Institut für Theoretische Physik

Bachelor Thesis

Enhancements for the Quantum Tree Generator: Application to Multidimensional Knapsack Problems

Robert Karimov

10038547

under supervision of

Prof. Dr. Tobias J. Osborne

19. June 2024

Uh, you know my body quality over quantity (Uh)
He couldn't possibly made another quantum leap

ROC MARCIANO

Abstract

The Quantum Tree Generator (QTG), related to the search method pioneered by Grover, is a novel development in the realm of quantum algorithms which offers an approach to solving certain combinatorial optimisation problems. This thesis is concerned with the adaptation of the QTG framework to the multidimensional 0-1 knapsack problem as well as its simulation and numerical evaluation following the methods of the original QTG proposal to show quantum advantage over a classical solver.

Contents

1 Introduction

Quantum computation has become one of the most promising technological innovations of the 21st century, garnering widespread media attention as well as commercial interest[1][2]. Ever since its independent conception by Yuri Manin[3] and Richard Feynman[4] in the early 80s and especially after the discovery of its impact on encryption by Peter Shor in 1994[5], huge efforts have been put into the development of architectures which support quantum computation, centered around the efficient and scalable realisation of the qubit, the quantum unit of information. These efforts have spawned a wide variety of quantum computer designs, making use of superconducting circuits[6], trapped ions[7], quantum dots[8], and many other effects to implement qubits along with appropriate methods to alter their states. Over the past 20 years, the number of physical qubits in quantum processors has experienced a steady growth, and current day market leaders D-Wave[9] and IBM[10] are able to offer QPU solutions supporting over 5000 and 1000 qubits, respectively. This development is expected to continue, making a 10000-qubit quantum computer possible within the next decade[11][12].

The reason for this accelerated progress lies in the fact that quantum computers have been show to solve certain problems in significantly less time compared to their classical counterparts, first theoretically[13] and recently also in practice[14], making solutions obtainable within realistic instead of astronomical time spans. At this point in time, however, this comes with a slight caveat: The quantum advantage demonstrated thus far holds only for specifically tailored problems, as the still limited size of the systems restricts any real application. Nevertheless, areas for which a quantum advantage is projected in the future include the simulation of quantum systems as is of interest for material and chemical sciences, linear algebra and quantum machine learning, as well as optimisation[15]. The problem tackled in this thesis falls under this last category.

Optimisation problems, which require maximising or minimising one value while fulfilling a set of constraints, arise naturally in many fields such as software engineering, artificial intelligence and applied mathematics, with applications in economics, logistics and energy systems. When the space of feasible solutions is finite, the corresponding optimisation problem is said to be combinatorial and the task can be reframed as a search for the optimum in the solution space. Perhaps one of the best-known combinatorial problems is the 0-1 knapsack problem, which, in its basic form, asks for the optimal profit obtainable by a set of valuable items, each consuming some amount of a finite resource and being includable only once. From the point of complexity theory, this problem is NP-complete, admitting no known efficient (polynomial-time) algorithm for its solution. Approaches to solving the 0-1 knapsack problem exactly in a classical setting often involve dynamic programming or branch-and-bound methods, and there are also heuristics for obtaining approximate solutions.

On the side of quantum algorithms there exist multiple approaches, both exact and approximate, to solving combinatorial optimisation problems. These methods include the quantum adiabatic optimisation algorithm[16] as well as the quantum approximate optimisation algorithm[17], where the search for the (approximately) optimal solution is tied to finding the ground state of a certain Hamiltonian. An ansatz for optimisation in certain combinatorial settings, coined Quantum Tree Generator (QTG), has been recently proposed in a preprint by Wilkening et. al.[18], where it is applied to the 0-1 knapsack problem. Their scheme iteratively makes use of a generalised version of Grover's algorithm[19], which provides a quadratic speedup for the lookup of an element in an unstructured set over a classical search, implemented on the space of feasible solutions to

the problem (represented by qubit states) by filtering for states which allow for a higher profit than a set lower bound threshold. Coupled to the QTG algorithm, the authors also provide a method to estimate the resources needed for its execution on realistic benchmark instances by means of a high-level simulator. This is relevant as, again, current quantum platforms are not capable of solving large-scale problems, leaving the proposed simulation as the only way to get a grasp on the resulting resource requirements. Ultimately, the preprint provides evidence for advantage of the QTG over COMBO[20], the current champion algorithm for the 0-1 knapsack problem, starting at input sizes of approximately $N = 600$.

In this thesis we will follow closely the original QTG proposal in the development of a quantum algorithm for a variation of the 0-1 knapsack problem: The multidimensional 0-1 knapsack problem. First, its definition and classical methods for solving it are briefly presented, followed by a rundown of the needed quantum computational background as well as a detailed description of the algorithm and the resource estimate. Subsequently, the corresponding simulator is introduced and implemented to obtain resource data of the QTG for several benchmark problem classes, the results are then compared to those of the CPLEX optimisation software[21], an industry standard solver for a variety of optimisation problems. We close with a discussion of the findings and give an outlook on possible future research.

2 Theoretical Groundwork

In this Section, we will begin with a brief presentation of the multidimensional 0-1 knapsack problem, treating its definition as well as classical approaches to its solution. We follow this up with an overview of the quantum computational theory needed for understanding the QTG, introducing the relevant notation and working up to Grover's algorithm in order to consider its generalisation manifested in QTG.

2.1 The Multidimensional 0-1 Knapsack Problem

The multidimensional 0-1 knapsack problem (0-1 MDKP) can be considered to be a generalisation of the 0-1 knapsack problem described in the introduction, in that each item requires a certain amount from multiple resources. As a simple example of application, picture the following:
In a small town, a busy carpenter is known for their custom pieces out of oak, of which they have a set supply for the remaining month, after which it is restocked. Their work is highly regarded by the townsfolk, which is why they have more orders than they could possibly get done until their upcoming vacation in two weeks. The carpenter now has the task of selecting those orders which a) give the most profit while b) being possible with the material they have and c) taking no longer than two weeks to complete all together, as to not delay their vacation (once again). This presents a two dimensional 0-1 knapsack problem, as the obtainable profit from the items (orders) has to comply with two constraints (wood supply, time). The dimension of the problem increases further if, say, the carpenter begins to offer pieces made from oak and mahogany, or oak, mahogany and walnut, or any number of different woods, with each material drawing from an additional supply and thus increasing the number of constraints by one.

2.1.1 Definition

The 0-1 MDKP with N items and M constraints can be defined as the following Integer Linear Program (ILP):

$$\textbf{maximise} \sum_{i=1}^{N} p_i x_i, \quad x_i \in \{0,1\}$$
$$\textbf{subject to} \sum_{i=1}^{N} W_{ij} x_i \leq c_j \quad \forall j \in \{1,...,M\},$$

where x_i is the inclusion variable for the item i, p_i is its corresponding profit, c_j is the capacity of the resource j and W_{ij} is the amount of that resource required by including item i, also called its weight with respect to resource j. All quantities are positive, and viewing them as integers gives no loss of generality when it comes to their finite representation on a computer, which we are ultimately interested in. We can also format them as the vectors $\boldsymbol{x} \in \{0,1\}^N$ (also called path), $\boldsymbol{p}, \boldsymbol{c} \in \mathbb{N}^N$ and the matrix $W \in \mathbb{N}^{M \times N}$, giving a more compact definition:

$$\textbf{maximise } \boldsymbol{p}^{\mathrm{T}} \boldsymbol{x}$$
$$\textbf{subject to } W\boldsymbol{x} \leq \boldsymbol{c},$$

where the $\leq$ relation is only fulfilled if it holds in each component. With this definition, we can interpret the constraint geometrically: $\boldsymbol{x}$ is a vertex of an N-dimensional unit hypercube, which is mapped to a vertex of an M-dimensional cuboid by the linear transformation given by W. The constraint is now fulfilled by those vertices which lie in the rectangular prism whose interior diagonal is given by $\boldsymbol{c}$. If $\boldsymbol{x}$ is mapped to this region, we call it a feasible solution to the problem; otherwise, $\boldsymbol{x}$ violates the constraint and is thus not of interest when it comes to maximising the profit.
As an example we may consider the following problem instance with $N = M = 2$ and profits, capacities and weights given by

$$\boldsymbol{p} = \begin{pmatrix} 5 \\ 3 \end{pmatrix}, \quad \boldsymbol{c} = \begin{pmatrix} 6 \\ 5 \end{pmatrix}, \quad W = \begin{pmatrix} 5 & 1 \\ 2 & 5 \end{pmatrix}.$$

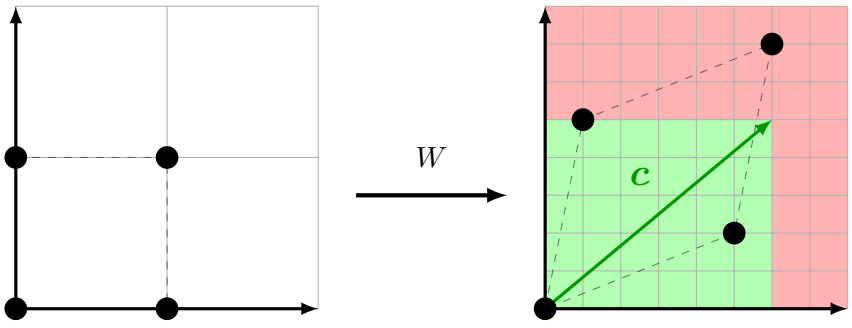

Figure 1: The vertices of the unit square are mapped by W to those of a parallelogram. Shaded in green on the right is the feasible region as defined by $\boldsymbol{c}$, beyond that lies the infeasible region in red.

We observe that $\boldsymbol{x}_0 = (0,0)$, $\boldsymbol{x}_1 = (0,1)$ and $\boldsymbol{x}_2 = (1,0)$ are possible solutions to the problem, while $\boldsymbol{x}_3 = (1,1)$ violates the second constraint $c_2 = 5$. Of the feasible solutions,

$\boldsymbol{x}_2$ has the largest profit $\boldsymbol{p}^{\mathrm{T}}\boldsymbol{x}_2 = 5$ associated to it, making it the optimal solution in this instance. Of course, finding the feasible solution which grants the most profit in the general case is no easy feat, hence giving rise to the issue of how to handle the problem algorithmically.

2.1.2 Overview of Classical Algorithms

In the literature, one finds many approaches when it comes to tackling 0-1 MDKP. For one, various heuristic algorithms have been developed, such as the application of Lagrange multipliers to find an upper bound on the achievable profit, first proposed by Magazine and Oguz[22] and later improved by Volgenant and Zoon[23], as well as greedy strategies like the method of Akçay et al.[24] for the general MDKP. More involved metaheuristics like the genetic algorithm by Chu and Beasley[25] or the artificial fish swarm algorithm by Azad et al.[26] also exist. On the side of exact solving, methods include the branch-and-bound algorithm of Gavish and Pirkul[27] involving surrogate relaxation and the CORAL procedure developed by Speranza and Mansini[28], which breaks the problem into smaller, easier-to-solve subproblems.
Another option for obtaining exact solutions to 0-1 MDKP is the general-purpose solver CPLEX from IBM[21]. Meant to solve any kind of linear optimisation or Linear Programming problem, it is also capable of tackling Quadratic and Mixed Integer Programming problems, of which the last category includes 0-1 MDKP. For their solution, CPLEX makes use of a branch-and-cut approach which we outline in the later Section 5.2. While not being aimed directly at solving 0-1 MDKP, CPLEX still represents the industry standard when it comes to optimisation, which is why we deem it appropriate to use as the classical method to which we will later compare the quantum algorithm.

2.2 Quantum Computation

The following presentation is roughly based on Chapters 4, 5 and 6 of *Quantum Computation and Quantum Information*[29] by Nielsen and Chuang, the standard textbook on quantum computation. It is meant to give some background on the theory needed for the subsequent treatment of the QTG, without delving to deeply into any particular subject (as this is beyond the scope of this thesis). The reader is assumed to be familiar with the fundamentals of quantum mechanics.

2.2.1 Preliminaries

The process of a computation on a quantum system can be essentially broken down into three separate parts:

1. The system is prepared to have a known initial state.

2. The state is evolved under application of unitary transformations.

3. A measurement of the system is conducted, resulting in a final state.

This is very similar to how one might describe a general quantum mechanical experiment, the key difference being that the state evolution occurs in a controlled fashion. Ultimately, it is this evolution which constitutes a quantum algorithm, which is why we first need some background on these transformations, as well as the state space they act on.

The stage needed to have any quantum computation play out is given by the qubit, which in turn means that our theoretical understanding is to be built upon its underlying two-dimensional Hilbert space $\mathcal{H} = \mathbb{C}^2$. We denote the orthonormal basis vectors of this space, with respect to which we make measurements, by $|0\rangle$ and $|1\rangle$, suggesting a relation to the states of a classical bit insofar as that the result of measuring the qubit state will be either $|0\rangle$ or $|1\rangle$. A general qubit state $|\Psi\rangle \in \mathbb{C}^2$ is given by a superposition

$$|\Psi\rangle = a|0\rangle + b|1\rangle, \quad a, b \in \mathbb{C},$$

that can also be represented as

$$|\Psi\rangle = \begin{pmatrix} a \\ b \end{pmatrix}$$

where the coefficients or amplitudes a and b fulfill $\langle \Psi \,|\, \Psi \rangle = |a|^2 + |b|^2 = 1$.
The invariance of this property is exactly what gives rise to unitary transformations, that is to say, linear maps (operators) $U : \mathbb{C}^2 \to \mathbb{C}^2$ for which we have

$$\langle \Psi | U^\dagger U | \Psi \rangle = \langle \Psi \,|\, \Psi \rangle = 1 \Rightarrow U^\dagger U = \mathbb{1}.$$

Since they are acting on $\mathbb{C}^2$, these maps can simply be represented by complex 2×2-matrices sufficing the above property. In context of quantum computation these operations are also called gates, as they allow for manipulation of a given state in analogy to a gate in a classical logic circuit. An easy example to illustrate this point is the X gate,

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

which has the simple property of flipping the amplitudes of a state $|\Psi\rangle$:

$$X|\Psi\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} b \\ a \end{pmatrix}$$

Applying X to a basis state we thus have $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$, which corresponds to the classical NOT gate. Another important gate is the Hadamard gate

$$\mathrm{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},$$

the action of which on a basis state results in a uniform superposition

$$\mathrm{H}|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \quad \mathrm{H}|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

The creation of state superposition is one aspect which separates quantum from classical computation, allowing for simultaneous consideration of multiple states. We will also make heavy use of this property later on. As the last example we consider the rotation gate:

$$R_k = \begin{pmatrix} 1 & 0 \\ 0 & \exp\left(\frac{2\pi i}{2^k}\right) \end{pmatrix}$$

Its effect is the introduction of the phase factor $\exp\left(\frac{2\pi i}{2^k}\right)$ to the $|1\rangle$ state, and it will also play a role in the further theory. For all of these and any other gates, we can of course

describe the action of subsequently applied gates by the product of their matrix representations.

Now, a single qubit is of as little use as a single classical bit. To conduct any real computation, we need to combine multiple qubits into a composite quantum system, which is mathematically described by the tensor product of the individual qubit spaces. Therefore, to reason about the state of N qubits we must consider the product Hilbert space

$$\mathcal{H}^{\otimes N} = \underbrace{\mathcal{H} \otimes \mathcal{H} \otimes ... \otimes \mathcal{H}}_{N \text{ times}} \cong \mathbb{C}^{2^N}.$$

We will also refer to this as a qubit register, especially when we delineate separate spaces of a combined system based on their purpose. As the basis for this space we can consider product states composed of the basis elements of the single qubit space, such as

$$|0\rangle \otimes |0\rangle \otimes ... \otimes |0\rangle = |00...0\rangle \in \mathcal{H}^{2^N}.$$

This set of states is often referred to as the computational basis, since we can view such states as corresponding to a classical bit string of length N. We will abbreviate their notation to this bit string or the base 10 representation, such that the states are labelled as $|0\rangle$, $|1\rangle$, ... $|2^N - 1\rangle$.
Similarly to the states, unitary transformations of a single qubit can also be composed via the tensor product in order to obtain a unitary transformation for the tensor product space. For instance, $\mathrm{H}^{\otimes N}$ allows for the creation of a uniform superposition of all computational basis states. Just using gates created this way does not allow for any computation however, since the single-qubit states are acted on individually by each tensor factor gate, meaning there is no interaction between them. We must include at least one additional unitary gate, which cannot be decomposed into operations on all single qubits, to allow for such cross-talk. The standard choice is the controlled-NOT (CNOT) gate for a two-qubit system, which flips the second (target) state if and only if the first (control) is $|1\rangle$. Thus, the CNOT gate is essentially just an X gate applied to the second qubit which is controlled on the first. This can be represented in a quantum circuit diagram where the evolution of each qubit is represented by a line, with the initial state at the left and the final state at the right end.

