal{O}(2^{0.451n})$ steps. The quantum Branch-and-Bound, on the other hand, has complexity $\mathcal{O}(2^{0.226n})$ on the same instances providing a nearly quadratic speedup ([9], pp.4-5).

3.5 Possible Improvements

Montanaro states that by further developing the ideas of Jarret and Wan [5] it might be possible to decrease the complexity's depth dependency by a factor of n ([9], p.4). Moreover Ambainis and Kokainis ([1], pp.992-993) present an approach to improve Montanaro's Backtracking algorithm [10]. However, Montanaro doubts that their idea of improvement is effective in practice ([9], pp.4-5).

4 Quantum Backtracking by Montanaro

In 2016 [10] Montanaro proves that there is a quantum algorithm which works similar to backtracking, providing a quadratic speedup over its classical equivalent. The algorithm uses a quantum walk based on a special case of the earlier work of Aleksandrs Belovs [3].

4.1 The Classical Backtracking

Backtracking is a technique for solving computational problems such as constraint satisfaction problems. By exploiting knowledge about partial solutions, the algorithm can sort out sets of candidate solutions without fully calculating them, reducing the computational effort required.

Obviously, this method depends on additional knowledge about partial solutions. Therefore a predicate for determining whether a complete solution is valid expanded to partial solutions is required. For a decision tree with degree d and n levels, it is defined as follows:

$$P(\mathbf{x}) = \begin{cases} \textbf{True}, \text{ if } \mathbf{x} \text{ is valid} \\ \textbf{Indeterminate}, \text{ if } \mathbf{x} \text{ can be extended to a valid solution} \\ \qquad\qquad\qquad\qquad \text{(or if it is unknown whether it can be)} \\ \textbf{False}, \text{ if } \mathbf{x} \text{ is invalid or cannot be extended to a valid solution} \end{cases} \tag{26}$$

where x is a (partial) solution and being valid means either x itself is a complete valid solution or all its branches only lead to valid solutions.

Let $x \in \{0, 1, ..., d-1, *\}^n =: \mathcal{D}$ denote a partial assignment using stars $(*)$ to represent unassigned values. Furthermore, it is necessary to know the assignment's sequence which is given by the function $h : \mathcal{D} \to \{0, ..., n\}$. For any assignment x, it gives the index of the next variable to be assigned to a value. From now on, partial assignments, that can be expanded to a valid solution will be called 'valid' and otherwise 'invalid'.

A concrete version of a classical Backtracking algorithm is shown in Fig. 16. The idea is to check in every step, if a partial assignment can be completed to a valid solution. Starting at the root, all possible 'first' assignments are checked for their validity. If they are not valid, this branch will not be further exploited. If they are, all possible 'next' assignments are checked in the same manner. This procedure continues until the algorithm reaches all complete valid solutions (marked leafs of the tree) and returns them. If there are no valid solutions, there will be no output.

Assuming that P and h both are running in polynomial time, the complexity of classical Backtracking depends mostly on the tree size which grows exponentially in n ([10], pp.2-3). There are several successful applications of this method such as the 'Chess Problem', 'Fixed Length Error Correcting Codes' or the 'Social Golfer Problem' all of which are constrained satisfaction problems ([12], pp.361-362).

Assume that we are given access to a predicate $P : \mathcal{D} \to \{\text{true}, \text{false}, \text{indeterminate}\}$, and a heuristic $h : \mathcal{D} \to \{1, \ldots, n\}$ which returns the next index to branch on from a given partial assignment.

Return `bt`$(*^n)$, where `bt` is the following recursive procedure:

`bt`(x):

1. If $P(x)$ is true, output x and return.
2. If $P(x)$ is false, or x is a complete assignment, return.
3. Set $j = h(x)$.
4. For each $w \in [d]$:
   (a) Set y to x with the j'th entry replaced with w.
   (b) Call `bt`(y).

Fig. 16: Classical Backtracking Algorithm ([10], p.3)

4.2 Setting and Requirements

For the quantum algorithm, now consider an implicitly defined tree with tree size T and depth n, whose vertices are labelled $r, 1, ..., T-1$. Here r denotes the root, which is assumed not to be marked (i.e. there are solutions in the tree that are not valid). Let h and P be the oracles explained above. Furthermore assume that the distance $l(x)$ from a vertex x to the root can be determined for every x without knowing the whole tree. Let d_x be the degree of a vertex x, meaning the number of vertices connected to it. In the case of a tree, the degree is the number of children plus one: $d_x = |\{y : x \to y\}| + 1$ and for the root it is $d_r = |\{y : r \to y\}|$, where $x \to y$ means that y is a child of x. In the following it will be necessary to divide all vertices into two disjoint sets $A = \{x : l(x) \text{ even}\} \cup \{r\}$ and $B = \{x : l(x) \text{ odd}\}$.

The quantum walk is realised by using so called diffusion operators D_x defined below. What they do essentially is to locally create superpositions of states (diffusing):

$$D_x = \begin{cases} \mathbb{1}\,, & \text{if } x \text{ is marked} \\ \mathbb{1} - 2\left|\psi_x\right\rangle\left\langle\psi_x\right|, & \text{if } x \text{ is not marked} \\ \mathbb{1} - 2\left|\psi_r\right\rangle\left\langle\psi_r\right|, & \text{if } x \text{ is the root} \end{cases} \tag{27}$$

with

$$\begin{aligned} |\psi_x\rangle &= \frac{1}{\sqrt{d_x}}\Big[\,|x\rangle \;+ \sum_{\{y:x\to y\}} |y\rangle\Big] \\ |\psi_r\rangle &= \frac{1}{\sqrt{1+nd_r}}\Big[\,|r\rangle \;+ \sqrt{n} \sum_{\{y:r\to y\}} |y\rangle\Big]. \end{aligned} \tag{28}$$

The Hilbert space acted on is $\mathcal{H} = span(\{|r\rangle, |1\rangle, ..., |T-1\rangle\})$. Each node of the tree corresponds to one of basis states.

For each state $|x\rangle$, D_x only acts on triangles in the tree - the subspace spanned by $|x\rangle$ itself and its children - creating a superposition out of them. If the node x is

marked (all its children lead to valid solutions only), the corresponding diffusion operator does nothing.

These diffusion operators are composed as direct sums, spreading out the diffusion over the entire tree:

$$R_A = \bigoplus_{x \in A} D_x \quad \text{and} \quad R_B = |r\rangle\langle r| + \bigoplus_{x \in B} D_x. \tag{29}$$

All $D_{x \in A}$ act on disjoint subspaces, therefore R_A being their direct sum maintains the property of only diffusing on seperate triangles, although it acts on the whole tree. The same reasoning applies to R_B, of course. Fig. 17 visualizes these triangles for both R_A and R_B on an examplary tree ([10], pp.9-10).

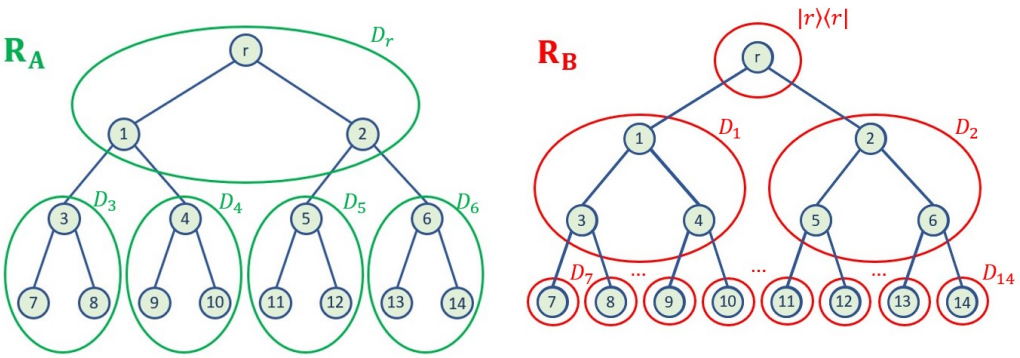

Fig. 17: Example for diffusion subspaces of the Operators R_A and R_B

The idea of the quantum walk is to start at the root, spreading out from there and stopping at the nodes that are marked (those in whose branch any solution will be valid). The product $R_B R_A$ can do exactly this, when used repeatedly: Applied to the root-state $|r\rangle$, $R_B R_A$ will create a superposition of the root, its children and grandchildren. Thus, applying $R_B R_A$ will each time push the quantum walk two levels further in the tree.

4.3 Algorithm

The essence of this algorithm is that the eigenvalue of $R_B R_A$ is directly related to the existence of a marked node: According to Lemma 1 (see Appendix), applying Phase Estimation to $R_B R_A$ on the state $|r\rangle$ will give 1 with probability $Pr(1) \geq 1/2$, if there is a marked node and with $Pr(1) \leq 1/4$, if there is no marked node. The algorithm for detecting a marked vertex is then as follows (Fig. 18).

The idea of the algorithm simply is to check, if $R_B R_A$ has one eigenvalue one. In order to realise a total failure probability of at most ϵ, the Phase Estimation has to be applied $K = \gamma \ln(1/\epsilon)$ times with precision $\beta/\sqrt{T_{max} n}$. The precision is defined within Lemma 2 (see Appendix). If thereby the phase is estimated to be 1 at least $3K/8$ times, the algorithm returns '*marked vertex exists*'. Here γ and β are both

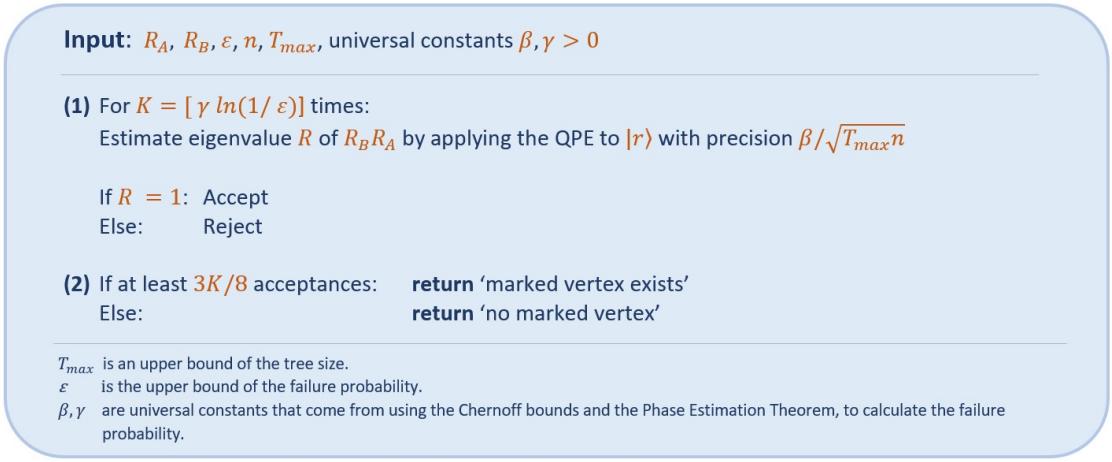

Fig. 18: Quantum Backtracking Algorithm ([10], p.10)

universal constants. β is calculated in the proof of Lemma 3 and γ is determined in the calculation below.

Given that the Quantum Backtracking Algorithm uses the Phase Estimation $\mathcal{O}(\ln(1/\epsilon))$ times with precision $\mathcal{O}(1/\sqrt{Tn})$, the total use of R_A and R_B is of order $\mathcal{O}(\sqrt{Tn}\ln(1/\epsilon))$ each ([10], pp.9-10).

The validity of the failure probability ϵ can be proven using the Chernoff bound. Having independent random variables $X_i \in \{0, 1\}$ the following inequalities hold for any $\delta > 0$:

$$\begin{aligned} Pr(X \leq (1-\delta)\mu) &\leq e^{-\delta^2\mu/2} \\ Pr(X \geq (1+\delta)\mu) &\leq e^{-\delta^2\mu/(2+\delta)}, \end{aligned} \tag{30}$$

where X is the sum $\sum_i X_i$ over all random variables and μ is the expected value of this sum ([8], pp.68-70). Performing Phase Estimation several times, the measured eigenvalues will be independent. Representing all eigenvalues $\neq 1$ by "0", this is a suitable use case of the Chernoff bounds. Thus, the statement is proven, if the following equivalences are verified:

$$\text{If there is a marked vertex: } \epsilon \overset{!}{\geq} Pr(X \leq 3K/8) \Leftrightarrow K = \gamma \ln(1/\epsilon) \tag{31}$$

$$\text{If there is no marked vertex: } \epsilon \overset{!}{\geq} Pr(X \geq 3K/8) \Leftrightarrow K = \gamma' \ln(1/\epsilon) \tag{32}$$

Proof : First calculate the bounds of μ (using the result from Lemma 1).

$$\mu = E[\sum_{i=1}^{K} X_i] = \sum_{i=1}^{K} E[X_i] = \sum_{i=1}^{K} p_i X_i = K p_i \begin{cases} \geq K/2, \text{ if marked vertex exists} \\ \leq K/4, \text{ if no marked vertex exists} \end{cases} \tag{33}$$

From inserting $(1 \pm \delta)\mu = 3K/8$ into (30) it follows

$$(1 \mp \delta)\mu = 3K/8 \quad \Leftrightarrow \quad \delta = \pm 1 \mp \frac{3K}{8\mu} \begin{cases} \geq 1/4, \text{ if marked vertex exists} \\ \geq 1/2, \text{ if no marked vertex exists} \end{cases} . \tag{34}$$

With these found inequalities, (31) and (32) can be derived:

$$Pr(X \leq 3K/8) \leq e^{\frac{-\mu\delta^2}{2}} \leq e^{-\frac{K}{64}} =: \epsilon \quad \Leftrightarrow \quad K = 64 \ \ln(1/\epsilon) \tag{35}$$

$$\begin{gathered} Pr(X \geq 3K/8) \leq e^{-\delta^2\mu/(2+\delta)} = \ e^{-(\frac{3K}{8}-\mu)^2/(\mu+\frac{3K}{8})} \leq e^{-K/40} =: \epsilon \\ \Leftrightarrow \\ K = 40 \ \ln(1/\epsilon) \end{gathered} \tag{36}$$

Thus, choosing $\gamma = 64$, the statement is right for both cases. However, it is likely that there are also smaller possible values. □

4.4 Implementing R_A and R_B

The main challenge in applying the Quantum Backtracking Algorithm is the implementation of the operators R_A and R_B. This is realised by another algorithm given in Montanaro's paper ([10], pp.16-18).

The algorithm in Fig. 19 shows how the operator R_A can be implemented. A partial assignment $x \in \{0, 1, ..., d-1, *\}^n$ is represented by the tuple

$$(i_1, v_1), ..., (i_l, v_l), (0, *)^{n-l} \tag{37}$$

where $l \in \{0, 1, ..., n\}$ is the number of assigned variables and $v_k \in \{0, 1, ..., d-1\}$ denotes the value assigned to the variable with index $i_k \in \{1, 2, ..., n\}$. All values, not yet assigned are denoted by the tuple $(0, *)$. The child of an assignment (node) x will then be $(i_1, v_1)...(i_n, v_n)(j, \omega)$, where $j = h(x)$ is the index of the variable to be assigned next and $\omega \in \{0, 1, ..., d-1\}$.

Now define the state $|\psi_{\alpha,S}\rangle$ as

$$|\psi_{\alpha,S}\rangle := \frac{1}{\sqrt{\alpha|S|+1}} \Big[|*\rangle + \sqrt{\alpha} \sum_{i \in S} |i\rangle \Big]. \tag{38}$$

by setting S to be the set of a node's (allowed) children and $\alpha = 1$ or $\alpha = n$, this corresponds to the states $|\psi_x\rangle$ and $|\psi_r\rangle$ defined in (28).

Aiming to implement $R_A = \bigoplus_{x \in A} D_x$, the first step is to realise that for every basis element $|y\rangle$ exactly one of these D_x effectively acts on it: If y is in an even distance from the root, D_y will act on this node. If the distance is odd, the diffusion operator of the node's parent will be the one acting on it. Thus, on a pure state y, R_A

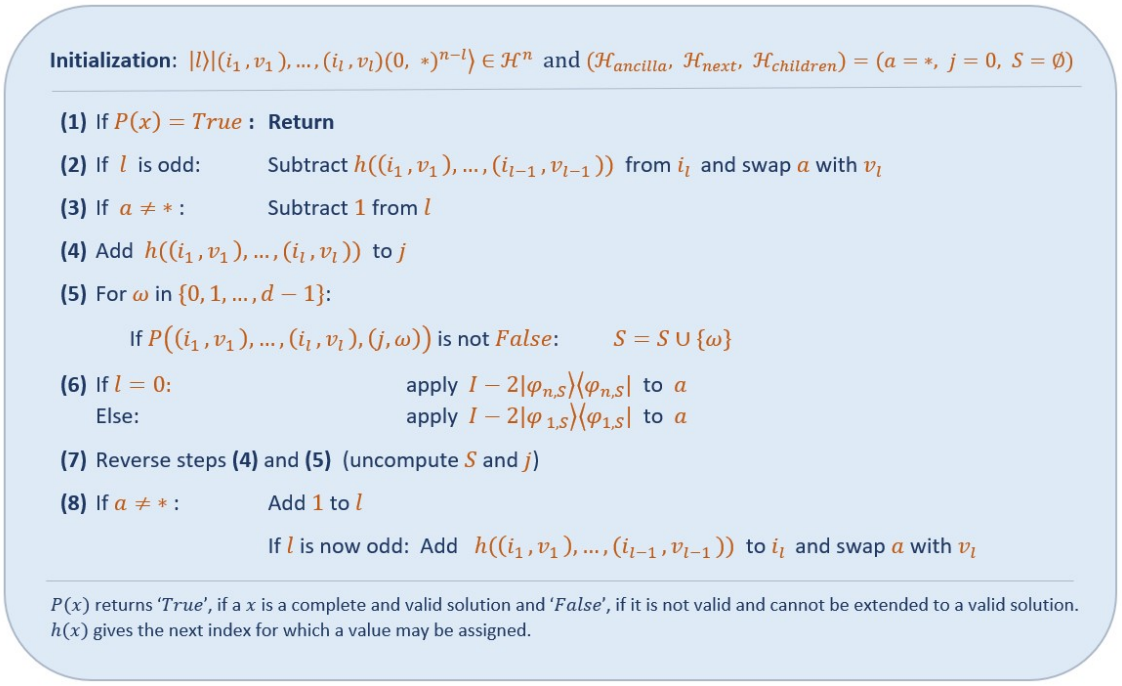

Fig. 19: Implementation of the operator R_A ([10], p.17)

effectively acts as one of the D_x-operators. This is exploited in the algorithm below (see Fig. 19).