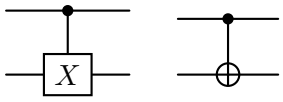

Figure 2: Two circuit representations of the CNOT gate, the right being used more commonly. The control of the X gate is drawn as a black circle on the corresponding qubit's line.

It is possible to also have the gate be controlled on the $|0\rangle$ state, which we mark in a diagram by an unfilled circle:

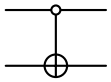

Figure 3: Circuit representation of the CNOT gate controlled on $|0\rangle$.

CNOT gates can be used to execute any unitary single-qubit gate in dependence of a control qubit. If we supply the CNOT gate with another control we get another important gate, the Toffoli gate:

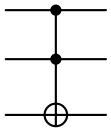

Figure 4: Circuit representation of the Toffoli gate.

We can extend the number of control qubits even further and consider a unitary single-qubit gate U being controlled on m qubits. Such a gate can be constructed by using $2(m-1)$ Toffoli gates and a single-controlled version of the unitary operation, at the cost of requiring $m-1$ additional so-called ancilla qubits to be added to the system. These ancilla qubits are initialised in the $|0\rangle$ state and must be reset to this state after being used (as they might be reused for other purposes). For example, taking $m = 3$ we have the following circuit diagram of the construction:

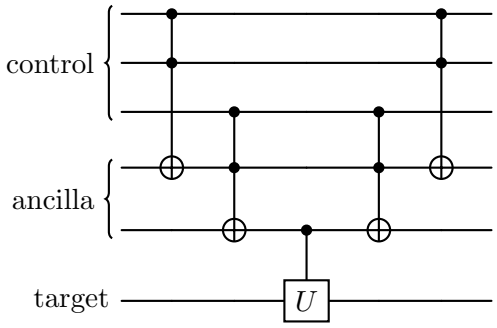

Figure 5: Decomposition of a 3-controlled gate into 4 Toffoli gates and one single-controlled gate under utilisation of 2 ancilla qubits.

The Toffoli gate alongside single-qubit gates and single-controlled rotation gates form what is known as a universal set of gates, which means that any given unitary operation on m qubits can be decomposed into these elementary gates up to an arbitrary accuracy. It is this set which we will also rely on for the construction of the QTG.

2.2.2 Quantum Comparison

As a special case of the multicontrolled application of a gate, we can consider the condition being that an integer A stored on a qubit register is greater or less than some other integer

B, which is given classically. $A = A_1 \cdot 2^{N-1} + ... + A_{N-1} \cdot 2^1 + A_N \cdot 2^0$ is encoded via its binary representation $A_1...A_{N-1}A_N$ as an N-qubit state $|A\rangle = |A_1...A_{N-1}A_N\rangle$, and we assume $B = (B_1...B_{N-1}B_N)_2$ to have the same binary length N as A (which can be achieved in general by padding the lesser integer with sufficiently many zeros). The control structure can then be deduced by viewing the bits A_i, B_i of the integers as Boolean variables and constructing the truth value of the comparison out of their logical negation, con- and disjunction (denoted respectively by $\overline{A_i}, A_i \wedge B_i$ and $A_i \vee B_i$), analogously to a classical digital comparator. Introducing the bit-wise equality variables

$$C_i = \overline{A_i B_i} \vee A_i B_i,$$

which are 1 only if $A_i = B_i$, the value of the strict inequality $A > B$ is achieved by

$$(A > B) = \bigvee_{i=1}^{N} \left(A_i \overline{B_i} \bigwedge_{j=i+1}^{N} C_j \right).$$

Since we know B and thus its bits B_i, all C_i are predetermined (being either A_i or $\overline{A_i}$), meaning the expression is in disjunctive normal form, where the disjoint variables themselves contain no disjunctions nor negated conjunctions. Additionally, they are also orthogonal, as only one of the conjuctions produces a value of 1. This makes it easy to translate the comparator into a quantum circuit: We take N gates which are controlled on those bits of A which make up the conjunctions. As the expression for $A > B$ is orthogonal, we are guaranteed the execution of only one of the multicontrolled gates. We can further leave out any gates where $\overline{B_i} = 0 \Leftrightarrow B_i = 1$, as these do not contribute to the comparison. In a practical example, we implement a gate U being controlled by the comparison $A > 10 = 1010_2$. We introduce one gate for each 0-bit, which are controlled on the expressions

$$A_2 \wedge A_3 \wedge \overline{A_4} \text{ and } A_4.$$

The corresponding circuit diagram then is:

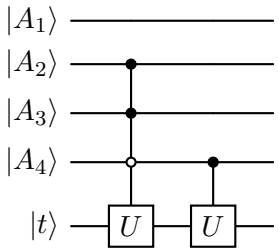

Figure 6: Application of U to the target qubit $|t\rangle$ only occurs if the integer stored on the first four qubits is larger than 10.

Using the same control sequence we can also implement inequalities $A \geq B$; for this, we simply make the expression strict by subtracting one from B to get $A > B - 1$.

2.2.3 Quantum Fourier Transform Adder

With the gates discussed thus far we are able to implement the addition of two integers A and B represented binarily on qubit registers. This is done via performing a quantum Fourier transform on the register which will store the sum, followed by several rotation

gates R_k controlled on the other register in a method developed by Draper[30]. For our purposes, it is however enough to consider only one number stored in qubits, while the other is given as a piece of classical data.
We shall first briefly give an overview of the quantum Fourier transform (QFT). A is, again, represented binarily by the N-qubit state $|A\rangle = |A_1...A_{N-1}A_N\rangle$. The QFT of $|A\rangle$ is

$$\mathcal{QFT}|A\rangle = \frac{1}{2^{\frac{N}{2}}} \sum_{l=0}^{2^N-1} \exp\left(\frac{2\pi i A l}{2^N}\right)|l\rangle.$$

This resulting state can be expressed as the tensor product

$$\begin{aligned}\mathcal{QFT}|A\rangle = &\frac{1}{2^{\frac{N}{2}}}(|0\rangle + \exp(2\pi i \cdot 0.A_N)|1\rangle) \otimes (|0\rangle + \exp(2\pi i \cdot 0.A_{N-1}A_N)|1\rangle) \otimes \\ &... \otimes (|0\rangle + \exp(2\pi i \cdot 0.A_1...A_{N-1}A_N)|1\rangle),\end{aligned}$$

where $0.A_1...A_{N-1}A_N$ denotes the binary fraction $A_1 \cdot 2^{-1} + ... + A_{N-1} \cdot 2^{-(N-1)} + A_N \cdot 2^{-N}$. As we can see, each factor is a superposition of the single-qubit basis states where the $|1\rangle$ state has obtained a certain phase factor dependent on the digits A_l. Naturally, to implement QFT we must use Hadamard gates H and rotation gates R_k. The strategy involves transforming each digit $|A_l\rangle$ individually by first applying H, giving the superposition

$$\mathrm{H}|A_l\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + \exp(2\pi i \cdot 0.A_l)|1\rangle),$$

and then repeatedly applying R_k controlled on the subsequent digits $|A_{l+1}\rangle$ through $|A_N\rangle$, where $k \in \{2, ..., N - l + 1\}$, thus increasing the phase of $|1\rangle$ up to $\exp(2\pi i \cdot 0.A_l...A_N)$. The last digit $|A_N\rangle$ is only put through a Hadamard gate. A scheme of the exact circuit is provided below[1].

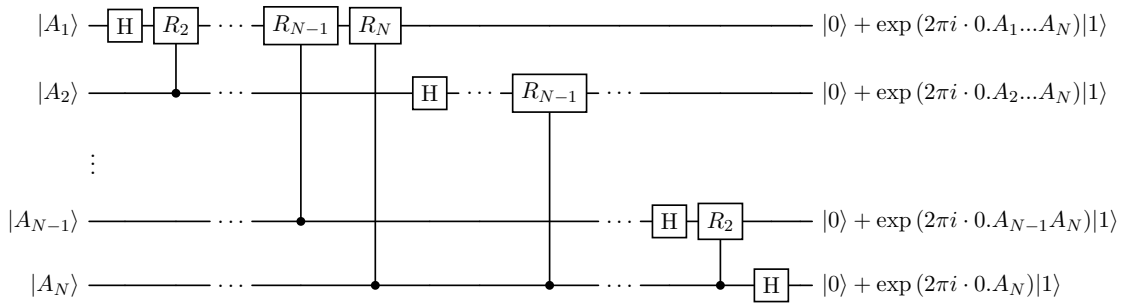

Figure 7: Circuit scheme of QFT (normalisation not included).

It should be noted that this technically does not give the QFT as defined above, since the order of the tensor factors is reversed. Restoring the order is in principle possible via unitary swapping of the states, however unnecessary in practice since any application of QFT will be eventually complemented by its inverse operation (achieved by mirroring the circuit vertically), whereby we also reset the ordering, saving the gates for swapping.
Having transformed our register, we can now make use of the fact that the integer is stored

[1] Adapted from [29], p. 219.

in the phases of the states to add another integer to it. We just apply more rotation gates depending on the second integer in order to increase the phases further until they store the sum, which we then are able to retrieve via the inverse QFT. We thus consider the binary representation of another integer $B = B_1...B_{N-1}B_N$ and assume the sum to still fit on the register. Similarly to QFT itself, we apply the rotation gates R_k in ascending order to each qubit state with phase $\exp(2\pi i \cdot 0.A_l..A_N)$, replacing H with R_1 and ending with R_{N-l+1}. In addition to that, we associate each rotation to a digit of B, including R_k only if $B_k = 1$ and doing so from B_l up to B_N. The resulting product gate is

$$\bigotimes_{l=1}^{N} \prod_{k \in \mathcal{K}_l} R_{k-l+1}, \quad \mathcal{K}_l = \{k = l, l+1, ..., N \mid B_k = 1\}$$

which gives the desired mapping

$$|0\rangle + \exp(2\pi i \cdot 0.A_l...A_N)|1\rangle \rightarrow |0\rangle + \exp(2\pi i(0.A_l...A_N + 0.B_l...B_N))|1\rangle$$

in each tensor factor. As an example we can consider the addition of $A = 9 = 1001_2$ and $B = 5 = 0101_2$, illustrated in the following circuit diagram:

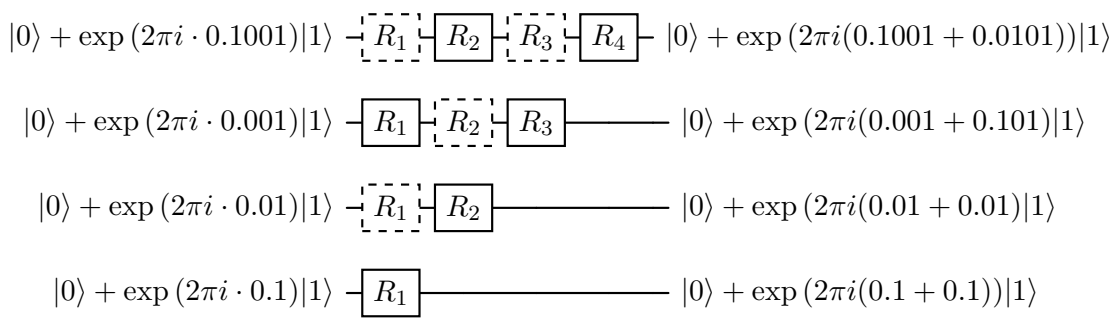

Figure 8: Circuit scheme of the addition of $B = 5$ to a transformed register storing $A = 9$. The dashed omitted gates are only drawn for clarity.

The parallel execution of the gates depicted here will not be possible in the QTG, as addition there is controlled on a qubit state. We will see that we are still able to run the gates in tandem, for that we however need to introduce ancilla qubits. As a concluding remark, we note that the same method can be used for subtraction; one only has to exchange the rotation gates for their adjoint $R_k^\dagger$, which contribute the phase factor $\exp\left(-\frac{2\pi i}{2^k}\right)$ to the $|1\rangle$ state.

2.2.4 Grover's Algorithm and Quantum Amplitude Amplification

As mentioned in the introduction, Grover's algorithm serves as a search method over an unstructured discrete space, whose elements we represent as a binary integer $\boldsymbol{x}$ via the computational basis of a qubit register with sufficient size N. We define a decision function $f(\boldsymbol{x})$ which outputs $f(\boldsymbol{x}) = 1$ if and only if $\boldsymbol{x}$ is a solution to the search problem and 0 otherwise, and thus is able to recognize the searched elements. This function can be extended to a unitary operator $\mathcal{S}_f$ called the phase oracle which flips the phase of a state $|\boldsymbol{x}\rangle$ depending on if $\boldsymbol{x}$ is a solution:

$$\mathcal{S}_f|\boldsymbol{x}\rangle = (-1)^{f(\boldsymbol{x})}|\boldsymbol{x}\rangle$$

To implement such a phase flip we need to add one additional qubit to the system, which is of practical interest but shall not bother us for the remaining discussion. We will also view $\mathcal{S}_f$ as a black box for the moment and not consider how it is constructed.
Using $\mathcal{S}_f$, we can now describe the procedure of Grover's algorithm:

0. Create a uniform superposition $|\Psi\rangle$ out of $|0\rangle$ by applying $\mathrm{H}^{\otimes N}$.

1. Apply the phase oracle $\mathcal{S}_f$.

2. Apply Hadamard gate $\mathrm{H}^{\otimes N}$.

3. Apply a phase shift of -1 to all computational basis states except $|0\rangle$ (using $\mathcal{S}_0|\boldsymbol{x}\rangle = -(-1)^{\delta_{\boldsymbol{x}0}}|\boldsymbol{x}\rangle$)

4. Again apply Hadamard gate $\mathrm{H}^{\otimes N}$.

Steps 1. through 4. are then iterated a certain number of times. The corresponding operations can be joined into the Grover operator $G := D\mathcal{S}_f$, where the first three gates can be viewed separately as the diffusion operator $D := \mathrm{H}^{\otimes N}\mathcal{S}_0\mathrm{H}^{\otimes N}$.
It is not immediately clear how this is supposed to provide a solution to the search problem. There is, thankfully, a great geometric visualisation of what the Grover operator accomplishes. For this, we take $\mathcal{L}$ to be the solution space of the search with cardinality $m = |\mathcal{L}|$ and view the two-dimensional subspace spanned by the states ($n := 2^N$)

$$|\alpha\rangle = \frac{1}{\sqrt{n-m}} \sum_{\boldsymbol{x}\notin\mathcal{L}} |\boldsymbol{x}\rangle, \quad |\beta\rangle = \frac{1}{\sqrt{m}} \sum_{\boldsymbol{x}\in\mathcal{L}} |\boldsymbol{x}\rangle.$$

The initial superposition

$$|\Psi\rangle = \mathrm{H}^{\otimes N}|0\rangle = \frac{1}{\sqrt{n}} \sum_{\boldsymbol{x}=0}^{n-1} |\boldsymbol{x}\rangle = \sqrt{\frac{n-m}{n}}|\alpha\rangle + \sqrt{\frac{m}{n}}|\beta\rangle$$

is within this subspace. In the first step, applying $\mathcal{S}_f$ to $|\Psi\rangle$ now flips the phase of the first term, performing a reflection along the $|\alpha\rangle$ state. Next, it can be shown that the diffusion operator D also achieves a similar reflection of $\mathcal{S}_f|\Psi\rangle$, this time about the initial superposition $|\Psi\rangle$. The resulting state has has thus been reflected twice, constituting a rotation which leaves $G|\Psi\rangle$ closer to the $|\beta\rangle$ state, meaning a measurement is more likely to give a state representing an element in the solution set $\mathcal{L}$. This action of G can be depicted[2] in the subspace spanned by $|\alpha\rangle$ and $|\beta\rangle$:

[2] Adapted from [29], p. 253.