Due to the definition (27), the first step of the algorithm is to check whether the node corresponding to the given assignment x is marked. If so, R_A acts as the identity, realised by immediately returning.

If x is not marked, in steps 2 and 3 the algorithm checks whether the input node is in an even distance from the root. If it is not, the state is changed to its parent and the previous value is stored in the ancilla register $\mathcal{H}_{ancilla}$.

In the next two steps (4 and 5), the node's children are determined, checked for their validity and added to the set S, if they are valid.

After that, in step 6 the effect of D_x is realised by applying the operator $\mathbb{1} - |\psi_{\alpha,S}\rangle\langle\psi_{\alpha,S}|$ to the ancilla state a. In order to realise definition (27) and (28), it is $\alpha = n$ for $l = 0$ and $\alpha = 1$ otherwise. Only the most recently assigned value (saved in the ancilla register $\mathcal{H}_{ancilla}$) is affected: One part stays undefined ($*$) and the other parts are changed to be the node's valid children. Therefore, this step effectively realises the whole diffusion operation.

In step 7, j and S are set to be their initial values, while in step 8 the output state is constructed by adapting the last assignment: The values saved in the ancilla register $\mathcal{H}_{ancilla}$ are set to be the 'last assigned' values of the state, the corresponding index is calculated and l state is adapted. After all that, a is set to be '$*$' again ([10], pp.16-18).

An input state of this algorithm has the form

$$|l\rangle_{\mathbb{C}^{n+1}} \left|(i_1, v_1)...(i_l, v_l)(0,*)^{n-l}\right\rangle_{C^{n+1}\otimes C^{d+1}} |a=*\rangle_{\mathcal{H}_{ancilla}} |j=0\rangle_{\mathcal{H}_{next}} |S=\emptyset\rangle_{\mathcal{H}_{children}}, \tag{39}$$

where the first register holds the number of assigned values, the second one holds the assignment, $\mathcal{H}_{next}$ contains the index of the variable to be assigned next (dimension n) and $\mathcal{H}_{children}$ holds the children, where the set S of the children can be represented with one qubit per element in the set of possible values (children). $\mathcal{H}_{ancilla}$ is the ancilla register used for the swaps and has dimension $d+1$.

For each use of R_A or R_B $\mathcal{O}(1)$ operations are required ([10], p.18).

4.4.1 Example

For example, given the input $|l=2\rangle\,|(1,0)(2,0)(0,*)...\rangle\,|a=*\rangle\,|j=0\rangle\,|S=\emptyset\rangle$ corresponding to the orange node in Fig. 20 and assuming that the set of assignment options is $\{0,1\}$ for every level in the tree, the algorithm will work as follows: Steps 1,2 and 3 are skipped, because the chosen node is not marked and it has even distance to the root. Step 4 and 5 then add $\{0\}$ and $\{1\}$ to the set S, because both options are valid.

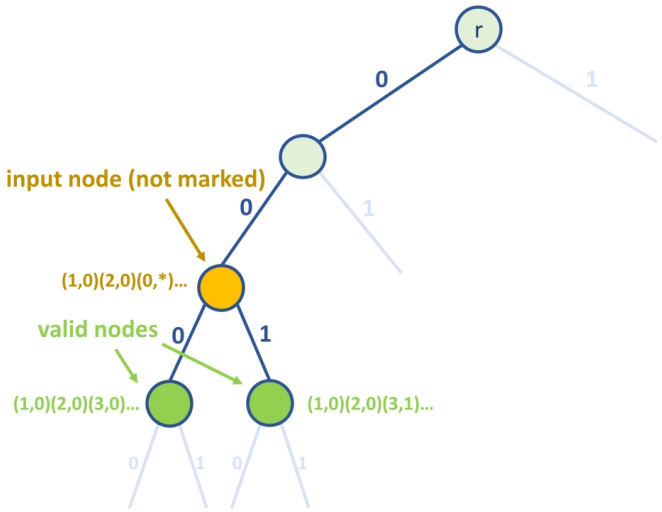

Fig. 20: Example for the implementation of R_A

The set S, for example, can be realised with one qubit per element in the set of possible values (children). Then a qubit is in the state $|1\rangle$, if the corresponding value is in S and in the state $|0\rangle$ otherwise. In this case $S=\{0,1\}$ is represented by the state $|11\rangle$.

In step 6, the diffusion operation is applied to the ancilla register $|a = *\rangle$:

$$\begin{aligned}\left[\mathbb{1} - 2\left|\psi_{1,\{0,1\}}\right\rangle\left\langle\psi_{1,\{0,1\}}\right|\right]|a = *\rangle &= \left[\mathbb{1} - \frac{2}{2+1}\Big[\,|*\rangle + |0\rangle + |1\rangle\Big]\Big[\,\langle *| + \langle 0| + \langle 1|\Big]\right]|*\rangle \\ &= |*\rangle - \frac{2}{3}\Big[\,|*\rangle + |0\rangle + |1\rangle\Big] \\ &= \frac{1}{3}|*\rangle - \frac{2}{3}\Big[\,|0\rangle + |1\rangle\Big]\end{aligned} \tag{40}$$

After changing j and S back to their initial states, step 8 realises the final transformation:

$$\begin{aligned}&|l = 2\rangle\,|(1,0)(2,0)(0,*)...\rangle\left[\frac{1}{3}|*\rangle - \frac{2}{3}(|0\rangle + |1\rangle)\right]|j = 0\rangle\,|S = \emptyset\rangle \\ \longmapsto\quad &\frac{1}{3}|l = 2\rangle\,|(1,0)(2,0)(0,*)...\rangle\,|*\rangle\,|j = 0\rangle\,|S = \emptyset\rangle \\ &-\frac{2}{3}|l = 3\rangle\,|(1,0)(2,0)(3,0)...\rangle\,|*\rangle\,|j = 0\rangle\,|S = \emptyset\rangle \\ &-\frac{2}{3}|l = 3\rangle\,|(1,0)(2,0)(3,1)...\rangle\,|*\rangle\,|j = 0\rangle\,|S = \emptyset\rangle\end{aligned} \quad . \tag{41}$$

This output state corresponds exactly to the desired diffusion the operator R_A is supposed to create, while all the other registers again have their initial values.

The algorithm for $R_B = |r\rangle\langle r| + \bigoplus_{x\in B} D_x$ is similar. However, a few changes need to be considered: In the first step, the algorithm has to return not only, if $P(x)$ is true, but also when $l = 0$. This is necessary, because R_B acts as the identity on the root. This leads to the fact that the case '$l = 0$' in step 6 can be removed. Additionally, in steps 2 and 8 'odd' has to be replaced by 'even', due to the definition of the set B.

4.5 Finding a marked node

The algorithm explained in Chapter 4 can also be used to find a marked vertex instead of checking its existence. To do so, the Backtracking algorithm is used within a binary search: Starting with the whole tree, in each step the algorithm checks, if there is a marked node in the tree. If there is, it continues with all subtrees, rooted at the children of the previous root, whereby only one of the marked ones is explored further. First it checks, if their root itself is marked. If it is, the label is returned and the algorithm stops. Otherwise the procedure continues. Assuming the degree of each vertex to be $\mathcal{O}(1)$, this routine will increase the time complexity of the Backtracking by a factor $\mathcal{O}(n)$. This increases the use of the operators to $\mathcal{O}(\sqrt{T}n^{3/2}\ln(n)\ln(1/\epsilon))$.

Given that there is a unique marked node in the tree, Montanaro shows that the time complexity of the algorithm can be reduced. The Backtracking algorithm then has time complexity of order $\mathcal{O}(\sqrt{Tn}\ln^3(n)\ln(1/\epsilon))$ ([10], pp.11-12).

5 Quantum Tree Size Estimation by Ambainis and Kokainis

In their paper from 2017 Andris Ambainis and Martins Kokainis [1], [2] further develop Montanaro's ideas into a quantum algorithm that estimates the size of a tree.

5.1 Setting and Requirements

The tree size estimation algorithm uses a tree, implicitly defined by an oracle. The only difference from the one used in Chapter 4 is that it does not matter anymore whether a node is marked or not. The node's labels are changed to integer numbers, first counting those nodes in the set $A = \{x : \ l(x)\ even\}\backslash\{r\}$ and then those in the set $B = \{x : \ l(x)\ odd\}$, where $l(x)$ is the distance of the node x to the root. In addition, this algorithm uses the operator $R_B R_A$ created from diffusion operators on the tree, similar to the one that Montanaro uses for the Backtracking (Chapter 4). It can be shown that performing a special form of the QPE directly provides an estimate for the tree size.