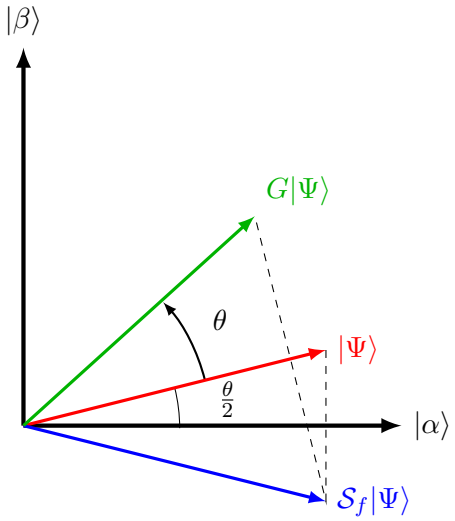

Figure 9: Geometric picture behind one Grover iteration.

If we denote by θ the angle of $|\Psi\rangle$ to $\mathcal{S}_f|\Psi\rangle$, meaning that initially

$$|\Psi\rangle = \cos\frac{\theta}{2}|\alpha\rangle + \sin\frac{\theta}{2}|\beta\rangle,$$

then this is the same angle between $|\Psi\rangle$ and $G\Psi$, thus giving

$$G|\Psi\rangle = \cos\frac{3\theta}{2}|\alpha\rangle + \sin\frac{3\theta}{2}|\beta\rangle.$$

From here we can see that a k-fold application of G yields

$$G^k\Psi = \cos\left(\frac{2k+1}{2}\theta\right)|\alpha\rangle + \sin\left(\frac{2k+1}{2}\theta\right)|\beta\rangle.$$

Since we are rotating the state by a fixed angle, the iteration number k must be chosen carefully so as to leave the state maximally close to $|\beta\rangle$. An upper bound for an optimal value of k_{opt} is given by

$$k_{opt} = \left\lceil \frac{\pi}{4}\sqrt{\frac{n}{m}} \right\rceil,$$

and so the Grover operator G has to be applied $k_{opt} = O(\sqrt{n/m})$ times, which gives the quadratic speedup over the $O(n/m)$ calls to the oracle a classical algorithm would require. We can formulate the algorithm for a search with a single solution $\hat{\boldsymbol{x}}$ ($m = 1$) and corresponding decision function f, which makes use of the Grover method, as follows:

Algorithm 1 GroverQSearch(f)

1: Initialize qubit register in $|0\rangle$;
2: Apply $\mathrm{H}^{\otimes N}$ to $|0\rangle$ giving $|\Psi\rangle$;
3: Apply G to $|\Psi\rangle$ repeatedly for $k_{opt} = \left\lceil \frac{\pi}{4}\sqrt{n} \right\rceil$ iterations;
4: Measure the register with probable outcome $|\hat{\boldsymbol{x}}\rangle$;
5: **end**

There is a way to generalise Grover's algorithm called Quantum Amplitude Amplification (QAA), first proposed by Brassard et al.[31]. Notice that we are working with the uniform

superposition of all computational basis states as achieved by the use of the Hadamard gate $\mathrm{H}^{\otimes N}$ on the initial $|0\rangle$ state. In QAA, $\mathrm{H}^{\otimes N}$ is exchanged for an arbitrary gate $\mathcal{A}$ which creates a superposition of states, giving control over the distribution of their amplitudes by employing an $\mathcal{A}$ which we can tailor to the specific search problem. The Grover operator G is then altered to the Amplitude Amplification operator

$$\mathcal{Q} := \mathcal{A}\mathcal{S}_0\mathcal{A}^\dagger\mathcal{S}_f.$$

This also performs the rotation of a state, however in a different subspace since the amplitudes of the basis states making up $|\Psi'\rangle = \mathcal{A}|0\rangle$ are not necessarily equal anymore. The algorithm which involves $\mathcal{Q}$ also provides a significant benefit over Grover's version, in that one does not have to know a priori the size m of the solution set, making it applicable to a wider class of problems. A slightly modified version of the algorithm will be presented in the next Section.

3 QTG for 0-1 MDKP

Having set the foundations for it, we are now able to describe the QTG algorithm for the 0-1 MDKP. Put briefly, we make use of QAA to boost the probability of measuring an optimum in the finite space of possible solutions. For this, we will need a special Amplitude Amplification operator $\mathcal{G}$ (itself called the Quantum Tree Generator), which creates a state superposition by subsequent consideration of the N items.
We start off with a short rundown of the register Hilbert spaces and follow with an explanation of how the QTG $\mathcal{G}$ is constructed. Then, we introduce the **QMaxSearch** routine which employs $\mathcal{G}$ as part of QAA. Our treatment follows the original proposal by Wilkening et al.[18], going into detail where it is appropriate.

3.1 Register Spaces

Before we examine the actual algorithm, it is worth to briefly consider the space on which it runs. As described in Section 2.1.1, an instance of the 0-1 MDKP is defined by the path $\boldsymbol{x} \in \{0,1\}^N$ of N items with profits $\boldsymbol{p} \in \mathbb{N}^N$ and weights $W \in \mathbb{N}^{M\times N}$ as well as M capacities $\boldsymbol{c} \in \mathbb{N}^M$. In the algorithm, we want to be able to create superpositions of all possible $\boldsymbol{x}$ and regulate their measurement probabilities depending on if they are a) feasible and b) of certain profit, meaning our quantum system has to accommodate at least the path $\boldsymbol{x}$ and the corresponding remaining capacity $\boldsymbol{C}(\boldsymbol{x}) := \boldsymbol{c} - W\boldsymbol{x}$ as well as total profit $P(\boldsymbol{x}) := \boldsymbol{p}^{\mathrm{T}}\boldsymbol{x}$. We store these quantities in three registers:

1. The path register $\mathcal{H}_1$ for the binary vector $|\boldsymbol{x}\rangle^1$, consisting of $\mathbf{Q}_{\mathrm{PATH}} = N$ qubits.

2. The composite capacity register

$$\mathcal{H}_2 = \bigotimes_{j=1}^{M} \mathcal{H}_{2,j},$$

where $\mathcal{H}_{2,j}$ stores the remaining capacity $|C_j(\boldsymbol{x})\rangle^{2,j}$. We write as a shorthand

$$|\boldsymbol{C}(\boldsymbol{x})\rangle^2 := \bigotimes_{j=1}^{M} |C_j(\boldsymbol{x})\rangle^{2,j}.$$

These need at least $\lfloor \log_2(c_j) \rfloor + 1$ bits to be represented, giving a total qubit number of

$$\mathbf{Q}_{\text{CAPACITY}} = \sum_{j=1}^{M} (\lfloor \log_2(c_j) \rfloor + 1).$$

3. The profit register $\mathcal{H}_3$ for the total profit $|P(\boldsymbol{x})\rangle^3$, which is made of as many qubits as is necessary to represent the optimal profit. Since this is not known beforehand, we require the register to store any upper bound on the profit P_{UB}, meaning $\mathbf{Q}_{\text{PROFIT}} = \lfloor \log_2(P_{\text{UB}}) \rfloor + 1$.

Notationally, we use superscripts to relate a given state $|\Psi\rangle^i$ to the space $\mathcal{H}_i$ of which it is an element. The algorithm will also require a certain amount of ancilla qubits, whose count we will determine during resource estimation. For the analysis, it suffices to restrict our attention to the three registers listed above, making $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2 \otimes \mathcal{H}_3$ the stage on which the QTG algorithm plays out. We denote its tensor product states by

$$|\boldsymbol{x}\rangle^1 \otimes |\boldsymbol{C}(\boldsymbol{x})\rangle^2 \otimes |P(\boldsymbol{x})\rangle^3 \in \mathcal{H}.$$

3.2 Quantum Tree Generator $\mathcal{G}$

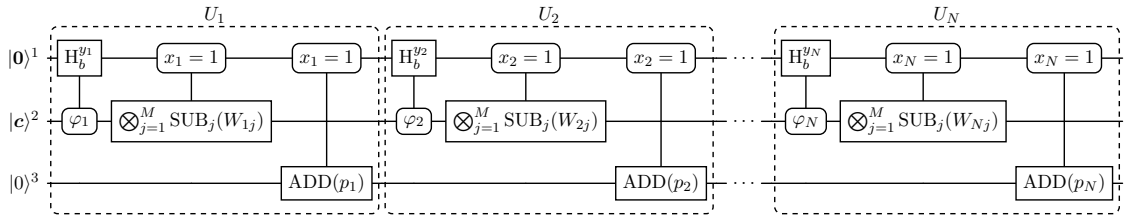

Figure 10: Basic circuit scheme of $\mathcal{G}$.

With the scene set, we now construct the operator $\mathcal{G}$ to generate the superposition required for QAA. Its general logic is to sequentially consider each item $i \in \{1, ..., N\}$, checking first if its weights W_{ij} comply with the remaining capacities C_j. Only if this is the case, a biased version of the Hadamard gate

$$\mathrm{H}_b^{y_i} = \frac{1}{\sqrt{b+2}} \begin{pmatrix} \sqrt{1+(1-y_i)b} & \sqrt{1+y_i b} \\ \sqrt{1+y_i b} & -\sqrt{1+(1-y_i)b} \end{pmatrix}$$

is applied to the state $|x_i\rangle^1$ representing the item's inclusion, leading to its superposition. Here, y_i is the i-th bit of an intermediate solution path $\boldsymbol{y}$ and b represents a bias parameter. An appropriate choice of b results in a more favourable amplitude distribution; this, as well as the intermediate solution $\boldsymbol{y}$, are explained shortly. Next, the remaining capacities C_j as well as the currently available profit P have to be updated by subtraction of W_{ij} and addition of p_i, respectively. This has to be done via operations acting on $|\boldsymbol{C}\rangle^2$ and $|P\rangle^3$ which are controlled on $|x_i\rangle^1$, since we are considering the cases where i is in- and excluded simultaneously via the superposition obtained in the first step. We can summarise the QTG as:

1: **for** each item $i = 1, ..., N$ **do**
2: Apply the biased Hadamard gate $\mathrm{H}_b^{y_i}$ to $|x_i\rangle^1$ if $W_{ij} \leq C_j$ for all $j = 1, ..., M$;
3: Controlled on $|x_i\rangle^1$, update $|\boldsymbol{C}\rangle^2$ such that $C'_j = C_j - W_{ij}$;
4: Controlled on $|x_i\rangle^1$, update $|P\rangle^3$ such that $P' = P + p_i$
5: **end for**

If we start with the state $|\mathbf{0}\rangle^1 \otimes |\boldsymbol{c}\rangle^2 \otimes |0\rangle^3$ and execute this procedure with all items, we are left with a state which is the sum of all states representing feasible solutions to the given instance of 0-1 MDKP, each with a certain amplitude depending on the chosen bias b.
After this brief overview of the functioning of $\mathcal{G}$, we now turn to its mathematical representation. As the items are considered one by one, we can formulate a general unitary operator U_i which implements the three steps described above for a single item i. We denote by $C_\varphi^k(A)$ the conditional action of an operator A depending on a Boolean expression φ which is checked on the register $\mathcal{H}_k$. Addition of an integer a to the profit register $\mathcal{H}_3$ is given the notation $\mathrm{ADD}(a)$, while its subtraction on the j-th capacity register $\mathcal{H}_{2,j}$ is written as $\mathrm{SUB}_j(a)$. Each step itself can now be given a unitary gate:

$$\text{Step 1: } U_i^1 = C_{\varphi_i}^2(\mathrm{H}_b^{y_i}), \quad \text{where } \varphi_i = \bigwedge_{j=1}^{M} (W_{ij} \leq C_j)$$

$$\text{Step 2: } U_i^2 = C_{x_i=1}^1 \left(\bigotimes_{j=1}^{M} \mathrm{SUB}_j(W_{ij}) \right)$$

$$\text{Step 3: } U_i^3 = C_{x_i=1}^1(\mathrm{ADD}(p_i))$$

The exact implementation of these controlled gates will be treated separately as part of the resource estimation in Section 4. We now combine them to form the desired operator $U_i = U_i^3 U_i^2 U_i^1$ for the i-th layer of the QTG. Composing the layer unitaries of each item, we have the QTG itself:

$$\mathcal{G} = \prod_{i=1}^{N} U_i.$$

Figure 10 shows a rough sketch of its circuit.

3.3 But what is a Quantum Tree?

The action of $\mathcal{G}$ can be understood intuitively as the creation of a rooted tree graph, whose nodes represent the feasible states obtained at each level of item in-/exclusion, each with an associated measurement probability. The branching of a node at height i into two at height $i + 1$ occurs only if the remaining capacities of the corresponding state allow for the inclusion of item i, otherwise the node is simply repeated at the next height. The probabilities of the node children are modified in accordance with application of the biased Hadamard gate, if no branching occurs the child simply inherits the parent's probability. As this is precisely what we make use of for the simulation of the QTG algorithm, the exact details are reserved for Section 5. For now, we give a simple demonstration using the example provided in Section 2.1.1 with the bias set to $b = 0$ such that the probabilities are evenly distributed:

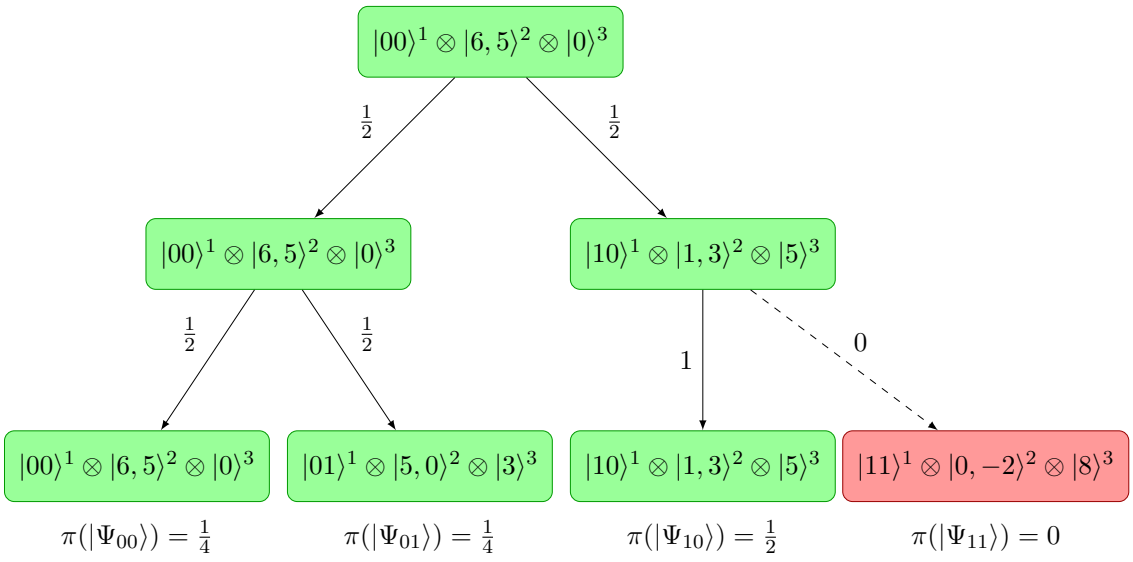

Figure 11: Tree graph of the example in Section 2.1.1. Attached below each leaf node is the probability $\pi(|\Psi_{\boldsymbol{x}}\rangle)$ of obtaining $|\Psi_{\boldsymbol{x}}\rangle = |\boldsymbol{x}\rangle^1 \otimes |\boldsymbol{C}(\boldsymbol{x})\rangle^2 \otimes |P(\boldsymbol{x})\rangle^3$ as the outcome of measuring the composite register $\mathcal{H}$. Note that the infeasible path $\boldsymbol{x}_3 = 11$ is discarded.