Due to the changed setting, the diffusion operators now do not take into account whether nodes are marked and are therefore defined slightly differently:

$$D_x = \begin{cases} \mathbb{1} - 2\left|\psi'_r\right\rangle\!\left\langle\psi'_r\right|, \text{ if } x \text{ is the root} \\ \mathbb{1} - 2\left|\psi'_x\right\rangle\!\left\langle\psi'_x\right|, \text{ else} \end{cases} \tag{42}$$

where

$$\begin{aligned} |\psi_x\rangle &= |x\rangle + \sum_{\{y:x\to y\}} |y\rangle \\ |\psi_r\rangle &= |r\rangle \ + \alpha \sum_{\{y:r\to y\}} |y\rangle\,. \end{aligned} \tag{43}$$

$|\psi'_x\rangle$ and $|\psi'_r\rangle$ in (42) are the normalized states of (43).

The operators $R_A = \bigoplus_{x\in A} D_x$ and $R_B = |r\rangle\langle r| + \bigoplus_{x\in B} D_x$ are defined as before.

Now let $\mathcal{H}_A$ be the span of all $|\psi_x\rangle$ with $x \in A \cup \{r\}$. Defining $\mathcal{H}_B$ analogously (for $x \in B$), the operators R_A and R_B act as the reflections $ref_{\mathcal{H}_A^\perp}$ and $ref_{\mathcal{H}_B^\perp}$:

$$R_A = \begin{cases} \mathbb{1} \text{ on } \mathcal{H}_A^\perp \\ -\mathbb{1} \text{ on } \mathcal{H}_A \end{cases} \quad \text{and} \quad R_B = \begin{cases} \mathbb{1} \text{ on } \mathcal{H}_B^\perp \\ -\mathbb{1} \text{ on } \mathcal{H}_B \end{cases} \tag{44}$$

5.2 Algorithm

Ambainis and Kokainis claim that their algorithm (Fig. 21) estimates the size T of a tree with a correctnesss propability of $1-\epsilon$ in the following manner: Given an upper bound T_0 for the tree size, it returns 'T contains more than T_0 nodes', if $(1+\delta)T \geq T_0$ and the estimate $\hat{T}$ satisfying $\left|\hat{T}-T\right| \leq \delta T$ otherwise. It uses $\mathcal{O}(\frac{\sqrt{nT_0}}{\delta^{3/2}} d \ln^2 \frac{1}{\epsilon})$ queries to the oracle and $\mathcal{O}(\ln T_0)$ other operations per query ([1], p.991).

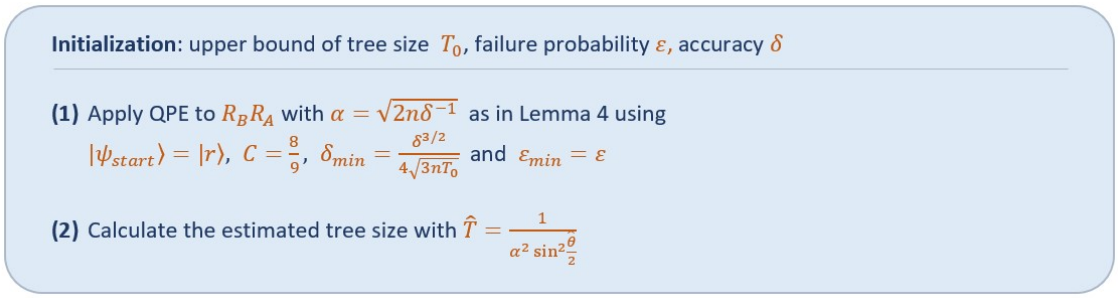

Fig. 21: Algorithm for tree size estimation ([1], p.992)

To prove the correctness of their algorithm, Ambainis and Kokainis needed the following lemmas and a few additional lemmas.

Lemma 4: ([1], p.991) *Let U be a unitary with eigenstates $|\psi_k\rangle$ such that $U|\psi_k\rangle = e^{i\theta_k}|\psi_k\rangle$. Given eigenstates $|\psi_+\rangle$ and $|\psi_-\rangle$ with eigenvalues $e^{\pm i\theta_{min}}$ closest to 1 and a state $|\psi_{start}\rangle$, that is orthogonal to all 1-eigenvectors of U and fullfills $|\langle\psi_+|\psi_{start}\rangle|^2 + |\langle\psi_-|\psi_{start}\rangle|^2 \geq C$ for some C, the following holds:*

The phase $\theta_{min} \neq 0$ closest to zero can be estimated with precision δ_{min} and success probability $Pr(|\theta_{min} - \hat{\theta}| \leq \delta_{min}) \geq 1 - \epsilon_{min}$ with $\mathcal{O}(\frac{1}{C\delta_{min}} \ln \frac{1}{C} \ln^2 \frac{1}{\epsilon_{min}})$ uses of controlled U.

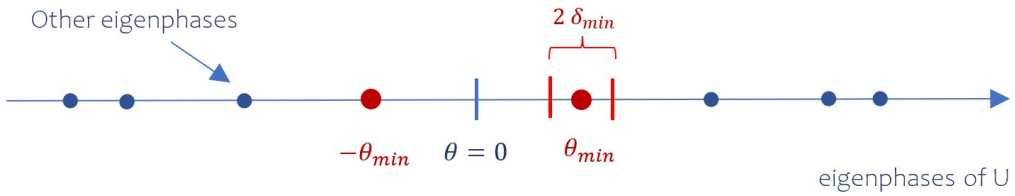

Fig. 22: Setting of Lemma 4

Due to the properties of R_A and R_B stated above, Lemma 5 (see Appendix) can be applied. This leads to the fact that all eigenvectors of $R_B R_A$ with eigenvalue 1 are either in $\mathcal{H}_A \cap \mathcal{H}_B$ or in $\mathcal{H}_A^\perp \cap \mathcal{H}_B^\perp$. Lemma 6 states that $dim(\mathcal{H}_A \cap \mathcal{H}_B) = 0$ and that all states in $\mathcal{H}_A^\perp \cap \mathcal{H}_B^\perp$ are orthogonal to the root $|r\rangle$. Finally, by Lemma 7, the root fulfills the condition $|\langle\psi_+|r\rangle|^2 + |\langle\psi_+|r\rangle|^2 \geq \frac{4}{9} + \frac{4}{9} = \frac{8}{9}$.

Thus, Lemma 4 can indeed be applied, using the root $|r\rangle$ as $|\psi_{start}\rangle$ and $C = \frac{8}{9}$.

The following lemma now verifies the connection between the QPE result and the tree size:

Lemma 8: ([1], p.992) *Let T be the tree size and let θ be the smallest eigenphase of $R_B R_A$. Let $\delta \in (0,1)$ and $\alpha = \sqrt{2n\delta^{-1}}$. Assume that the smallest eigenphase $\hat{\theta} \in (0, \pi/2)$ measured by QPE (Lemma 4) satisfies*

$$\left|\theta - \hat{\theta}\right| \leq \frac{\delta^{3/2}}{4\sqrt{3nT}}. \tag{45}$$

Then

$$(1-\delta)T \leq \frac{1}{\alpha^2 \sin^2(\frac{\hat{\theta}}{2})} \leq (1+\delta)T. \tag{46}$$

Last but not least, the time complexity's correctness can be proven by using Lemma 4, Lemma 9 and the variables from the algorithm:

According to Lemma 4, the algorithm uses

$$\mathcal{O}(\frac{1}{C\delta_{min}} \ln \frac{1}{C} \ln^2 \frac{1}{\epsilon_{min}}) = \mathcal{O}(\frac{4\sqrt{3nT_0}}{C\delta^{3/2}} \ln \frac{1}{C} \ln^2 \frac{1}{\epsilon}) = \mathcal{O}(\frac{\sqrt{nT_0}}{\delta^{3/2}} \ln^2 \frac{1}{\epsilon}) \tag{47}$$

controlled $R_B R_A$ operations which, by Lemma 9, can be implemented with $\mathcal{O}(d)$ calls of the oracle and $\mathcal{O}(d \ln T_0)$ other operations.

5.3 Implementing $\boldsymbol{R_A}$ and $\boldsymbol{R_B}$

Aiming to implement R_A, the first thing to notice is the fact that it is a direct sum of the diffusion operators D_x. Thus, the action of R_A can be completely determined by knowing all D_x.

Let $\mathcal{H}_1 = A \cup \{r\}$ and $\mathcal{H}_2 = B' = B \cup \{r_{-1}\}$, where r_{-1} is an additional node, connected only to the root r with label $\#B + 1$. Then due to the uniqueness of parents, x can also be expressed as follows:

$$|x\rangle \equiv \begin{cases} |x, x_p\rangle \in \mathcal{H}_1 \otimes \mathcal{H}_2, \text{ if } |x\rangle \in \mathcal{H}_1 \\ |x_p, x\rangle \in \mathcal{H}_1 \otimes \mathcal{H}_2, \text{ if } |x\rangle \in \mathcal{H}_2, \end{cases} \tag{48}$$

where x_p is the parent of the node x (see Fig. 23). With this new notation, each $|\psi_x\rangle$ can be expressed as $|x\rangle \otimes |\psi'_x\rangle$ for some $|\psi'_x\rangle$. Likewise, the diffusion operator can be expressed as a tensor product: $D_x = |x\rangle\langle x| \otimes D'_x$, with $D'_x = \mathbb{1} - \frac{2}{\||\psi'_x\rangle\|^2} |\psi'_x\rangle\langle\psi'_x|$ ([2], p.13).