3.4 Quantum Maximum Search

With $\mathcal{G}$ in hand, we proceed with its introduction into the **QSearch** algorithm to filter for feasible solutions $\boldsymbol{x}$, whose profit fulfills $f(P(\boldsymbol{x})) = 1$ for a decision function f we have yet to choose. We may first consider the action of the Amplitude Amplification operator $\mathcal{Q} = \mathcal{G}\mathcal{S}_0\mathcal{G}^\dagger\mathcal{S}_f$ on the superposition obtained by applying $\mathcal{G}$ to the initial state $|\Psi_0\rangle = |\boldsymbol{0}\rangle^1 \otimes |\boldsymbol{c}\rangle^2 \otimes |0\rangle^3$. For this, let $\mathcal{F}$ be the set of feasible paths and $\mathcal{L}$ the set of solution paths to f. We can then write

$$|\Psi_{\mathcal{G}}\rangle = \mathcal{G}|\Psi_0\rangle = \sum_{\boldsymbol{x}\in\mathcal{L}} a_{\boldsymbol{x}}|\Psi_{\boldsymbol{x}}\rangle + \sum_{\boldsymbol{x}'\in\mathcal{F}\setminus\mathcal{L}} a_{\boldsymbol{x}'}|\Psi_{\boldsymbol{x}'}\rangle = a_T|\mathcal{L}\rangle + a_R|\mathcal{F}\setminus\mathcal{L}\rangle,$$

where states fulfilling/not fulfilling $f(P(\boldsymbol{x})) = 1$ are respectively collected in $|\mathcal{L}\rangle, |\mathcal{F}\setminus\mathcal{L}\rangle$, and their amplitudes are combined into a_T, a_R which obey

$$\begin{aligned} |a_T|^2 &:= \sum_{\boldsymbol{x}\in\mathcal{L}} |a_{\boldsymbol{x}}|^2 = \pi(|\mathcal{L}\rangle), \\ |a_R|^2 &= 1 - |a_T|^2 = \pi(|\mathcal{F}\setminus\mathcal{L}\rangle). \end{aligned}$$

For j-fold application of $\mathcal{Q}$ to $|\Psi_{\mathcal{G}}\rangle$ we now obtain a familiar looking result[31]:

$$\mathcal{Q}^j|\Psi_{\mathcal{G}}\rangle = \frac{\sin\left((2j+1)\arcsin(\sqrt{\pi(|\mathcal{L}\rangle)})\right)}{\sqrt{\pi(|\mathcal{L}\rangle)}}|\mathcal{L}\rangle + \frac{\cos\left((2j+1)\arcsin(\sqrt{\pi(|\mathcal{L}\rangle)})\right)}{\sqrt{1-\pi(|\mathcal{L}\rangle)}}|\mathcal{F}\setminus\mathcal{L}\rangle. \tag{1}$$

In contrast to Grover's algorithm 1, however, we do not know what j gives us the maximal probability of measuring a solution. To work around this, we employ the following algorithm:

Algorithm 2 QSearch$(\mathcal{G}, f, M_{\text{iter}})$

1: Set $l = 0, m_{tot} = 0$ and let c be any constant such that $1 < c < 2$;
2: Increase l by 1 and set $m = \lceil c^l \rceil$;
3: Apply $\mathcal{G}$ to the initial state $|\Psi_0\rangle = |\mathbf{0}\rangle^1 \otimes |\boldsymbol{c}\rangle^2 \otimes |0\rangle^3$;
4: Choose $j \in [1, m]$ uniformly at random and increase m_{tot} by $2j + 1$;
5: Apply $\mathcal{Q}^j$ to the superposition $\mathcal{G}|\Psi_0\rangle$;
6: Measure the registers $\mathcal{H}_1$ and $\mathcal{H}_3$ with outcomes $|\boldsymbol{x}\rangle^1$ and $|P(\boldsymbol{x})\rangle^3$;
7: **if** $f(P(\boldsymbol{x})) = 1$ or $m_{tot} \geq M_{\text{iter}}$ **then**
8: return $\boldsymbol{x}, P(\boldsymbol{x})$;
9: **else**
10: go to step 2;
11: **end if**

In essence, the approach revolves around drawing the iteration number j from $[1, m]$ at random. The upper endpoint of this interval $m = \lceil c^l \rceil$ increases exponentially with the number l of individual runs, such that the optimal j is quickly included in the probing space. This version has been modified from the original proposal[31], which runs forever if no solution for f exists, by the addition of a maximum iteration number M_{iter}. This is important, as we now take the decision function to give the result 1 only if $P(\boldsymbol{x}) > T$, where T is some set profit threshold. The idea behind **QMaxSearch**, first given by Dürr and Høyer for the inverse case of finding the minimum[32], is to begin with some initial threshold T_0 and execute **QSearch** with f deciding $P(\boldsymbol{x}) > T_0$. In practice, T_0 may be obtained by a classical greedy algorithm for 0-1 MDKP, as proposed in the QTG algorithm for 0-1 KP. The returned pair $(\boldsymbol{x}_1, P(\boldsymbol{x}_1))$ is then set to be the new threshold $T_1 := P(\boldsymbol{x}_1)$, for which we repeat **QSearch** with an updated decision function. This is iterated until **QSearch** cannot find a better solution anymore, which necessitates the introduction of a cutoff number M_{iter} to guarantee its termination. The **QMaxSearch** routine can be summarised as:

Algorithm 3 QMaxSearch$(\mathcal{G}, M_{\text{iter}})$

1: Greedily calculate initial threshold T and corresponding path $\boldsymbol{x}_{out}$;
2: **while** true **do**
3: $\boldsymbol{x}, P(\boldsymbol{x}) =$ **QSearch**$(\mathcal{G}, f_{P(\boldsymbol{x})>T}, M_{\text{iter}})$;
4: **if** $P(\boldsymbol{x}) > T$ **then**
5: Set $T = P(\boldsymbol{x})$ and $\boldsymbol{x}_{out} = \boldsymbol{x}$;
6: **else**
7: return $\boldsymbol{x}, P(\boldsymbol{x})$;
8: **end if**
9: **end while**

3.4.1 The Biased Hadamard Gate

Before moving on to the resource estimation, we still owe an explanation of the biased Hadamard gate $\mathrm{H}_b^{y_i}$ introduced earlier. To see its effect easier, we split its representation from the general case into $y_i = 0$ and $y_i = 1$:

$$\mathrm{H}_b^0 = \frac{1}{\sqrt{b+2}} \begin{pmatrix} \sqrt{1+b} & 1 \\ 1 & -\sqrt{1+b} \end{pmatrix}, \quad \mathrm{H}_b^1 = \frac{1}{\sqrt{b+2}} \begin{pmatrix} 1 & \sqrt{1+b} \\ \sqrt{1+b} & -1 \end{pmatrix}$$

We now consider the gate as part of the first layer unitary U_1. For this, we take $\boldsymbol{y}$ to be the intermediate solution, whose profit serves as the current threshold for **QSearch**, and $\boldsymbol{x}, \boldsymbol{x}'$ to be arbitrary paths which differ only in the first bit, $x_1 = 0, x_1' = 1$. If the first bit of the intermediate solution is, say, $y_i = 1$, and assuming all constraints are met when including the first item, we have U_1 acting on $\boldsymbol{x}$ as

$$U_1|\boldsymbol{x}\rangle^1 \otimes |\boldsymbol{C}(\boldsymbol{x})\rangle^2 \otimes |P(\boldsymbol{x})\rangle^3 = \frac{1}{\sqrt{b+2}}|\boldsymbol{x}\rangle^1 \otimes |\boldsymbol{C}(\boldsymbol{x})\rangle^2 \otimes |P(\boldsymbol{x})\rangle^3 + \frac{\sqrt{b+1}}{\sqrt{b+2}}|\boldsymbol{x}'\rangle^1 \otimes |\boldsymbol{C}(\boldsymbol{x}')\rangle^2 \otimes |P(\boldsymbol{x}')\rangle^3.$$

We observe that the amplitude of the state whose path $\boldsymbol{x}'$ shares the first bit with $\boldsymbol{y}$ is modified differently than that of the state corresponding to $\boldsymbol{x}$, which does not agree with $\boldsymbol{y}$ in the first bit. One can now extrapolate that a state, whose path $\boldsymbol{x}$ has a Hamming distance of $\Delta := \Delta(\boldsymbol{x}, \boldsymbol{y})$ (that is, differs from $\boldsymbol{y}$ in Δ bits), will have an associated amplitude of

$$a_{\boldsymbol{x}} = \pm \left(\frac{\sqrt{b+1}}{\sqrt{b+2}}\right)^{N-\Delta} \left(\frac{1}{\sqrt{b+2}}\right)^{\Delta}$$

and thus the probability

$$\pi(\Psi_{\boldsymbol{x}}) = |a_{\boldsymbol{x}}|^2 = \left(\frac{b+1}{b+2}\right)^{N-\Delta} \left(\frac{1}{b+2}\right)^{\Delta} \tag{2}$$

of being measured among all feasible paths. We can determine the bias b_{opt} which maximises this probability to be

$$b_{opt} = \frac{N}{\Delta} - 2 \approx \frac{N}{\Delta} \text{ for } N \text{ sufficiently large.}$$

Choosing this value for b increases the probability of measuring paths with Hamming distance Δ to the intermediate solution $\boldsymbol{y}$, which gives the ability to focus the search on the vicinity of a given $\boldsymbol{y}$. This is important, as the greedily calculated solution marking the first threshold for **QSearch** already contains items of good efficiency, which are then often part of the optimal solution. We refer to a remark by Wilkening et al.[18], where they note that the introduction of the biased Hadamard gates is necessary to obtain a competitive edge on classical solvers for 0-1 KP.

4 Resource Estimation

We will now go into detail on the resources required by **QMaxSearch**, while also giving a precise description on how each part of the algorithm is constructed. The performance of the QTG algorithm for a given instance of the 0-1 MDKP can be characterised by three benchmark parameters: The total number **Q** of qubits involved, the total number **G** of elementary gates applied and the total count **C** of QPU cycles, that is, the number of simultaneous gate executions. For the calculations of their estimates, we make the following assumptions:

(i) All qubits and gates are ideal, noiseless logical components.

(ii) It is of equal computational expense to implement any gate of the universal set consisting of single-qubit gates, single-controlled single-qubit rotations and Toffoli gates.

(iii) All disjoint gates can be executed as part of the same QPU cycle.

(iv) All m-controlled single qubit gates can be constructed from $2(m-1)$ Toffoli gates and one single-controlled single-qubit gate under the use of $m-1$ ancilla qubits.

(v) All multicontrolled gates have access to the same ancilla qubits.

(vi) It is of equal computational expense to have a gate be controlled on 0 or 1.

When multiple implementations of a subroutine are possible, we always choose the one with the least amount of cycles required, as to achieve a more favourable result when comparing to the classical solver.

4.1 Qubits

We begin with the qubit count. In Section 3.1 we already determined the amounts of quantum memory needed for storing the path variable, the remaining capacities and the total profit as

$$\mathbf{Q}_{\text{PATH}} = N, \quad \mathbf{Q}_{\text{CAPACITY}} = \sum_{j=1}^{M} |c_j|_2, \quad \mathbf{Q}_{\text{PROFIT}} = |P_{\text{UB}}|_2,$$

where we introduce the shorthand $|A|_2 = \lfloor \log_2(A) \rfloor + 1$ to denote the binary length of an integer A. To this we now have to add a certain amount $\mathbf{Q}_{\text{ANCILLA}}$ of ancilla qubits, stemming from the multicontrolled Hadamard gates $C^2_{\varphi}(\mathrm{H}^{y_i}_b)$, the addition and subtraction on the profit and capacity registers as well as the phase oracles $\mathcal{S}_f$, $\mathcal{S}_0$.
The control condition for the Hadamard gates is

$$\varphi = \bigwedge_{j=1}^{M} (W_{ij} \leq C_j),$$

which means we need to consider M comparisons of the type $A \geq B$, whose results have to be conjoined. By (iv), the multicontrolled gates for the comparisons can be implemented using at most $|c_j|_2 - 1$ ancillas for each constraint, which, for simplicity, we assume to be actually needed in each constraint for at least one of the comparisons. For the conjunction, we save the results of the comparisons on M additional qubits, on which we then control the Hadamard gate. This would require $M-1$ more gates, however we are able to reuse at least a part of the qubits initially needed for the comparisons[3]. The total number of qubits needed for the control of the Hadamard gate thus reads:

$$\begin{aligned}
\mathbf{Q}_{\leq} &= \sum_{j=1}^{M} (|c_j|_2 - 1) + M + \max\left(0, M - 1 - \sum_{j=1}^{M} (|c_j|_2 - 1)\right) \\
&= \sum_{j=1}^{M} |c_j|_2 + \max\left(0, 2M - 1 - \sum_{j=1}^{M} |c_j|_2\right)
\end{aligned}$$

For the additions and subtractions we will employ a strategy to reduce the cycle count, which involves copying the inclusion variable to a number of ancilla qubits dependent on

[3] In practice, we will certainly be able to reuse all qubits, as for most cases $|c_j|_2$ is larger than 1 for all j.

the size of the larger register. More concretely, the corresponding counts are

$$\mathbf{Q}_{\text{COPY}} = \max\left(|P_{\text{UB}}|_2 - \min_i(\text{LSO}(p_i)), \sum_{j=1}^{M}|c_j|_2 - \min_i(\text{LSO}(W_{ij}))\right),$$

where $\text{LSO}(B)$ denotes the index of the least significant one in the binary representation of an integer B. The reason for this expression will become apparent in the upcoming Section 4.3 on the resource requirements of the QFT addition circuit.
Next, the phase oracle $\mathcal{S}_f$ flips the phase of all states in the superposition whose path $\boldsymbol{x}$ is a solution to $f(\boldsymbol{x}) = 1$. We have set f to decide $P(\boldsymbol{x}) > T$, meaning $\mathcal{S}_f$ can be implemented by CNOT gates, which execute the comparison, controlled on the profit register and acting on the single ancilla qubit needed for the phase flip. The multicontrolled gates themselves require $\mathbf{Q}_f = |P_{\text{UB}}| - 1$ ancilla qubits.
Similarly, $\mathcal{S}_0$ produces a phase flip on the state corresponding to $\boldsymbol{x} = \mathbf{0}$, which implies an N-controlled CNOT gate needing $\mathbf{Q}_0 = N - 1$ ancilla qubits.
For the total ancilla count, we are allowed by (v) to take the maximum of all qubit counts which are required for separate processes, instead of the sum. The phase qubit is needed at all times and cannot be absorbed into any other ancilla collection.

$$\begin{aligned}
\mathbf{Q}_{\text{ANCILLA}} &= \max(\mathbf{Q}_{\leq}, \mathbf{Q}_{\text{COPY}}, \mathbf{Q}_f, \mathbf{Q}_0) + 1 \\
&= \max\left(\sum_{j=1}^{M}|c_j|_2 + \max\left(0, 2M - 1 - \sum_{j=1}^{M}|c_j|_2\right), |P_{\text{UB}}|_2 - 1, N - 1\right) + 1 \\
&= \max\left(N, |P_{\text{UB}}|_2, \sum_{j=1}^{M}|c_j|_2 + \max\left(1, 2M - \sum_{j=1}^{M}|c_j|_2\right)\right)
\end{aligned}$$

This leaves us with a total qubit count of:

$$\begin{aligned}
\mathbf{Q} &= \mathbf{Q}_{\text{PATH}} + \mathbf{Q}_{\text{PROFIT}} + \mathbf{Q}_{\text{CAPACITY}} + \mathbf{Q}_{\text{ANCILLA}} \\
&= N + |P_{\text{UB}}|_2 + \sum_{j=1}^{M}|c_j|_2 \\
&\quad + \max\left(N, |P_{\text{UB}}|_2, \sum_{j=1}^{M}|c_j|_2 + \max\left(1, 2M - \sum_{j=1}^{M}|c_j|_2\right)\right)
\end{aligned}$$

For the remaining discussion we will first consider the gate and cycle counts for each subroutine of the QTG separately, combining them afterwards.