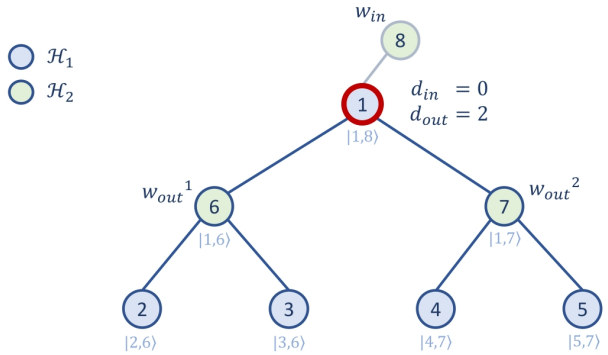

Fig. 23: Example tree for the implementation of D_x

For example, having a binary tree as shown in Fig. 23, for the root it is

$$\begin{aligned} |\psi_r\rangle = |\psi_1\rangle &= |1,8\rangle + \alpha \Big[|1,6\rangle + |1,7\rangle \Big] \\ &= |1\rangle \otimes \Big[|8\rangle + \alpha |6\rangle + \alpha |7\rangle \Big] \\ &= |1\rangle \otimes |\psi_1'\rangle \end{aligned} \tag{49}$$

and then

$$\begin{aligned} D_r = D_1 &= \mathbb{1} - \frac{2}{1+2\alpha^2} |1\rangle \otimes \Big[|8\rangle + \alpha |6\rangle + \alpha |7\rangle \Big] \langle 1| \otimes \Big[\langle 8| + \alpha \langle 6| + \alpha \langle 7| \Big] \\ &= |1\rangle\langle 1| \otimes \Big[|8\rangle\langle 8| + |6\rangle\langle 6| + |7\rangle\langle 7| \Big] \\ &\quad - \frac{2}{1+2\alpha^2} |1\rangle\langle 1| \otimes \Big[|8\rangle + \alpha |6\rangle + \alpha |7\rangle \Big] \Big[\langle 8| + \alpha \langle 6| + \alpha \langle 7| \Big] \\ &= |1\rangle\langle 1| \otimes D_1'. \end{aligned} \tag{50}$$

With this knowledge, R_A can be interpreted as an operator that for every $x \in \mathcal{H}_1$ acts on $\mathcal{H}_2$ as the corresponding D_x'. Finally, this operator can be implemented as follows:

1. At first, the 'parent' nodes w_i^{in} and the 'child' nodes w_i^{out} of x are determined and counted which is done by oracles. In the case of a tree with the added node r_{-1}, every node of the original tree has one incoming node. However, regarding the number of incoming nodes d_{in}, the additional node r_{-1} does not count. Thus, it is $d_{in} = 0$ for the root and $d_{in} = 1$ in any other case. The number of children d_{out} is between 0 and d (degree of graph) for every node. The nodes are saved in the registers $\mathcal{H}_j$ (further explained below).

2. Using a third register $\mathcal{H}_3$ with basis $\{|1\rangle, ..., |d+1\rangle\}$, where d is the degree of the tree, an operation is performed on $\mathcal{H}_2 \otimes \mathcal{H}_3$, mapping

$$\left|w^{in}\right\rangle|0\rangle, \left|w_1^{out}\right\rangle|0\rangle, ..., \left|w_{d_{out}}^{out}\right\rangle|0\rangle \;\mapsto\; |0\rangle|1\rangle, ..., |0\rangle|d_{in}+d_{out}\rangle, \tag{51}$$

whereby, if $x = r$, additionally $|r_{-1}\rangle|0\rangle$ gets mapped to $|0\rangle|d_{in}+d_{out}+1\rangle$.

3. One last operation is performed on $\mathcal{H}_3$: $\mathbb{1} - 2|\phi\rangle\langle\phi|$, where $|\phi\rangle \in \mathcal{H}_3$ is the normalized version of

$$|\phi'\rangle = \begin{cases} \sum_{i=1}^{d_{in}+d_{out}} |i\rangle & \text{, if } x \\ \alpha \sum_{i=1}^{d_{in}+d_{out}} |i\rangle + |d_{in}+d_{out}+1\rangle & \text{, if } x = r. \end{cases} \tag{52}$$

4. In the last step, the ancilla registers are set back by reversing steps 2 and 1

The implementation of the map given in step 2 uses registers $\mathcal{H}_j^{in}$ and $\mathcal{H}_{j'}^{out}$ storing the values w_j^{in} and $w_{j'}^{out}$ obtained in step 1. Thereby $j \in \{1\}$ and $j' \in \{1, ..., d_{out}+1\}$. For the incoming part $\mathcal{H}_j^{in}$, a unitary operator U_j acts on $\mathcal{H}_j^{in} \otimes \mathcal{H}_2 \otimes \mathcal{H}_3$, mapping $|w_j\rangle|w_j\rangle|0\rangle$ to $|w_j\rangle|0\rangle|j\rangle$. This map can be implemented in the following way.

2.1. On $\mathcal{H}_j \otimes \mathcal{H}_2$, perform the bitwise XOR operation $|x\rangle|y\rangle \;\mapsto\; |x \oplus y\rangle$.

2.2. If the second register is zero, add j to the third register.

2.3. If the third register is not j, reverse the first step (in this case its the same as repeating step 1).

For the registers $\mathcal{H}_{j'}^{out}$, similar unitary operators $U_{j+d_{in}}$ perform the map $|w_{j'}\rangle|w_{j'}\rangle|0\rangle \mapsto |w_{j'}\rangle|0\rangle|j'+1\rangle$. Additionally, the unitary operator $U_{d_{in}+d_{out}+1}$ is performed, mapping $|r_{-1}\rangle|0\rangle$ to $|0\rangle|d_{in}+d_{out}+1\rangle$ ([2], p.13).

As for time complexity, the first step of this implementation requires up to $d+2$ queries to the oracle (two for d_{in} and d_{out} and at most d to obtain all the values). Moreover, it can be shown that $\mathcal{O}(d \ln T)$ other operations are required. R_B can be implemented analogously ([1], p.992).

Note: The proofs and algorithms Ambainis and Kokainis present, originally use basis elements that correspond to the graph's edges (instead of its nodes) treating a more general type of graph. However, we consider here only the special case of trees (and no directed acyclic graphs in general), where by adding an additional edge to the root, edges and nodes become equivalent: A node can be assigned to the edge it shares with its parent. In order to combine all three discussed algorithms, they need to operate on the same basis. Therefore the algorithm of Ambainis and Kokainis is translated, such that it assigns nodes to basis elements (instead of edges). However, this might cause some unintuitive notations in this chapter.

5.3.1 Example

Continuing the above example, the aim is to determine the impact of D_r' on the node $|6\rangle$. First, step 1 outputs $d_{in} = 0$, $d_{out} = 2$ and the connected nodes $w^{in} = |r_{-1}\rangle$,

$w_1^{out} = |6\rangle$ and $w_2^{out} = |7\rangle$ which are stored in registers $\mathcal{H}_j^{in}$ and $\mathcal{H}_{j'}^{out}$. Thus, after step 1 the registers look as follows.

$$\begin{aligned} &|d_{in}\rangle\ |d_{out}\rangle\ |r\rangle_{\mathcal{H}_1}\ |w\rangle_{\mathcal{H}_2}\ |0\rangle_{\mathcal{H}_3}\ \left|w^{in}\right\rangle_{\mathcal{H}_1^{in}}\left|w_1^{out}\right\rangle_{\mathcal{H}_1^{out}}\ \ldots \\ =&|0\rangle\quad |2\rangle\quad |1\rangle_{\mathcal{H}_1}\ |6\rangle_{\mathcal{H}_2}\ |0\rangle_{\mathcal{H}_3}\ |8\rangle_{\mathcal{H}_1^{in}}\quad |6\rangle_{\mathcal{H}_1^{out}}\ |7\rangle_{\mathcal{H}_1^{out}} \end{aligned} \tag{53}$$

Now the first map (51) is performed on $\mathcal{H}_2 \otimes \mathcal{H}_3$ using the procedure explained above: The register $\mathcal{H}_j^{in}$ only contains one state $|8\rangle = |1000\rangle$, where the right hand side is the binary representation. Thus, the three steps (2.1 to 2.3) are only done for $j = 1$.

$$\begin{aligned} &1.\ |8\rangle_{\mathcal{H}_{j=1}^{in}}|6\rangle_{\mathcal{H}_2}|0\rangle_{\mathcal{H}_3} = |1000\rangle|0110\rangle|0\rangle \longmapsto |1000\rangle|1110\rangle|0\rangle \\ j=1\qquad &2.\ |0\rangle_{\mathcal{H}_3} \longmapsto |0\rangle \neq j \\ &3.\ |1000\rangle_{\mathcal{H}_{j=1}^{in}}|1110\rangle_{\mathcal{H}_2}|0\rangle_{\mathcal{H}_3} \longmapsto |1000\rangle|0110\rangle = |8\rangle|6\rangle \end{aligned} \tag{54}$$