4.2 QFT

We realise the quantum Fourier transform as described in Section 2.2.3. From Figure 7 we can deduce a gate count of

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2}$$

for an N-qubit register to which the QFT is applied, which in our case translates to gate counts of

$$\mathbf{G}_{\text{QFT-CAPACITY}} = \frac{1}{2}\sum_{j=1}^{M}|c_j|_2(|c_j|_2 + 1), \quad \mathbf{G}_{\text{QFT-PROFIT}} = \frac{1}{2}|P_{\text{UB}}|_2(|P_{\text{UB}}|_2 + 1)$$

for transforming each of the capacity registers as well as the profit register.
As for the cycle count, the original circuit can be improved upon by executing gates in parallel. The strategy involves shifting all gates acting on a given qubit $|A_{i+1}\rangle$ for $i = 1, ..., N-2$ to the left, until the respective Hadamard gate is applied in parallel with the R_3 gate at the $|A_i\rangle$-level. For $|A_N\rangle$ this is of course not possible, as $|A_{N-1}\rangle$ is not acted on by an R_3 gate. We sketch how this is done below[4]:

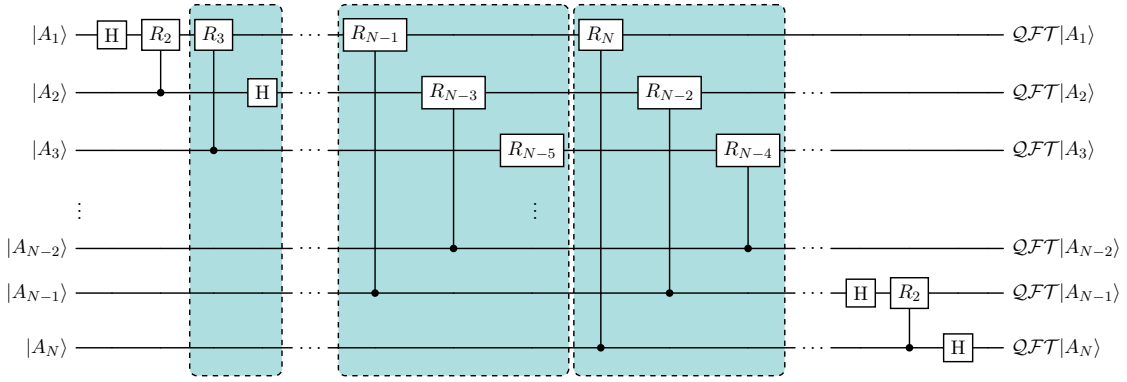

Figure 12: Optimally parallelised QFT circuit. Boxed gates act on or are controlled by separate qubits and can hence be applied simultaneously.

From this, we get that for $i = 1, ..., N-1$ the gate sequence for $|A_{i+1}\rangle$ begins two layers after the start of the sequence for $|A_i\rangle$, meaning we accumulate $2N-1$ layers in total (as the N-th qubit only contributes with one Hadamard gate). Employing this optimisation on the capacity and profit registers we obtain the cycle counts

$$\mathbf{C}_{\text{QFT-CAPACITY}} = \max_j(2|c_j|_2 - 1) = 2\max_j(|c_j|_2) - 1, \quad \mathbf{C}_{\text{QFT-PROFIT}} = 2|P_{\text{UB}}|_2 - 1,$$

where for the capacity registers only the largest cycle count contributes, as the QFT can be executed at the same time on all of them.

4.3 QFT Adder

Before we derive the counts for gates and cycles, we need to introduce a modification to the direct addition circuit described in Section 2.2.3. The QTG calls for the addition $U_i^3 = C^1_{x_i=1}(\text{ADD}(p_i))$ to be controlled on the path variable $|x_i\rangle^1$, this requires all rotation gates to be supplied with a corresponding control. Doing this naively means we have to execute each gate separately, as they are controlled on the same input, however we can again apply a parallelisation strategy. We observe first that, for each qubit in the register, we can bring all gates together to then have their product be controlled on $|x_i\rangle^1$. In another step, we copy this path variable to ancilla qubits, such that the product rotation gates can be controlled on the separately stored copies, allowing for their simultaneous execution. We give as an example the optimised version of the circuit shown in Figure 8:

[4] Adapted from [18].

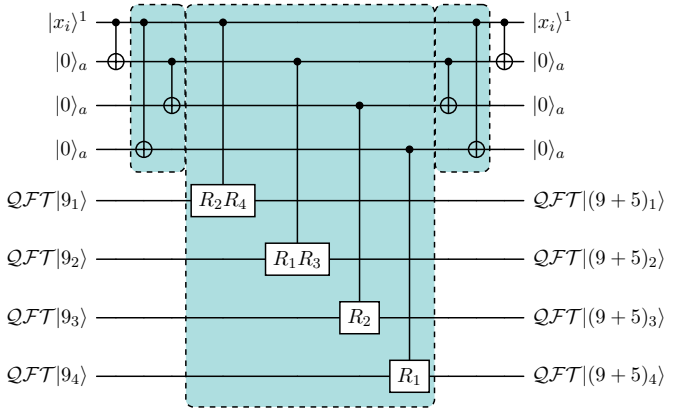

Figure 13: Optimally parallelised QFT adder controlled on the path variable $|x_i\rangle^1$. Again, boxed gates can be run in parallel.

For the gate count, we can see from the example of the uncontrolled circuit in Figure 8 that the least significant one of the classical integer results in the most gates, as the corresponding rotations are applied to each qubit of the register. More precisely, if $\mathrm{LSO}(B)$ denotes the index of the least significant one of B and N is the register size, the number of gates it introduces is $N - \mathrm{LSO}(B) + 1$. By our parallelisation approach, this is actually already the amount of controlled rotation gate products. For their simultaneous execution, we copy the path variable to $N - \mathrm{LSO}(B)$ ancilla qubits using $2(N - \mathrm{LSO}(B))$ CNOT gates, which gives a total gate count of $3(N - \mathrm{LSO}(B)) + 1$. This means

$$\mathbf{G}_{\mathrm{ADD}}(p_i) = 3(|P_{\mathrm{UB}}|_2 - \mathrm{LSO}(p_i)) + 1$$

gates are needed to add p_i to the profit register. From this, we can immediately conclude the number of gates involved for the pure addition subcircuit of the QTG:

$$\mathbf{G}_{\mathrm{ADD}}(\boldsymbol{p}) = \sum_{i=1}^{N} \mathbf{G}_{\mathrm{DIRECT}}(p_i) = \sum_{i=1}^{N} (3(|P_{\mathrm{UB}}|_2 - \mathrm{LSO}(p_i)) + 1).$$

With the parallelisation technique, addition is computed in a single cycle. To copy the path variable onto the ancilla qubits, we add to that a count which is logarithmic in the number of copies needed, achieved by parallel execution of the CNOT gates. The count for one addition reads

$$\mathbf{C}_{\mathrm{ADD}}(p_i) = 2\lceil \log_2(|P_{\mathrm{UB}}|_2 - \mathrm{LSO}(p_i)) \rceil + 1.$$

For all additions within one QTG iteration we then have

$$\mathbf{C}_{\mathrm{ADD}}(\boldsymbol{p}) = \sum_{i=1}^{N} \mathbf{C}_{\mathrm{ADD}}(p_i) = \sum_{i=1}^{N} (2\lceil \log_2(|P_{\mathrm{UB}}|_2 - \mathrm{LSO}(p_i)) \rceil + 1).$$

4.4 QFT Subtractor

As mentioned in the closing remark to Section 2.2.3, subtraction only differs from addition in using the adjoints of the rotation gates, which is why we can carry over most of the

insights gained in the previous Section. As such, the gate count for subtracting the weight W_{ij} of a single item in a single constraint is

$$\mathbf{G}_{\mathrm{SUB}}(W_{ij}) = \mathbf{G}_{\mathrm{ADD}}(W_{ij}) = 3(|c_j|_2 - \mathrm{LSO}(W_{ij})) + 1.$$

We extend this to all constraints for the weight vector $\boldsymbol{W}_i$ of one item by taking the sum over j:

$$\mathbf{G}_{\mathrm{SUB}}(\boldsymbol{W}_i) = \sum_{j=1}^{M} \mathbf{G}_{\mathrm{DIRECT}}(W_{ij}) = \sum_{j=1}^{M} (3(|c_j|_2 - \mathrm{LSO}(W_{ij})) + 1)$$

The gate count for all items can be reduced by removing the subtraction circuit in the final layer, as the registers are not checked afterwards. This results in a total gate number of

$$\mathbf{G}_{\mathrm{SUB}}(W) = \sum_{i=1}^{N-1} \mathbf{G}_{\mathrm{SUB}}(\boldsymbol{W}_i) = \sum_{i=1}^{N-1} \sum_{j=1}^{M} (3(|c_j|_2 - \mathrm{LSO}(W_{ij})) + 1)$$

Instead of carrying out each subtraction separately, we can yet again parallelise the process by first copying the path variable for all subtractions, which we can then do in one go. The cycle count for subtraction of $\boldsymbol{W}_i$ then becomes

$$\mathbf{C}_{\mathrm{SUB}}(\boldsymbol{W}_i) = 2 \left\lceil \log_2 \left(\sum_{i=1}^{M} |c_j|_2 - \mathrm{LSO}(W_{ij}) \right) \right\rceil + 1$$

and thus the cycle count for all subtractions totals to

$$\mathbf{C}_{\mathrm{SUB}}(W) = \sum_{i=1}^{N-1} \mathbf{C}_{\mathrm{SUB}}(\boldsymbol{W}_i) = \sum_{i=1}^{N-1} \left(2 \left\lceil \log_2 \left(\sum_{i=1}^{M} |c_j|_2 - \mathrm{LSO}(W_{ij}) \right) \right\rceil + 1 \right).$$

4.5 Comparison Circuits

There are three processes within one iteration of the QTG which are controlled on comparisons: The biased Hadamard gate $U_i^1 = C_\varphi^1(\mathrm{H}_b^{y_i})$ requires the conjoined results of M comparisons with the remaining capacities, the oracle $\mathcal{S}_f$ applies the phase flip on the condition $P(\boldsymbol{x}) > T$ and $\mathcal{S}_0$ only flips the phase of the state with $\boldsymbol{x} = \mathbf{0}$. We will go into each process one by one.
For the biased Hadamard, we execute M comparisons $W_{ij} \leq C_j$ for each item, whose results are stored on M ancillas on which $H_b^{y_i}$ is finally controlled. To implement the comparisons, we make use of the circuit explained in Section 2.2.2, substituting the general single-controlled gate for a CNOT gate acting on the corresponding result ancilla. There are two possible implementations:

1. Test directly for $W_{ij} - 1 < C_j$ and flip the result qubit accordingly.
2. First flip the result qubit unconditionally, then flip again if $W_{ij} > C_j$.

For all comparisons, we will choose the option which introduces the least amount of gates and cycles.
Each comparison is comprised of a certain number of multicontrolled NOT gates, which

depends on the bits of the considered W_{ij}. For our first implementation strategy, the exact amount is given by

$$\mathbf{G}_{\leq}^{(1)}(W_{ij}) = \sum_{k=1}^{|W_{ij}|_2} (1 - (W_{ij} - 1)_k)(2(|c_j|_2 - k) + 1),$$

where the first factor accounts for the exclusion of those multicontrolled gates where the k-th bit of $(W_{ij}-1)$ is 1, and the second factor corresponds to the number of elementary gates into which each multicontrolled gate decomposes. We observe that this implementation yields less gates for weights which do not produce many ones in the significant bits of $(W_{ij} - 1)$. If, however, this is the case, we rather choose the second option, which has an associated gate count of

$$\mathbf{G}_{\leq}^{(2)}(W_{ij}) = 1 + \sum_{k=1}^{|W_{ij}|_2} (W_{ij})_k(2(|c_j|_2 - k) + 1).$$

Taking the more efficient implementation gives us the number of gates for a single comparison:

$$\mathbf{G}_{\leq}(W_{ij}) = \min\left(\mathbf{G}_{\leq}^{(1)}(W_{ij}), \mathbf{G}_{\leq}^{(2)}(W_{ij})\right)$$

To perform all M comparisons for one item and apply the biased Hadamard controlled on the result qubits, we thus require

$$\mathbf{G}_{\leq}(\boldsymbol{W}_i) = 2\sum_{j=1}^{M}(\mathbf{G}_{\leq}(W_{ij})) + 2(M-1) + 1$$

gates. The factor of 2 in front of the sum stems from the fact that the result ancillas have to be reset after the multicontrolled Hadamard gate has been executed, for this we need to reapply all individual comparisons.
For each multicontrolled gate, we can apply a parallelisation technique which improves the cycle count to be logarithmic in the involved gates when compared to the standard approach, for which the cycles increase linearly[5]. The strategy is outlined in the below example[6]:

[5] See [29], p. 183f.
[6] Adapted from [18].

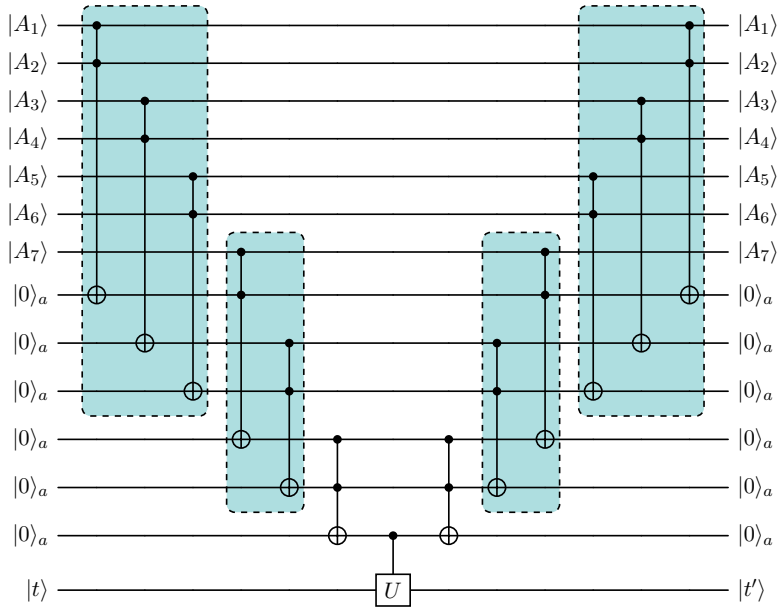

Figure 14: Optimal decomposition of a 7-controlled gate U.

With this decomposition method, we have as the cycle counts for the two comparison options:

$$\mathbf{C}_{\leq}^{(1)}(W_{ij}) = \sum_{k=1}^{|W_{ij}|_2} (1-(W_{ij}-1)_k)(2\lceil \log_2(|c_j|_2 - k)\rceil + 1),$$
$$\mathbf{C}_{\leq}^{(2)}(W_{ij}) = 1 + \sum_{k=1}^{|W_{ij}|_2} (W_{ij})_k(2\lceil \log_2(|c_j|_2 - k)\rceil + 1)$$

We, again, choose for each W_{ij} the lesser of the two.

$$\mathbf{C}_{\leq}(W_{ij}) = \min\left(\mathbf{C}_{\leq}^{(1)}(W_{ij}), \mathbf{C}_{\leq}^{(2)}(W_{ij})\right)$$

When considering the total cycles of a comparison with $\boldsymbol{W}_i$ including the result conjunction, we notice that the individual comparisons occur on separate registers and can thus be run in parallel. This means that only the comparison with the most cycles contributes, which gives us a total cycle count of

$$\mathbf{C}_{\leq}(\boldsymbol{W}_i) = 2\max_j(\mathbf{C}_{\leq}(W_{ij})) + 2\lceil \log_2(M-1)\rceil + 1,$$

where we also applied the optimisation technique to the M-controlled Hadamard.
Next, the phase oracle $\mathcal{S}_f$ controlled on the profit register is constructed in a similar fashion to an individual comparison on a capacity register, only differing in that the condition $P(\boldsymbol{x}) > T$ is now a strict inequality. The gate and cycle counts immediately follow from the discussion above:

$$\mathbf{G}_<(T) = \min\left(\mathbf{G}_<^{(1)}(T), \mathbf{G}_<^{(2)}(T)\right), \text{ where}$$
$$\mathbf{G}_<^{(1)}(T) = \sum_{k=1}^{|T|_2}(1-(T)_k)(2(|P_{\mathrm{UB}}|_2-k)+1),$$
$$\mathbf{G}_<^{(2)}(T) = 1+\sum_{k=1}^{|T|_2}(T+1)_k(2(|P_{\mathrm{UB}}|_2-k)+1).$$

$$\mathbf{C}_<(T) = \min\left(\mathbf{C}_<^{(1)}(T), \mathbf{C}_<^{(2)}(T)\right), \text{ where}$$
$$\mathbf{C}_\leq^{(1)}(T) = \sum_{k=1}^{|T|_2}(1-(T)_k)(2\lceil\log_2(|P_{\mathrm{UB}}|_2-k)\rceil+1),$$
$$\mathbf{C}_\leq^{(1)}(T) = 1+\sum_{k=1}^{|T|_2}(T+1)_k(2\lceil\log_2(|P_{\mathrm{UB}}|_2-k)\rceil+1).$$

Finally, the other phase oracle $\mathcal{S}_0$ is realised by a single N-controlled NOT gate applied to the phase ancilla only if the stored state is $|\mathbf{0}\rangle^1$. The decomposition of this gate gives a gate count of

$$\mathbf{G}_=(\mathbf{0}) = 2(N-1)+1 = 2N-1$$

and an optimal cycle count of

$$\mathbf{C}_=(\mathbf{0}) = 2\lceil\log_2(N-1)\rceil+1.$$

4.6 QTG

Using the results for all of its subcircuits, we now dedicate ourselves to the total amounts of gates and cycles needed for implementing the QTG $\mathcal{G}$. As previously mentioned, we leave out the final subtraction U_N^2 on the capacity registers to reduce the counts. Another optimisation we can employ is in the creation of the path variable copies for subtraction and addition: We produce only as many as are needed by the operation which demands the most copies. In practice, this means taking the maximum of the gate counts for copying in both methods instead of their sum.
One more improvement measure involves the notion of a break item. This is understood to be the first item which violates a constraint when we consecutively add all items. Formally, we can write it as

$$i_{\mathrm{B}} = \min\left\{ i \in \{1,...,N\} \middle| \exists j \in \{1,...,M\} : \sum_{k=1}^{i} W_{kj} > c_j \right\}.$$

We obtain this number via an efficient classical preprocessing step, simply by including each item until a constraint is violated. Knowledge of this item is useful, as we are not required to do any comparisons on the capacity registers until its corresponding layer, implying that the biased Hadamard gates can be applied unconditionally for the first $i_{\mathrm{B}}-1$ layers.
Lastly, we consider the number of times the QFT has to be applied to the profit and capacity registers. For the prior, we notice that the profit is not checked on until the end of one QTG iteration, when it serves as the control for the phase oracle $\mathcal{S}_f$. We can thus execute the QFT once in the beginning and its inverse also once after all profits have been added. The capacity registers, on the other hand, are checked in each layer starting with the break item. This means we have to transform them once before the weights of the first item are subtracted, apply its inverse before the break item is considered and then alternate between QFT and inverse QFT in each layer beginning with the break item up until the $N-1$-st layer, as we skip the last subtraction.