Now proceed analogously with $\mathcal{H}_j^{out} \otimes \mathcal{H}_2 \otimes \mathcal{H}_3$:

$$\begin{aligned} j'=1\quad |6\rangle_{\mathcal{H}_{j'=1}^{out}}|6\rangle_{\mathcal{H}_2}|0\rangle_{\mathcal{H}_3} &= |0110\rangle|0110\rangle|0\rangle \\ &\mapsto |0110\rangle|0000\rangle|1\rangle = |6\rangle_{\mathcal{H}_1^{out}}|0\rangle_{\mathcal{H}_2}|1\rangle_{\mathcal{H}_3} \\ j'=2\quad |7\rangle_{\mathcal{H}_{j'=2}^{out}}|0\rangle_{\mathcal{H}_2}|1\rangle_{\mathcal{H}_3} &= |0111\rangle|0000\rangle|1\rangle \\ &\mapsto |0111\rangle|0111\rangle|1\rangle \\ &\mapsto |0111\rangle|0000\rangle|1\rangle = |7\rangle_{\mathcal{H}_2^{out}}|0\rangle_{\mathcal{H}_2}|1\rangle_{\mathcal{H}_3} \end{aligned} \tag{55}$$

Completing step 2 yields the state

$$|0\rangle\ |2\rangle\ |r\rangle_{\mathcal{H}_1}\ |0\rangle_{\mathcal{H}_2}\ |1\rangle_{\mathcal{H}_3}\ |8\rangle_{\mathcal{H}_1^{in}}\ |6\rangle_{\mathcal{H}_1^{out}}\ |7\rangle_{\mathcal{H}_1^{out}} \tag{56}$$

The final operation in step 3 is acting with the operator $\mathbb{1} - 2|\phi\rangle\langle\phi|$ on $\mathcal{H}_3$, where

$$\begin{aligned} |\phi\rangle &= \frac{1}{\sqrt{\alpha^2(d_{in}+d_{out})+1}}\left[\alpha\sum_{i=1}^{d_{in}+d_{out}}|i\rangle + |d_{in}+d_{out}+1\rangle\right] \\ &= \frac{1}{\sqrt{2\alpha^2+1}}\Big[\alpha|1\rangle + \alpha|2\rangle + |3\rangle\Big]. \end{aligned} \tag{57}$$

Applying this operator gives

$$|0\rangle\ |2\rangle\ |r\rangle_{\mathcal{H}_1}\ |0\rangle_{\mathcal{H}_2}\ \left[|1\rangle - \frac{2\alpha}{2\alpha^2+1}\Big[\alpha|1\rangle_{\mathcal{H}_3} + \alpha|2\rangle_{\mathcal{H}_3} + |3\rangle_{\mathcal{H}_3}\Big]\right]|8\rangle_{\mathcal{H}_1^{in}}\ |6\rangle_{\mathcal{H}_1^{out}}\ |7\rangle_{\mathcal{H}_1^{out}}. \tag{58}$$

Last but not least, reversing steps 2 and 1 leads to the final result

$$|0\rangle\ |0\rangle\ |r\rangle_{\mathcal{H}_1}\ \left[|6\rangle_{\mathcal{H}_2} - \frac{2\alpha}{2\alpha^2+1}\left[\alpha\,|6\rangle_{\mathcal{H}_2} + \alpha\,|7\rangle_{\mathcal{H}_2} + |8\rangle_{\mathcal{H}_2}\right]\right]\ |0\rangle_{\mathcal{H}_3}\,|0\rangle_{\mathcal{H}_1^{in}}\ |0\rangle_{\mathcal{H}_1^{out}}\ \ldots \tag{59}$$

With the notation from above, this result corresponds to a superposition of the nodes:

$$|r,6\rangle - \frac{2\alpha}{2\alpha^2+1}\big[|r,8\rangle + \alpha\,|r,6\rangle + \alpha\,|r,7\rangle\big] \mathrel{\hat{=}} |6\rangle - \frac{2\alpha}{2\alpha^2+1}\big[|1\rangle + \alpha\,|6\rangle + \alpha\,|7\rangle\big] \tag{60}$$

To check this result, compare it with the one obtained from the original definition (42). Having $D_r = D_1 = \mathbb{1} - \frac{2}{\||\psi_1\rangle\|^2}\,|\psi_1\rangle\langle\psi_1|$ with $|\psi_1\rangle = |1\rangle + \alpha\big[\,|6\rangle + |7\rangle\big]$, it is

$$D_1\,|6\rangle = |6\rangle - \frac{2\alpha}{2\alpha^2+1}\left[|1\rangle + \alpha\,|6\rangle + \alpha\,|7\rangle\right] \tag{61}$$

which indeed is the same.

6 Summary and outlook

The overall concept of Montanaro's quantum Branch-and-Bound speedup has been fully presented in this work. Starting from the overarching concept of the quantum Branch-and-Bound algorithm in Chapter 3, the general mode of operation was explained, solving a Knapsack problem using both the classical and quantum versions of the Branch-and-Bound algorithm as an example. Furthermore, in Chapters 4 and 5 the applied subroutines have been explained in detail, including most of the proofs. Several examples were used to explain the different parts of the algorithms in small steps to clarify the theoretical ideas and the notations used.

The detailed explanations in conjunction with the accompanying examples in this work should be sufficient as a basis to successfully implement simulations of Montanaro's algorithm as a whole. In fact, the implementation is already underway within the group of Tobias Osborne at Leibniz University Hannover. Further work could include, for example, reducing the number of qubits needed for the implementation. One possibility would be to revisit the original Tree Size Estimation algorithm, which uses edges instead of nodes, and investigate whether there are other ways to apply it to trees that require less qubits.

The availability of quantum computers still remains a big obstacle. Within the next decades, however, large problem instances might be solvable running this algorithm on real quantum devices.

7 Appendix

Lemma 1. *Spectral gap Lemma ([10], p.8 and [7], pp.347-348)*

Let Π_A and Π_B both be projectors in the Hilbert space $\mathcal{H}$. Consider the operators $R_A = 2\Pi_A - \mathbb{1}$ and $R_B = 2\Pi_A - \mathbb{1}$. Choosing $\chi > 0$, let P_χ be the projector onto the subspace of $R_B R_A$'s eigenvectors with eigenvalue $e^{2i\theta}$, where $|\theta| \leq \chi$

$$\|P_\chi \Pi_B |\psi\rangle\| \leq \chi \||\psi\rangle\| \tag{62}$$

for any vector $|\psi\rangle$ with $\Pi_A |\psi\rangle = 0$.

Lemma 2. *Quantum Phase Estimation ([10], pp.8-9)*

Given the integer $s \geq 1$ and a unitary U on m qubits, there exists a quantum circuit C on $m+s$ qubits such that

1. C contains $\mathcal{O}(2^s)$ controlled-U operators and $\mathcal{O}(2^s)$ other gates.

2. $C |\psi_k\rangle |0^s\rangle = |\psi_k\rangle |0^s\rangle$ holds for all eigenvectors $|\psi_k\rangle$ of U with eigenvalue 1

3. $C |\psi_k\rangle |0^s\rangle = |\psi_k\rangle |\omega_k\rangle$ with $|\langle\omega_k|0^s\rangle|^2 = \sin^2(2^s\theta_k)/(2^{2s}\sin^2\theta)$ holds for any eigenvector $|\psi_k\rangle$ with eigenvalue $e^{2i\theta_k}$, $\theta_k \in (0, \pi)$.

4. $C |\phi\rangle |0^s\rangle = \sum_k \lambda_k |\psi_k\rangle |\omega_k\rangle$ holds for arbitrary states expanded as $|\phi\rangle = \sum_k \lambda_k |\psi_k\rangle$

Call 2^{-s} the precision of the circuit.

To distinguish the eigenvalue 1 from others, given that the smallest nonzero phase is ϵ, $\mathcal{O}(1/\epsilon)$ controlled-U operations are required.

Lemma 3. *([10], pp.10-11)*

Let $R_B R_A$ be defined as in Chapter 4. Then there exists a universal constant $\beta \leq \frac{1}{4\pi}$ of order $\mathcal{O}(1)$, such that applying Phase Estimation with precision $\beta/\sqrt{Tn}$, will give eigenvalue $\eta = 1$ with the following probabilities:

$$\begin{aligned} &Pr(\eta = 1) \geq 1/2, \text{ if there exists a marked vertex} \\ &Pr(\eta = 1) \leq 1/4, \text{ if there exists no marked vertex} \end{aligned} \tag{63}$$

Proof : ([10], pp.10-11) Let x_0 be a marked node in the tree and let $x : x \rightsquigarrow x_0$ denote the vertices on the direct way from the root r to x_0 (including x_0, excluding r). Define the state

$$|\phi\rangle = \sqrt{n}\,|r\rangle + \sum_{x : x \rightsquigarrow x_0} (-1)^{l(x)} |x\rangle . \tag{64}$$

This state is orthogonal to all ψ_x and to ψ_r defined in (28):

Because for every node x, $|\psi_x\rangle$ contains states corresponding to the node itself and to all its children, $|\psi_x\rangle$ and $|\phi\rangle$ include either zero or two (consecutive) common

nodes, if x is not marked. Thus, their scalar product will give $\langle\psi_x|\phi\rangle = 0$. For $|\psi_r\rangle$ orthogonality can also be calculated easily:

$$\begin{aligned}\langle\psi_r|\phi\rangle =& \left[\frac{1}{\sqrt{1+nd_r}}\left[\langle r| + \sqrt{n}\sum_{\{y:r\to y\}}\langle y|\right]\right]\left[\sqrt{n}\,|r\rangle + \sum_{x:x\rightsquigarrow x_0}(-1)^{l(x)}\,|x\rangle\right] \\ =& \frac{1}{\sqrt{1+nd_r}}(\sqrt{n}-\sqrt{n}) = 0\end{aligned} \tag{65}$$

As a consequence of this orthogonality and the fact that $R_B R_A$ is constructed out of diffusion operators projecting on these exact states (27), $|\phi\rangle$ is an eigenvector of $R_B R_A$ with eigenvalue 1.

Furthermore $|\phi\rangle$ fulfills $\||\phi\rangle\|^2 = n + l(x_0) \leq 2n$.

Thus it follows that

$$\frac{\langle r|\phi\rangle}{\||\phi\rangle\|} \geq \frac{\sqrt{n}}{2\sqrt{n}} = \frac{1}{2}. \tag{66}$$

Therefore, applying Phase Estimation to the state $|r\rangle$, the returned eigenvalue will be 1 with a probability of $Pr(1) \geq 1/2$, which proves the first statement of the Lemma.

For the case of no marked vertex, consider the state

$$|\eta\rangle = |r\rangle + \sqrt{n}\sum_{x\neq r}|x\rangle\,. \tag{67}$$

Let the states $\left|\psi^{\perp}_{x\in A}\right\rangle$ be orthogonal to the corresponding $|\psi_x\rangle$ from definition (28) and let them be supported on the same set $\mathcal{H}_x = \{|x\rangle\} \cup \{|y\rangle : x \to y\}$. Then the subspace spanned by these states is left invariant under R_A. For R_B it works analogously (only the support is extended by $\{|r\rangle\}$). Let now Π_A and Π_B be the projectors onto these invariant subspaces of R_A and R_B. Then

$$\Pi_A\,|\eta\rangle = 0 \text{ and } \Pi_B\,|\eta\rangle = |r\rangle\,, \tag{68}$$

because on each subspace $\mathcal{H}_x$ the state $|\eta\rangle$ is proportional to $|\psi_x\rangle$. Let now P_χ be the projector onto the eigenvectors of $R_B R_A$ with eigenvalues $e^{2i\theta}$ satisfying $|\theta| \leq \chi$. Then

$$\|P_\chi\,|r\rangle\| = \left\|\sum_{k:|\theta_k|\leq\chi}|\psi_k\rangle\langle\psi_k|\sum_{k'=1}^{T}\lambda_{k'}\,|\psi_{k'}\rangle\right\| = \left\|\sum_{k:|\theta_k|\leq\chi}\lambda_k\,|\psi_k\rangle\right\| = \sqrt{\sum_{k:|\theta_k|\leq\chi}|\lambda_k|^2}. \tag{69}$$

In the case where there is no marked vertex, it is easy to see that $R_A = 2\Pi_A - \mathbb{1}$ and $R_B = 2\Pi_B - \mathbb{1}$. This allows applying the spectral gap Lemma (Lemma 1). Thus, choosing a $\chi \leq \frac{1}{\sqrt{8Tn}}$, the following inequation holds:

$$\|P_\chi\,|r\rangle\| = \|P_\chi\Pi_B\,|\eta\rangle\| \leq \chi\||\eta\rangle\| \leq \chi\sqrt{Tn} \leq 1/\sqrt{8}. \tag{70}$$

Combined with (69) this leads to

$$\sum_{k:|\theta_k|\leq\chi} |\lambda_k|^2 \leq \frac{1}{8}. \tag{71}$$

The final step is to look at the probability for measuring eigenvalue 1 (i.e $\theta_k = 0$). Therefore, the required probability is the one of measuring $|0^s\rangle$ in the ancilla register. In the notation from the Phase Estimation (Lemma 2) this probability is given by

$$\Pr(\text{ev} = 1) = \sum_k |\lambda_k|^2 |\langle\omega_k|0^s\rangle|^2. \tag{72}$$

Combining Lemma 3 and the inequalities $\sin x \leq x$ and $\sin x \geq 2x/\pi$ it is

$$|\langle\omega_k|0^s\rangle|^2 = \frac{\sin^2(2^s\theta_k)}{2^{2s}\sin^2\theta} \leq \frac{1}{2^{2s}\sin^2\theta_k} \leq \frac{\pi^2}{2^{2s}4\theta_k^2}. \tag{73}$$

Thus, using (71) and (73) the required inequaltity can now be derived:

$$\begin{aligned}
\Pr(\text{ev} = 1) &= \sum_k |\lambda_k|^2 |\langle\omega_k|0^s\rangle|^2 \\
&= \sum_{k:|\theta_k|\leq\chi} |\lambda_k|^2 |\langle\omega_k|0^s\rangle|^2 + \sum_{k:|\theta_k|>\chi} |\lambda_k|^2 |\langle\omega_k|0^s\rangle|^2 \\
&\leq \sum_{k:|\theta_k|\leq\chi} |\lambda_k|^2 + \sum_{k:|\theta_k|>\chi} |\lambda_k|^2 \frac{\pi^2}{2^{2s}4\theta_k^2} \\
&\leq \frac{1}{8} + \frac{\pi^2}{2^{2s}4\chi^2} \overset{!}{\leq} \frac{1}{4}
\end{aligned} \tag{74}$$

Finally, by using $\chi \leq \frac{1}{\sqrt{8Tn}}$, this is equivalent to the condition

$$\beta = 2^{-s} \leq \sqrt{\frac{4}{8\pi^2}\chi^2} \leq \sqrt{\frac{1}{16\pi^2 Tn}} \leq \frac{1}{4\pi\sqrt{Tn}} \tag{75}$$

In the case of no existing marked vertex, it is therefore proven that for any $\beta \leq 1/(4\pi)$, applying Phase Estimation to $R_B R_A$ with precision $\beta/\sqrt{Tn}$ will return the eigenvalue 1 with a probability of at most 1/4. □

Note that even though this estimation for β is universal, it is likely that one can find higher bounds for it.

Lemma 4. *([1], p.991) Let U be a unitary operator with eigenstates $|\psi_k\rangle$ such that $U|\psi_k\rangle = e^{i\theta_k}|\psi_k\rangle$. Given eigenstates $|\psi_+\rangle$ and $|\psi_-\rangle$ with eigenvalues $e^{\pm i\theta_{min}}$ closest to 1 and a state $|\psi_{start}\rangle$, that is orthogonal to all 1-eigenvectors of U and fullfills $|\langle\psi_+|\psi_{start}\rangle|^2 + |\langle\psi_-|\psi_{start}\rangle|^2 \geq C$ for some C, the following holds:*

The phase $\theta_{min} \neq 0$ closest to zero can be estimated with precision δ_{min} and success probability $Pr(|\theta_{min} - \hat{\theta}| \leq \delta_{min}) \geq 1 - \epsilon_{min}$ with $\mathcal{O}(\frac{1}{C\delta_{min}} \ln \frac{1}{C} \ln^2 \frac{1}{\epsilon_{min}})$ uses of controlled U.

Proof ([2], p.12): Let $t = \frac{1}{C} \ln \frac{2}{\epsilon_{min}}$ be the number of times, QPE is applied to the state $|\psi_{start}\rangle$ with precision $\delta_{est} = \delta_{min}$ and failure probability $\epsilon_{est} = \epsilon_{min}/(2t)$. Then the probability for only estimating phases other than θ_{min} is

$$\begin{aligned} Pr(\text{not } \theta_{min}) &= [1 - (|\langle\psi_+|\psi_{start}\rangle|^2 + |\langle\psi_-|\psi_{start}\rangle|^2)]^t \\ &\leq (1 - C)^t \leq e^{-Ct} \\ &= e^{-\ln \frac{2}{\epsilon_{min}}} = \epsilon_{min}/2. \end{aligned} \tag{76}$$

Furthermore, the probability for measuring any phase, $\hat{\theta}$ with $|\hat{\theta} - \theta| \geq \delta_{min}$ is at most $t\epsilon_{est} = \epsilon_{min}/2$. Thus, the total probability for estimating the right phase with the right precision is $Pr(|\theta_{min} - \hat{\theta}| \leq \delta_{min}) \geq 1 - \epsilon_{min}$.