With these optimisations, the total gate count for one iteration of the QTG becomes:

$$\begin{aligned}\mathbf{G}_{\text{QTG}} = (i_{\text{B}} - 1) &+ \sum_{i=i_{\text{B}}}^{N} \mathbf{G}_{\leq}(\boldsymbol{W}_i) + 2\mathbf{G}_{\text{QFT-PROFIT}} + 2(N - i_{\text{B}})\mathbf{G}_{\text{QFT-CAPACITY}} \\ &+ 2\sum_{i=1}^{N-1}\left(\max\left(|P_{\text{UB}}|_2, \sum_{j=1}^{M}|c_j|_2\right) - \min\left(\text{LSO}(p_i), \sum_{j=1}^{M}\text{LSO}(W_{ij})\right)\right) \\ &+ \sum_{i=1}^{N-1}\left(|P_{\text{UB}}|_2 - \text{LSO}(p_i) + 1 + \sum_{j=1}^{M}(|c_j|_2 - \text{LSO}(W_{ij}) + 1)\right) \\ &+ 3(|P_{\text{UB}}|_2 - \text{LSO}(p_N)) + 1.\end{aligned}$$

The first term accounts for the unconditioned Hadamard gates, the following three terms of the expression are self-explanatory. The fifth term corresponds to the gates from the optimised copy strategy, while the sixth contains the gate count for all but the last addition and subtraction operations. The final addition including copying of the path variable is represented by the last term.
For the total cycle count of the QTG, we observe that we must execute each unitary layer $U_i = U_i^1 U_i^2 U_i^3$ separately since the addition and subtraction are controlled on the state $|x_i\rangle^1$ of the respective path variable, which itself depends on the comparison between the remaining capacities and the weights of the item currently considered. We obtain the total cycle count of the QTG as the sum of cycles $\mathbf{C}_i$ in each individual layer:

$$\mathbf{C}_{\text{QTG}} = \sum_{i=1}^{N} \mathbf{C}_i.$$

That being said, we are still able to reduce the cycle counts within the layers. For this, we first notice that copying the path variable to the needed amount of ancilla qubits as well as resetting said qubits can be done in parallel with the QFT and the inverse QFT on the capacity registers, respectively. To this we add that, under a certain condition on the number of constraints M, the cycle count of the QFT is able to accommodate that of the copying procedure. This follows from the fact that the former is linear in the bit length of the largest capacity $c_{j'} := \max_j(c_j)$, while the latter is logarithmic in the profit length or the sum of all capacity lengths (minus the LSO's for each p_i, W_{ij}). In the case where the profit requires more copy ancillas, we have

$$\mathbf{C}_{\text{COPY}}(p_i) = \lceil \log_2(|P_{\text{UB}}|_2 - \text{LSO}(p_i)) \rceil < \lceil \log_2 |P_{\text{UB}}|_2 \rceil < 2|P_{\text{UB}}|_2 - 1 = \mathbf{C}_{\text{QFT-PROFIT}},$$

which is always fulfilled. For the other case, we can derive the condition on M by considering

$$\begin{aligned}\mathbf{C}_{\text{COPY}}(\boldsymbol{W}_i) &= \left\lceil \log_2\left(\sum_{j=1}^{M}|c_j|_2 - \text{LSO}(W_{ij})\right)\right\rceil < \left\lceil \log_2\left(\sum_{j=1}^{M}|c_j|_2\right)\right\rceil \\ &< \lceil \log_2(M|c_{j'}|_2) \rceil \leq \lceil \log_2 M \rceil + \lceil \log_2 |c_{j'}|_2 \rceil.\end{aligned}$$

If the cycle count for the QFT is to be larger than that for copying to the ancillas, we now impose it to also be greater than this upper bound:

$$\begin{aligned}\mathbf{C}_{\text{COPY}}(\boldsymbol{W}_i) < \lceil \log_2 M \rceil + \lceil \log_2 |c_{j'}|_2 \rceil &\overset{!}{<} 2|c_{j'}|_2 - 1 = \mathbf{C}_{\text{QFT-CAPACITY}} \\ &\Rightarrow \log_2 M < 2|c_{j'}|_2 - \log_2 |c_{j'}|_2 - 1.\end{aligned}$$

Exponentiating this expression yields the final form of the condition:

$$M < \frac{2^{2|c_{j'}|_2-1}}{|c_{j'}|_2} = \frac{2^{2\lfloor \log_2(c_{j'})\rfloor+1}}{\lfloor \log_2(c_{j'})\rfloor+1}.$$

As the bound is essentially quadratic in $c_{j'}$, this would only be violated for high-dimensional problems which involve capacities of only a few digits. As this is not the case for realistic problem instances, we will fit the cycle count for copying the path variable into that of the (inverse) QFT. Lastly, we remark that the QFT on the profit register can be executed simultaneously with any gates running on the capacity registers, since it does not depend on any control.
We are now ready to consider the cycle count for each layer of the QTG individually. For the first layer, we simultaneously apply the QFT to the profit and capacity registers $\mathcal{H}_1, \mathcal{H}_2$, meaning the longer of the two contributes. Besides that, we can be sure that the first item cannot be the break item and thus needs no comparison to be inserted - otherwise it would not contribute to the problem at all. Hence, the biased Hadamard gate is executed with no condition, in parallel with the transformations. As noted above, we also execute the copying procedure onto the ancillas for the addition of p_1 and the subtractions of W_{1j} in tandem with the QFT on $\mathcal{H}_2$. The single rotation cycle for one of the operations cannot be accommodated anywhere, as it has to be inserted after the longer transformation has been completed, the same applies to the reset of the ancillas. These considerations give an initial count of

$$\mathbf{C}_1^{(1)} = \max(\mathbf{C}_{\text{QFT-CAPACITY}}, \mathbf{C}_{\text{QFT-PROFIT}}) + 1 + \max(\mathbf{C}_{\text{COPY}}(p_1), \mathbf{C}_{\text{COPY}}(\boldsymbol{W}_1)).$$

Now, if the break item is $i_\text{B} = 2$, then the transformation on $\mathcal{H}_2$ has to be immediately reverted. We can fit the leftover rotation into its cycle count, together with the circuit for uncomputing ancilla register. In this case we obtain a count of

$$\mathbf{C}_1^{(2)} = \max(\mathbf{C}_{\text{QFT-CAPACITY}}, \mathbf{C}_{\text{QFT-PROFIT}}) + \mathbf{C}_{\text{QFT-CAPACITY}}.$$

The cycle count for these two cases may be jointly written as:

$$\begin{aligned}\mathbf{C}_1 =& \max(\mathbf{C}_{\text{QFT-CAPACITY}}, \mathbf{C}_{\text{QFT-PROFIT}}) + 1 + \max(\mathbf{C}_{\text{COPY}}(p_1), \mathbf{C}_{\text{COPY}}(\boldsymbol{W}_1)) \\ &+ \delta_{2i_\text{B}}(\mathbf{C}_{\text{QFT-CAPACITY}} - 1 - \max(\mathbf{C}_{\text{COPY}}(p_1), \mathbf{C}_{\text{COPY}}(\boldsymbol{W}_1))).\end{aligned}$$

For the case $i_\text{B} > 2$, we apply the Hadamard gate and add the profits and subtract the weights of the items $1 < i < i_\text{B}$ before the break item (BB). As before, we have to undo the QFT in the $(i_\text{B} - 1)$-st layer, parallel to which we can execute the rotation gates as well as the ancilla reset. We again have two cases,

$$\begin{aligned}\mathbf{C}_i^{\text{BB}(1)} &= 1 + \max(\mathbf{C}_{\text{ADD}}(p_i), \mathbf{C}_{\text{SUB}}(\boldsymbol{W}_i)) = 2 + 2\max(\mathbf{C}_{\text{COPY}}(p_i), \mathbf{C}_{\text{COPY}}(\boldsymbol{W}_i)), \\ \mathbf{C}_i^{\text{BB}(2)} &= 1 + \mathbf{C}_{\text{QFT-CAPACITY}} + \max(\mathbf{C}_{\text{COPY}}(p_i), \mathbf{C}_{\text{COPY}}(\boldsymbol{W}_i)),\end{aligned}$$

which we may both address with a single expression:

$$\begin{aligned}\mathbf{C}_i^{\text{BB}} =& 2 + 2\max(\mathbf{C}_{\text{COPY}}(p_i), \mathbf{C}_{\text{COPY}}(\boldsymbol{W}_i)) \\ &+ \delta_{i(i_\text{B}-1)}(\mathbf{C}_{\text{QFT-CAPACITY}} - 1 - \max(\mathbf{C}_{\text{COPY}}(p_i), \mathbf{C}_{\text{COPY}}(\boldsymbol{W}_i)))\end{aligned}$$

Starting with the break item (SB) we introduce the comparison cycles into each layer, always including the QFT and its inverse which overlap with the ancilla transformations

and the rotation on the profit register. The rotation on the capacity registers stands alone. This gives a count of

$$\mathbf{C}_i^{\mathrm{SB}} = \mathbf{C}_{\leq}(\boldsymbol{W}_i) + 2\mathbf{C}_{\text{QFT-CAPACITY}} + 1.$$

Finally, the last layer lacks any operations on $\mathcal{H}_2$, leaving only one comparison, the inverse QFT on $\mathcal{H}_3$ and a partly parallelised addition of p_N. The associated cycle count is

$$\mathbf{C}_N = \mathbf{C}_{\leq}(\boldsymbol{W}_N) + \mathbf{C}_{\text{QFT-PROFIT}} + \mathbf{C}_{\text{COPY}}(p_N) + 1.$$

All together, the cycle count of the QTG amounts to

$$\mathbf{C}_{\mathrm{QTG}} = \mathbf{C}_1 + \sum_{i=2}^{i_{\mathrm{B}}-1} \mathbf{C}_i^{\mathrm{BB}} + \sum_{i=i_{\mathrm{B}}}^{N-1} \mathbf{C}_i^{\mathrm{SB}} + \mathbf{C}_N.$$

4.7 QMaxSearch

While we were thus far able to precisely determine the counts of the individual components of $\mathcal{G}$ in order to compose them into the total number of gates and cycles, the probabilistic nature of the **QSearch** routine 2 prohibits us from deriving a similar absolute result for **QMaxSearch**, meaning we will have to rely on its simulation to infer the required resources. Nevertheless, we may still reason about the counts $\mathbf{G}_{\mathrm{QMaxSearch}}$ and $\mathbf{C}_{\mathrm{QMaxSearch}}$ by considering the sequence $(T_k)_{k=0}^{R}$ of profit thresholds which occur within one run of **QMaxSearch**. We recall that the first threshold T_0 is obtained by a classical greedy algorithm, it is then subsequently updated until a final profit T_R is achieved, which may be optimal. Each T_k is associated to a corresponding call of **QSearch**, determining the decision function f to check the comparison $P(\boldsymbol{x}) > T_k$. Within such a call, there is a series $(j_l^{(k)})_{l=1}^{S(k)}$ of powers for the Amplitude Amplification operator $\mathcal{Q} = \mathcal{G}\mathcal{S}_0\mathcal{G}^\dagger\mathcal{S}_f$, where $S(k)$ denotes the index of the last power applied, which are chosen at random. For $k < R$, application of the power $j_{S(k)}^{(k)}$ results in a solution to f being found at the end of the k-th **QSearch** run. In the last **QSearch**, however, we can be sure that the maximum iteration number M_{iter} has been reached, and so the sequence of powers for $k = R$ suffices the condition

$$\sum_{l=1}^{S(R)-1} 2j_l^{(R)} + 1 < M_{\mathrm{iter}} \leq \sum_{l=1}^{S(R)} 2j_l^{(R)} + 1.$$

We can now first consider the gates needed for the k-th run of **QSearch**. Each of its $S(k)$ iterations consists of one application of the QTG to create the initial superposition, followed by $j_l^{(k)}$-fold application of the Amplitude Amplification operator $\mathcal{Q}_k$ for the corresponding profit threshold T_k, which presents the following gate count:

$$\mathbf{G}_{\mathcal{Q}_k} = 2\mathbf{G}_{\mathrm{QTG}} + \mathbf{G}_{=}(\mathbf{0}) + \mathbf{G}_{<}(T_k).$$

This then implies the k-th **QSearch** requires an amount of gates equal to

$$\mathbf{G}_{\mathrm{QSearch}_k} = \sum_{l=1}^{S(k)} (2j_l^{(k)} + 1)\mathbf{G}_{\mathcal{Q}_k} + j_l^{(k)}(\mathbf{G}_{=}(\mathbf{0}) + \mathbf{G}_{<}(T_k)), \tag{3}$$

giving the final gate count for **QMaxSearch**:

$$\mathbf{G}_{\mathrm{QMaxSearch}} = \sum_{k=1}^{R} \mathbf{G}_{\mathrm{QSearch}_k}.$$

For the cycle count, there is no optimisation possible between the components of $\mathcal{Q}_k$, as $\mathcal{S}_0$ and $\mathcal{S}_f$ require access to the profit register which is modified by $\mathcal{G}$. Hence, the cycle count for **QSearch** is

$$\mathbf{C}_{\mathrm{QSearch}_k} = \sum_{l=1}^{S(k)} (2j_l^{(k)} + 1)\mathbf{C}_{\mathrm{QTG}} + j_l^{(k)}(\mathbf{C}_{=}(\mathbf{0}) + \mathbf{C}_{<}(T_k)) \tag{4}$$

and, ultimately, the one for **QMaxSearch** turns out to be

$$\mathbf{C}_{\mathrm{QMaxSearch}} = \sum_{k=1}^{R} \mathbf{C}_{\mathrm{QSearch}_k}.$$

These equations will be applied as part of the simulated QTG in the following Section.

5 Simulation and Evaluation

The high-level simulator for the QTG proposed in [18] is now modified to allow for the computation of the resources needed by **QMaxSearch** in the more general setting of the 0-1 MDKP. As the generalisation is fairly straight-forward, we follow the proposal in describing the simulation strategy. To give some context on the classical counterpart, we also present a brief overview of the approach used by CPLEX for solving the 0-1 MDKP. Lastly, we look at the problem instances used to compare the two methods in the following Section.

5.1 Simulation Strategy

In order to simulate the workings of **QMaxSearch**, we must iteratively simulate the action of **QSearch**, which we may describe as:

1. Produce a number of states whose profit exceeds the given threshold T.
2. Attach to each of these states the corresponding measurement probability as given by the Amplitude Amplification procedure.
3. Obtain $\boldsymbol{x}, P(\boldsymbol{x})$ as result of a measurement and return both if $P(\boldsymbol{x}) > T$.