Given the required number of controlled U operations for one QPE, this algorithm does $\mathcal{O}(t\frac{1}{\delta_{est}} \ln \frac{1}{\epsilon_{est}}) = \mathcal{O}(\frac{1}{C} \ln\left(\frac{1}{\epsilon_{min}}\right) \frac{1}{\delta_{est}} \ln \frac{1}{\epsilon_{est}}) \leq \mathcal{O}(\frac{1}{C} \ln\left(\frac{1}{\epsilon_{min}}\right) \frac{1}{\delta_{est}} \ln \frac{1}{C} \ln \frac{1}{\epsilon_{min}})$ such operations. □

Lemma 5. *(Spectral Lemma) ([1], p.995) [14]*

Let $\mathcal{A}$, $\mathcal{B}$ be subspaces of a Hilbert space $\mathcal{H}$, spanned by the orthonormal bases $\{\omega_1, ...\omega_k\}$ and $\{\tilde{\omega}_1, ...\tilde{\omega}_k\}$. Let $ref_{\mathcal{A}}$ denote the reflection, acting as the identity on $\mathcal{A}$ and negating all the vectors in $\mathcal{A}^\perp$ (analogously for $ref_{\mathcal{B}}$). Let M be a matrix defined by $M = Gram(\omega_1, ...\omega_k\, \tilde{\omega}_1, ...\tilde{\omega}_k) - \mathbb{1}$, where the Gram Matrix is defined by $Gram(v_1, ..., v_n)^{ij} = \langle v_i, v_j\rangle$. Then the eigenvalues of $ref_{\mathcal{B}} ref_{\mathcal{A}}$ are as follows:

1. eigenvalue 1 for vectors in $\mathcal{A} \cap \mathcal{B}$

2. eigenvalue $2\lambda^2 - 1 \mp 2i\lambda\sqrt{1-\lambda^2}$ for eigenvectors of the form $(|\tilde{a}\rangle - \lambda \left|\tilde{b}\right\rangle) \pm i\sqrt{1-\lambda^2} \left|\tilde{b}\right\rangle$ where a,b eigenvectors of M with eigenvalue $\lambda \in (0, 1)$ and

$$|\tilde{a}\rangle = \sum_{i=1}^{k} a_i\omega_i, \ \left|\tilde{b}\right\rangle = \sum_{i=1}^{l} b_i\omega_i. \tag{77}$$

3. eigenvalue -1 for vectors $|\tilde{a}\rangle$ and $\left|\tilde{b}\right\rangle$ where a and b have eigenvalue 0 (as eigenvectors of M)

Lemma 6. *([1], p.995) Let $\mathcal{H}_A$ and $\mathcal{H}_B$ be defined as in chapter 5. Then the root $|r\rangle$ is orthogonal to each state in $\mathcal{H}_A^\perp \cap \mathcal{H}_B^\perp$ and $dim(\mathcal{H}_A \cap \mathcal{H}_B) = 0$*

Proof : ([2], p.15) To prove the first statement, it needs to be shown that $|r\rangle \in (\mathcal{H}_A^{\perp} \cap \mathcal{H}_B^{\perp})^{\perp} = \mathcal{H}_A + \mathcal{H}_B$. It is

$$\sum_{x\in A}\Big[\sum_{\{y:x\to y\}}|y\rangle\Big] + |x\rangle = \sum_{x\in B}\Big[\sum_{\{y:x\to y\}}|y\rangle\Big] + |x\rangle\,, \tag{78}$$

because for every node x it gives the node itself and its children, while both A and B contain every second layer of the tree.

With this it follows that

$$\begin{aligned} |r\rangle = |r\rangle \;\; &+\alpha\sum_{x\in A}\Big[\sum_{\{y:x\to y\}}|y\rangle\Big] + |x\rangle - \alpha\sum_{x\in B}\Big[\sum_{\{y:x\to y\}}|y\rangle\Big] + |x\rangle \\ = |\psi_r\rangle &+\alpha\sum_{x\in A\setminus\{r\}}|\psi_x\rangle \qquad\qquad - \alpha\sum_{x\in B}|\psi_x\rangle\,. \end{aligned} \tag{79}$$

This shows that $|r\rangle \in \mathcal{H}_A + \mathcal{H}_B$.

To prove that $dim(\mathcal{H}_A \cap \mathcal{H}_B) = 0$, we use a contradiction: Suppose there is a state $|x\rangle$ with $|x\rangle \in \mathcal{H}_A$ and $|x\rangle \in \mathcal{H}_B$. Then it can be expressed in two ways:

$$\begin{aligned} |x\rangle &= \sum_{y\in A}\lambda_y \;\Big[|y\rangle + \sum_{\{y':y\to y'\}}|y'\rangle\Big] + \lambda_r\Big[|r\rangle + \sum_{\{y':r\to y'\}}|y'\rangle\Big] \\ &= \sum_{y\in B}\lambda_y \;\Big[|y\rangle + \sum_{\{y':y\to y'\}}|y'\rangle\Big] \end{aligned} \tag{80}$$

From this it is clear that $\lambda_r = 0$. Furthermore, due to the graph being connected, all coefficients have to be the same. Thus, only the null vector is in $\mathcal{H}_A \cap \mathcal{H}_B$. □

Lemma 7. *([1], p.996) Let n be the depth of a tree, $|r\rangle$ be its root, and $|\psi_{\pm}\rangle$ defined as in Lemma 4. Then*

$$\alpha \le \sqrt{2n} \Rightarrow \langle r|x\rangle \ge \frac{2}{3} \tag{81}$$

for states $|x\rangle \in span(\{|\psi_+\rangle\,, |\psi_-\rangle\})$.

Lemma 8. *([1], p.992) Let T be the tree size and let θ be the smallest eigenphase of $R_B R_A$. Let $\delta \in (0,1)$ and $\alpha = \sqrt{2n\delta^{-1}}$. Assume that the smallest eigenphase $\hat{\theta} \in (0, pi/2)$ measured by QPE (Lemma 4) satisfies*

$$\left|\theta - \hat{\theta}\right| \le \frac{\delta^{3/2}}{4\sqrt{3nT}}. \tag{82}$$

Then

$$(1-\delta)T \le \frac{1}{\alpha^2 \sin^2(\frac{\hat{\theta}}{2})} \le (1+\delta)T. \tag{83}$$

Lemma 9. *([1], p.992) The operators R_B and R_A defined in Chapter 5 can be implemented using $\mathcal{O}(d)$ queries to the oracle and $\mathcal{O}(d \ln T_0)$ other operations.*

References

[1] Andris Ambainis and Martins Kokainis. "Quantum Algorithm for Tree Size Estimation, with Applications to Backtracking and 2-Player Games". In: STOC 2017. Montreal, Canada: Association for Computing Machinery, 2017, pp. 989–1002. ISBN: 9781450345286. DOI: 10.1145/3055399.3055444. URL: https://doi.org/10.1145/3055399.3055444.

[2] Andris Ambainis and Martins Kokainis. *Quantum algorithm for tree size estimation, with applications to backtracking and 2-player games*. 2017. DOI: 10.48550/ARXIV.1704.06774. URL: https://arxiv.org/abs/1704.06774.

[3] Aleksandrs Belovs et al. "Time-Efficient Quantum Walks for 3-Distinctness". In: *Automata, Languages, and Programming*. Ed. by Fedor V. Fomin et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 105–122. ISBN: 978-3-642-39206-1.

[4] H. A. Eiselt and C.-L. Sandblom. "Branch and Bound Methods". In: *Integer Programming and Network Models*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 205–228. ISBN: 978-3-662-04197-0. DOI: 10.1007/978-3-662-04197-0_10. URL: https://doi.org/10.1007/978-3-662-04197-0_10.

[5] Michael Jarret and Kianna Wan. "Improved quantum backtracking algorithms using effective resistance estimates". In: *Phys. Rev. A* 97 (2 Feb. 2018), p. 022337. DOI: 10.1103/PhysRevA.97.022337. URL: https://link.aps.org/doi/10.1103/PhysRevA.97.022337.

[6] Josef Kallrath. *Gemischt-ganzzahlige Optimierung: Modellierung in der Praxis : mit Fallstudien aus Chemie, Energiewirtschaft, Papierindustrie, Metallgewerbe, Produktion und Logistik*. Wiesbaden: Springer Spektrum, 2013. ISBN: 9783658006891 3658006897. URL: http://www.worldcat.org/search?qt=worldcat_org_all&q=9783658006891.

[7] Troy Lee et al. "Quantum Query Complexity of State Conversion". In: Nov. 2011, pp. 344–353. DOI: 10.1109/FOCS.2011.75.

[8] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

[9] Ashley Montanaro. "Quantum speedup of branch-and-bound algorithms". In: *Phys. Rev. Research* 2 (1 Jan. 2020), p. 013056. DOI: 10.1103/PhysRevResearch.2.013056. URL: https://link.aps.org/doi/10.1103/PhysRevResearch.2.013056.

[10] Ashley Montanaro. "Quantum walk speedup of backtracking algorithms". In: *Theory of Computing* 14 (Sept. 2016). DOI: 10.4086/toc.2018.v014a015.

[11] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. DOI: 10.1017/CBO9780511976667.

[12] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[13] Peter W. Shor. “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer”. In: *SIAM J. Comput.* 26.5 (Oct. 1997), pp. 1484–1509. ISSN: 0097-5397. DOI: 10.1137/S0097539795293172. URL: https://doi.org/10.1137/S0097539795293172.

[14] M. Szegedy. “Quantum speed-up of Markov chain based algorithms”. In: *45th Annual IEEE Symposium on Foundations of Computer Science*. 2004, pp. 32–41. DOI: 10.1109/FOCS.2004.53.