The main idea of how to accomplish this is the following: We generate the final states for a given problem instance by considering the binary tree graph rooted at the initial state with $\boldsymbol{x} = \mathbf{0}, \boldsymbol{C}(\boldsymbol{x}) = \boldsymbol{c}, P(\boldsymbol{x}) = 0$ (compare with the example in Figure 11). Now, if we were to take into account all possible 2^N states, this would explode the memory requirements of the simulation. Instead, we immediately discard infeasible states when branching out from the root, as is done by the QTG itself. For the simulation, we additionally disregard states which do not admit total profits larger than the threshold T. We achieve this for each height i of the tree by employing CPLEX to solve the subproblems involving the last $N - i$ items with the remaining capacities of each node at that height, from which we get an upper bound on the obtainable profit within the respective subtrees. This drastically reduces the amount of states we are dealing with, making it possible to simulate larger instances.
For the branching process, we may consider a state comprising the superposition achieved after application of the i-th layer unitary to the initial state $|\mathbf{0}\rangle^1 \otimes |\boldsymbol{c}\rangle^2 \otimes |0\rangle^3$, which corresponds to a node at height i of the binary tree. Assume the node has remaining

capacities $\tilde{\boldsymbol{C}}$, a total profit of $\tilde{P}$ and a sampling probability of $\tilde{\pi}$, and the profit associated to the solution obtained by CPLEX is $\tilde{P}_{N-i}$. This node then branches out into two child nodes at height $i+1$ only if $\boldsymbol{W}_{i+1} \leq \tilde{\boldsymbol{C}}$ and $\tilde{P} + \tilde{P}_{N-i} > T$. The children then hold the following values:

$$
\begin{aligned}
(\tilde{\boldsymbol{C}}_{\text{left}}, \tilde{\boldsymbol{C}}_{\text{right}}) &= (\tilde{\boldsymbol{C}}, \tilde{\boldsymbol{C}} - \boldsymbol{W}_{i+1}) \\
(\tilde{P}_{\text{left}}, \tilde{P}_{\text{right}}) &= (\tilde{P}, \tilde{P} + p_{i+1}) \\
(\tilde{\pi}_{\text{left}}, \tilde{\pi}_{\text{right}}) &= \begin{cases} \left(\frac{\tilde{\pi}(b+1)}{b+2}, \frac{\tilde{\pi}}{b+2}\right), & \text{if } y_{i+1} = 0 \\ \left(\frac{\tilde{\pi}}{b+2}, \frac{\tilde{\pi}(b+1)}{b+2}\right), & \text{else.} \end{cases}
\end{aligned}
$$

The probabilities are propagated as imposed by the biased Hadamard gate $\mathrm{H}_b^{y_{i+1}}$, where y_{i+1} stems from the path $\boldsymbol{y}$ which constitutes the current threshold profit (compare with Equation (2)). If instead $\boldsymbol{W}_{i+1} > \tilde{\boldsymbol{C}}$ or $\tilde{P} + \tilde{P}_{N-i} \leq T$, the parent node only produces a single child with inherited parameters.
After the branching process, we are left with an array of states which are all feasible and hold a profit larger than T. Next, the probability associated to each state is altered in accordance with Amplitude Amplification. We recall Equation (1), by which the j-fold application of the corresponding operator $\mathcal{Q}$ to the initial superposition $|\Psi_{\mathcal{G}}\rangle$ yields

$$
\mathcal{Q}^j|\Psi_{\mathcal{G}}\rangle = \frac{\sin\left((2j+1)\arcsin(\sqrt{\pi(|\mathcal{L}\rangle)})\right)}{\sqrt{\pi(|\mathcal{L}\rangle)}}|\mathcal{L}\rangle + \frac{\cos\left((2j+1)\arcsin(\sqrt{\pi(|\mathcal{L}\rangle)})\right)}{\sqrt{1-\pi(|\mathcal{L}\rangle)}}|\mathcal{F}\setminus\mathcal{L}\rangle,
$$

where $|\mathcal{L}\rangle$ consists exactly of the branched states with solution paths $\boldsymbol{x} \in \mathcal{L}$. The effect of Amplitude Amplification is thus the multiplication of all probabilities with a common factor of

$$
\frac{\sin^2\left((2j+1)\arcsin(\sqrt{\pi(|\mathcal{L}\rangle)})\right)}{\pi(|\mathcal{L}\rangle)}.
$$

For the last part of measuring the states, we begin with a random number $r \in [0,1]$ and sum up the probabilities of all states making up $|\mathcal{L}\rangle$ one by one. If the sum becomes larger than r, the state whose probability was the last summand is declared the measurement result. Otherwise, if the sum of all probabilities falls short of r, we consider the measurement to have produced a state with a path in $\mathcal{F}\setminus\mathcal{L}$. We do not need to know what exact state would have been measured, as the **QSearch** procedure ignores states for which $P(\boldsymbol{x}) < T$ anyway, simply starting the next iteration with another value for j.
Using these three steps, we can simulate **QSearch** by iterating over random values of j as described in the **QSearch** pseudocode 2, either until $P(\boldsymbol{x}) > T$ is fulfilled for the last measured state or the maximum number of iterations M_{iter} is reached. This is then made part of the **QMaxSearch** routine by repetition over the threshold profit T. With each call of **QSearch**, we keep track of the gates and cycles which would have been used by the real quantum processor by accumulating their counts for the various internal iterations j, as demanded by Equation (3) and Equation (4). From this we can then conclude the total resource consumption of **QMaxSearch**.

5.2 CPLEX

On the classical side of things, solving the 0-1 MDKP using CPLEX is done via its framework for Mixed Integer Programming (MIP) problems, which may include both discrete

as well as continuous variables. The approach pursued by CPLEX for the solution of these problems is that of branch-and-cut[7]: First, the problem is relaxed by dropping the integrality condition on the discrete variables. In the case of 0-1 MDKP, this corresponds to changing the values taken by the path variables from $x_i \in \{0, 1\}$ to $x_i \in [0, 1]$, thus allowing the fractional inclusion of an item. Solving this continuous relaxation is generally much easier than its discrete counterpart, however the solutions obtained in each case will not match in general. This is easy to see in the example shown in Figure 1: In the continuous case, the entire unit square is mapped to the parallelogram, thus giving its intersection with the feasible region as the space in which to look for an optimal solution. Here, we can further increase the profit of the discrete solution $\boldsymbol{x}_2 = (1, 0)$ by simply introducing a fractional second component, and so the solutions differ.
Starting with the continuously relaxed problem and its solution, CPLEX then employs methods to cut away parts of the feasible region which contain fractional variables. If one of these variables still persists after all cuts have been made, two new subproblems are generated in which the variable is further constrained. For the binary variables $x_i \in \{0, 1\}$ ocurring in 0-1 MDKP, this means the corresponding fractional variable is fixed to 0 and 1 in the respective subproblems. This can be viewed as the creation of a tree graph whose root node is the continuous relaxation of the problem, with each child node being a more restricted subproblem. Branching of a node only occurs if the cutting procedure leaves another variable fractional, otherwise the node has only integer variables and presents a possible solution, or it is infeasible.
In contrast to the QTG, of course, the nodes have to be judged individually, which is why CPLEX makes use of heuristics, backtracking and many other tools on the way to finding an optimal solution to the given problem. The search behaviour is very tweakable, allowing the user to customize how CPLEX finds solutions based on their needs. One parameter relevant for us is the relative MIP gap tolerance, using which CPLEX decides on whether a solution is of sufficient quality to be put out as the result. In our case, it essentially gives the bound relative to the best profit known by CPLEX to be achievable, within which the profit of a solved subproblem node can fall to be considered optimal. By default, this is set to 10^{-4}, however we will change it to 0 so as to make sure that we really are given the optimal solution.

5.3 Problem Instances

Prior to benchmarking the performances of the simulated **QMaxSearch** and CPLEX, we introduce the problem instances they will be tasked to solve. Similar to [18], we also rely on randomly generated instances, which we create following a procedure first proposed by Fréville and Plateau[34]. The base parameters N and M are supplemented by a tightness ratio $\alpha \in (0, 1)$, which can be thought of as restricting each capacity relative to the amount which accommodates all item weights with respect to the corresponding resource. Altering this value affects the hardness of the resulting instance, as looser problems with larger α admit better greedy solutions than tighter ones, giving a good starting point for branching algorithms like CPLEX or QTG. Now, given N, M and α, a problem instance is generated by:

1. Drawing the weights W_{ij} uniformly from $\{0, 1, ..., 1000\}$ at random.

2. Calculating the capacities as $c_j = \lfloor \alpha \sum_{i=1}^{N} W_{ij} \rfloor$.

[7] Compare with the entries *"Invoking the optimizer for a MIP model."* and *"Branch & cut or dynamic search?"* in the CPLEX documentation on discrete optimisation[33].

3. Drawing random p_i uniformly from $\{1, 2, ..., 1000\}$.

The original proposal also featured the option to correlate the profits to the weights; we impose no such correlation on the random instances we generate, instead only varying the values of N, M, α in order to determine their effect on the computational resources used by each algorithm.

6 Numerical Experiments

Finally, we seek to compare the resource usages of the QTG algorithm and CPLEX, assuming the comparability of the cycles and qubits/bits needed by each method. To this end, the high-level simulator detailed in the previous Section was implemented in the C programming language, along with a solving routine making use of the Callable Library provided by IBM ILOG CPLEX Optimization Studio version 22.1.1. For the resources required by CPLEX, the assembly instruction `rdtsc` was used to mark the start and the end of the `CPXmipopt` routine which calculates an optimal solution, in order to obtain the needed CPU cycles, and the Memory Supervision System tool[35] was introduced, to accurately determine the amount of memory required. The algorithms were executed on a 3.6 GHz AMD Ryzen 5 3600 CPU with 16 GB 3000 MHz RAM.
For calculations of the qubit, gate and cycle counts, we implemented the method of Volgenant and Zoon[23] to determine an upper bound P_{UB} on the total profit. In order to obtain the first profit threshold T_0 for **QMaxSearch**, we unfortunately have no clear choice of a greedy algorithm, as is the case for the 0-1 KP. As the quality of the lower bound greatly impacts the problem sizes N we can feasibly simulate, we turn to CPLEX here as well, setting the relative MIP gap tolerance to the maximal value of 1 so as to terminate the solution search process early. This produces a good lower bound on the total profit, which is then used as T_0. Note that this technically introduces cycles on top of those required by **QMaxSearch**; we neglect them, arguing that the approximate CPLEX routine should be considered as a substitute for an actual efficient greedy heuristic whose cycle count is negligible.
To briefly introduce the methodology used: As the simulated **QMaxSearch** algorithm is, at its quantum heart, probabilistic in nature, its performance cannot be judged from a single execution. Instead, we average the results of 100 simulated measurements to obtain the required resources for solving a given problem, and estimate the cycles elapsed during the runtime of CPLEX in a similar fashion. We find, however, that for most examined problem classes the use of a single value of $M_{\mathrm{iter}} = 100$ or 200, as was done in [18], only produces an optimal profit in those cases where it is already given as the initial threshold by the relaxed CPLEX routine. As the maximum allowed number of Grover iterations within the **QSearch** subroutine is directly connected to the probability of measuring a favourable state, we instead apply **QMaxSearch** to each problem instance with varying M_{iter} and determine a reasonable estimate on the cycles and gates needed to amount a substantial probability of measuring the optimal solution to a given problem based on the minimal value of M_{iter} associated to at least one success among the **QMaxSearch** runs.

6.1 Performance with Respect to Problem Size N

In the first series of problems, we take values of N between 200 and 1800 at low dimensions M, choosing easy instances with $\alpha \in \{0.89, 0.9, 0.91\}$. For **QMaxSearch**, we fix the bias at $b = \frac{N}{4}$ and consider different numbers of maximum iterations M_{iter} between 200 and 4000.

6.1.1 One-Dimensional Problem Instances

Setting $M = 1$, we investigate the special case of 0-1 KP in order to compare with the findings in [18].

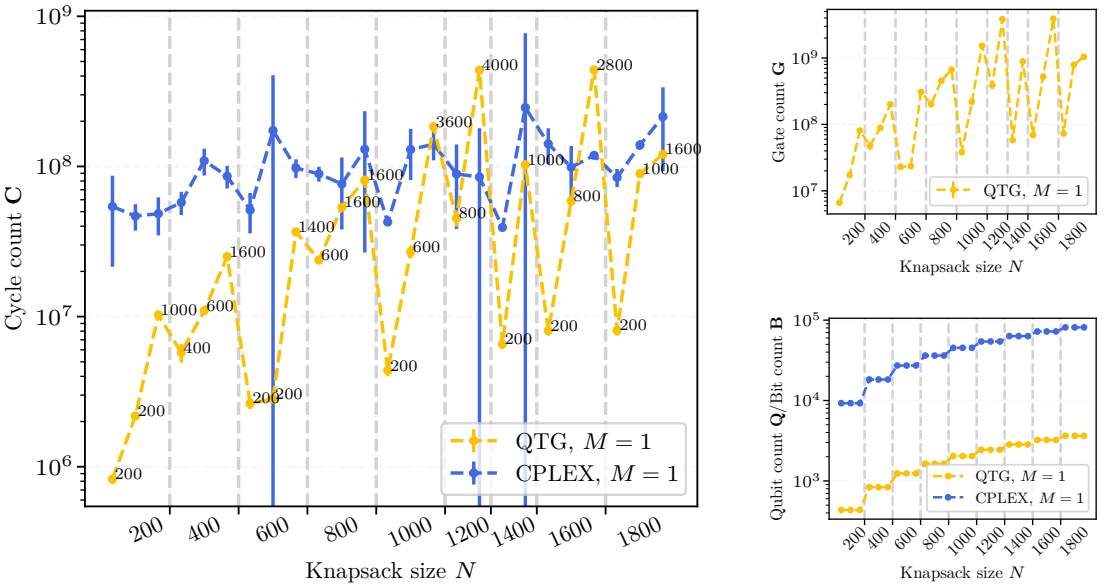

Figure 15: Resource requirements of QTG-based search and CPLEX for one-dimensional problem instances. **a)** Respectively, the cycle counts are depicted in yellow and blue with corresponding error bars. Several runs of CPLEX showed outliers with cycles approximately ten times larger than the respective mean, explaining the enormous errors. Next to each data point for QTG is the minimal M_{iter} for the measurement of an optimal item assignment among 100 iterations of **QMaxSearch**. Two data points at $N = 1200$ and $N = 1400$ have been surpressed, as no optimal measurement occured even at $M_{\text{iter}} = 4000$. **b)** The gate count of QTG closely follows the cycle count, albeit with a difference of a factor of ten. **c)** QTG requires about an order of magnitude less memory than CPLEX.

We notice that, interestingly enough, the QTG algorithm actually dominates CPLEX in the realm of $N \lesssim 800$, with the largest difference in cycle counts amounting to more than a factor of ten. This behaviour is inverse to that observed by Wilkening et al. in their benchmarking of the QTG against the COMBO algorithm, where the quantum method initially has a higher cycle count, but eventually overtakes at $N = 600$. In our case, the cycles of QTG starting at $N = 800$ instead remain in the same order of magnitude as those of CPLEX, surpassing them at several points. The discrepancy may stem from the different approaches used to generate problem instances; this is supported by the fact that the specialised COMBO shows[8] a steady increase in cycles up to beyond 10^9, while the all-rounder CPLEX barely crosses 10^8. It appears that the QTG does not scale well with problems generated using the procedure described in Section 5.3.
In terms of memory, we notice that the requirements of qubits/bits are separated by an order of magnitude, which is to be expected seeing as the single superposition produced by the QTG holds the full information which is distributed across the various subproblem

[8] Compare [18], Figure 3.

nodes stored by CPLEX. In this regard we obtain a similar result to [18], albeit the gap between the requirements of QTG and COMBO found there is larger, spanning three orders of magnitude.

6.1.2 Multidimensional Problem Instances

To follow up, we now increase the dimension of the problem instances to $M = 5$. We are confronted with the need to increase the values of M_{iter} even further to achieve results, and allow them now to range up to 8000.

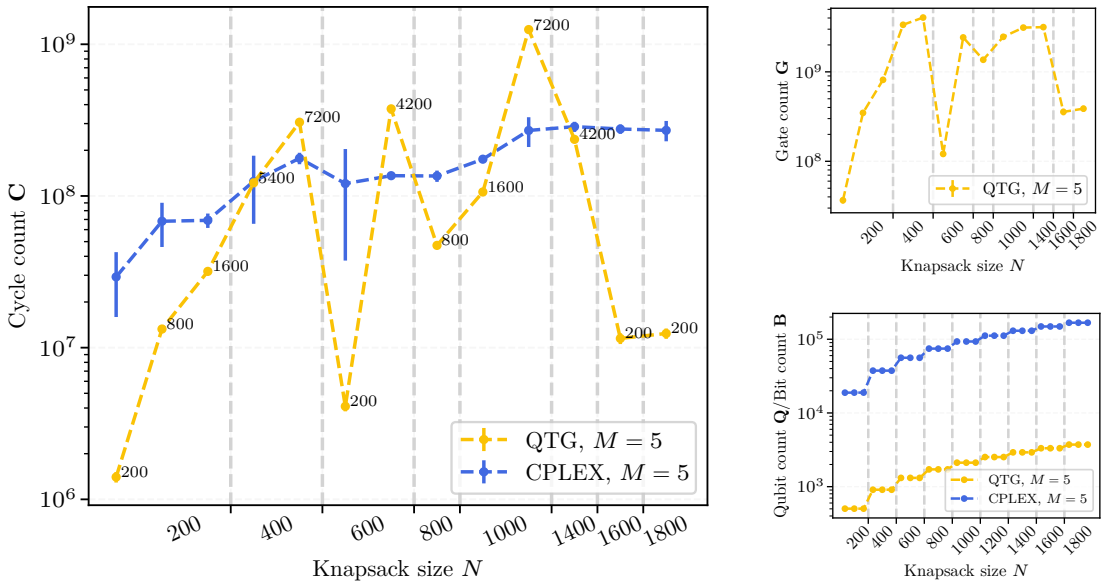

Figure 16: Resource requirements of QTG-based search and CPLEX for problem instances with $M = 5$. **a)** The cases in which **QMaxSearch** has given out an optimal result are even more infrequent, nevermind the higher ceiling of allowed Grover iterations. **b)** Again, the gate count mirrors the amount of elapsed quantum cycles. **c)** Both methods require more memory compared to the one-dimensional cases, however the relative gap is slightly larger.

The intersection of the two cycle counts has shifted to the left; the performance of CPLEX matches that of QTG already at $N = 400$. At the same time, the probability of **QMaxSearch** to produce an optimal solution has been reduced, an effect which we were only partially able to compensate with values of M_{iter} in the given range.
Nevertheless, the advantage in the realm of low N is still present, and we wish to focus on it next. For this, we limit the range to $N \in \{25, ..., 250\}$ and fix $\alpha = 0.9$, but also allow M to take values in $\{5, 10, 15, 20\}$. The number of Grover iterations is now again at most $M_{\text{iter}} = 4000$, larger values only improve already present, reasonable probabilities of success.

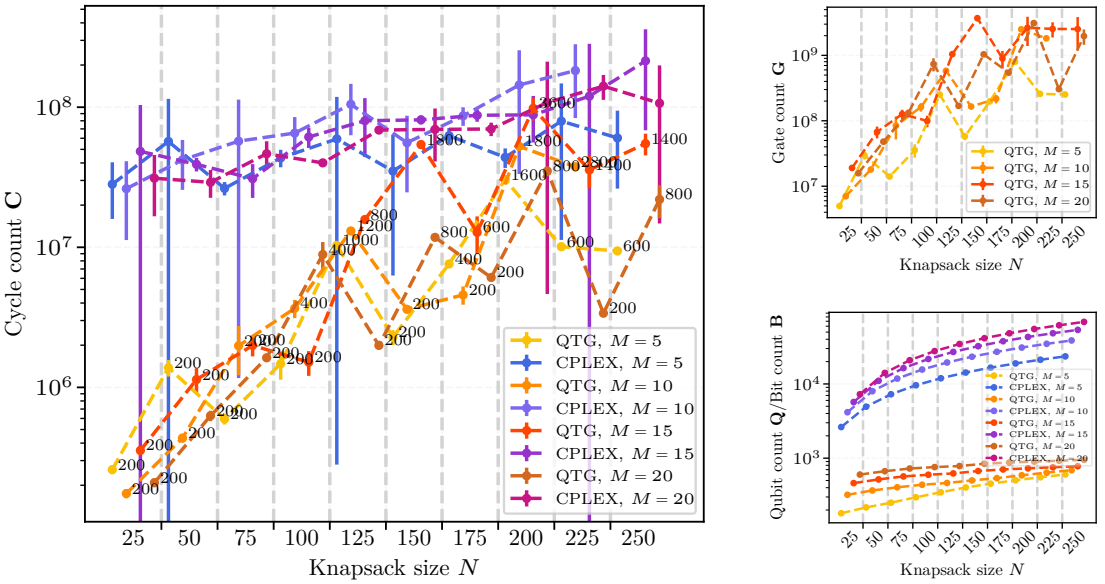

Figure 17: Resource requirements of QTG-based search and CPLEX for problem instances with $M \in \{5, 10, 15, 20\}$. **a)** The cycle count of QTG remains mostly below that of CPLEX for all dimensions. Noticeable is also that relatively low values of M_{iter} suffice to give a good likelihood of an optimal solution being measured, with the problem corresponding to $N = 250, M = 10$ marking the only exception (as it is not depicted). **b)** The required amounts of gates do not directly show an obvious dependency on M, however the problems of lowest dimensionality most often have the least count. **c)** The effect of both size N and dimension M of a problem on classical and quantum memory are illustrated clearly, as well as the separation between their used amounts.

The simulation of **QMaxSearch** in this region of problem sizes confirms the dominance over CPLEX hinted at by the previous examinations. The dimension of a problem for fixed sizes N has no significant impact on the runtime of either algorithm, at least for the select values of M imposed here.

6.2 Performance with Respect to Problem Dimension M

We shift our focus even more towards the influence of the problem dimension, now fixing $N \in \{30, 60\}$ and investigating the dependence of the resources on $M \in \{5, ..., 100\}$. The knapsack tightness is again varied between $\{0.89, 0.9, 0.91\}$ to achieve problems of similar hardness for a given M, and M_{iter} is incremented up to 4000.

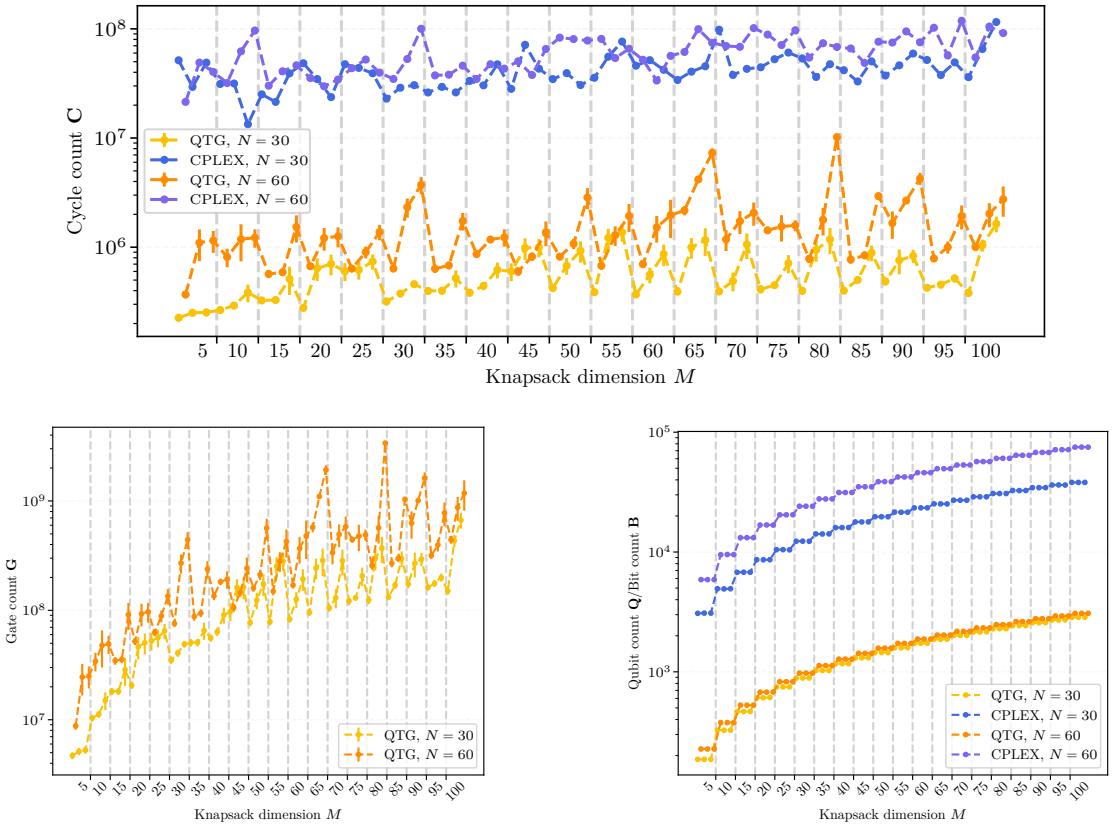

Figure 18: Resource requirements of QTG-based search and CPLEX for problem instances of different dimensions with $N \in \{30, 60\}$. **a)** The minimal values of M_{iter} alongside error bars of the CPLEX cycles have been omitted to avoid clutter. As the knapsack sizes lie in the region which favours QTG, the cycle counts are clearly separated. **b)** The corresponding amounts of gates for each N differ by a roughly constant factor. **c)** Memory savings of QTG for different M compared to CPLEX amount to an order of magnitude, similar to Figure 16 **c)**.

The elapsed cycles of both the QTG as well as the CPLEX method show only a very subtle trend upwards with increasing problem dimension, as is suggested by the plots prior, instead being affected more by the knapsack size N. For QTG this is unsurprising, seeing as the cycle count $\mathbf{C}_{\text{QTG}}$ of the maximally parallelised QTG depends largely on the greatest capacity $c_{j'}$, which for the examined instances is directly related to the number of items as presented in the problem generation scheme of Section 5.3. The qubit and gate counts $\mathbf{Q}_{\text{QTG}}, \mathbf{G}_{\text{QTG}}$, on the contrary, show dependence on all capacities c_j and thus increase with M, as depicted in Figure 18 **b)**, **c)**.

6.3 Performance with Respect to Knapsack Tightness α

The final set of problems is generated to have tightness parameters of $\alpha \in \{0.5, ..., 0.95\}$ in order to provide insight on the consumed classical and quantum resources in relation to the hardness of a given problem instance. The knapsack size is again kept constant, albeit now at a still QTG-friendly $N = 100$, while the number of constraints M is selected from $\{5, 10, 15, 20\}$.

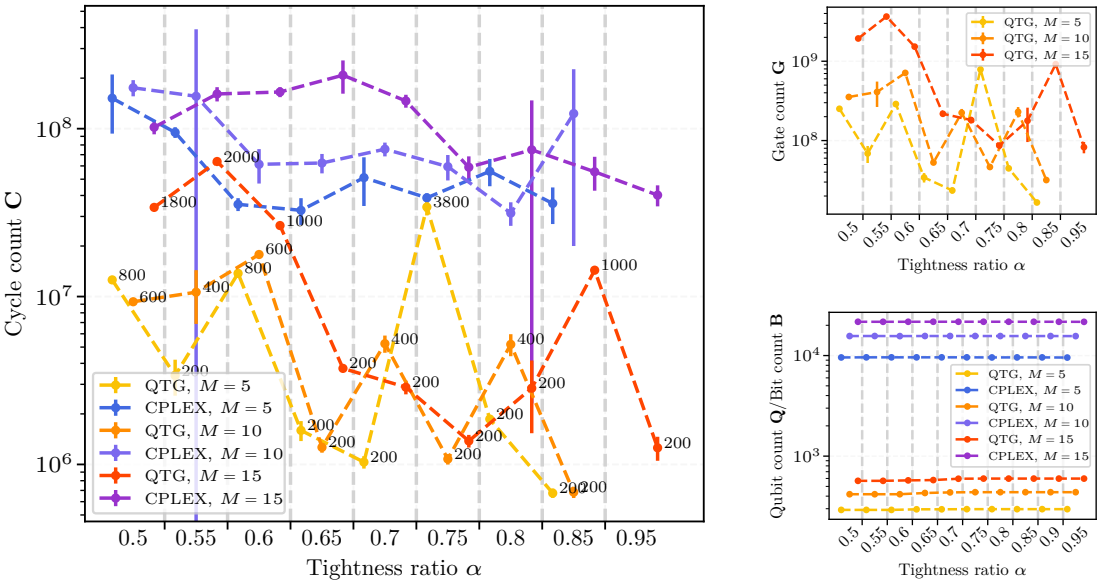

Figure 19: Resource requirements of QTG-based search and CPLEX for problem instances of different tightness parameters α with $M \in \{5, 10, 15\}$. **a)** Tighter problems give rise to larger cycle counts, both for CPLEX as well as the QTG algorithm. **b)** As in the first three studied cases, the gate counts here are also relate to the corresponding cycle counts. **c)** Since α at most influences the magnitudes of the capacities c_j, the near constant amounts of memory for each N are to be expected.

We find that decreasing values of α are linked to greater expenses of cycles in both **QMaxSearch** and CPLEX. For the former, a tighter knapsack means less items can be introduced before the break item is reached, in turn implying more cycles and gates as part of the involved additional comparisons. The data suggests CPLEX has a more favourable slope when it comes to problems of lesser α, however the QTG method still presents the least cycles for every problem.

7 Conclusion

Starting from its basic quantum computational components consisting of the quantum Fourier transform and associated addition and subtraction, as well as comparison circuits, to form its centerpiece, the Quantum Tree Generator $\mathcal{G}$, the QTG algorithm for the 0-1 multidimensional knapsack problem has been developed, making use of Quantum Amplitude Amplification and Quantum Maximum Search to comb the search space for solutions of increasing quality. Coupled to this, the required amounts of qubit memory, gates and cycles were precisely determined up to the randomness of the **QSearch** subroutine, in a way which prioritises a minimal cycle count by means of various parallelisation strategies. To obtain exact numbers on the resources needed for solving a given problem instance, the high-level QTG simulator for the 0-1 knapsack problem was altered to also mimic the algorithm for its more general version, and subsequently implemented to collect resource data for a variety of random problems alongside the all-purpose Linear Programming solver CPLEX. Their analysis for instance classes with varied knapsack sizes N, problem dimensions M and tightness ratios α as well as maximum iterations M_{iter} of $\mathcal{G}$ within **QSearch** leaves us with the surprising result that it is the low-variable realm of $N \lesssim 800$ in which the QTG algorithm excels, needing, at its best, less than a tenth of the cycles elapsed during the runtime of CPLEX on the same problem instance. The dimension M of the problems was determined to not affect the cycle count of either solving method as drastically as the knapsack size N, while the tightness α of the instances was found to have a larger effect on the QTG algorithm. Across all numerical experiments, the expected reduced memory usage of the quantum over the classical algorithm has been demonstrated. When it comes to further improvement of the QTG algorithm for 0-1 MDKP, we must first mention several options offered by Wilkening et al.[18], which are in principle also applicable to the algorithm treated in this thesis. For one, there is the option of altering the bias b to be dependent on the weights W_{ij}, as well as that of employing upper bound heuristics within the QTG to minimise the branching. Alternatively, the QTG algorithm may be used at the end of a branch-and-bound method to obtain the solution of the remaining subproblem. In a similar vein, we add to that by referring back to our reasoning in Section 6 about using a relaxed version of CPLEX for preprocessing; this may be exchanged for a dedicated efficient routine, potentially producing even better starting thresholds for **QMaxSearch**.

The numerical analysis conducted in this thesis can be expanded upon twofold: On one hand, a more specialised solving method such as the CORAL algorithm[28] can be introduced as the classical counterpart to the QTG algorithm, presenting a perhaps more fitting partner than the very general CPLEX. On the other hand, benchmark instances from the literature or problem sets generated by other methods than the one employed here can serve as the basis for the comparison of the required resources, granting different perspectives and combining to achieve a more balanced conclusion on the matter.

References

[1] *PsiQuantum Receives $940 Million AUD ($620M USD) to Install a 1 Million Qubit Machine in Australia by 2027*. Quantum Computing Report. Apr. 2024. URL: https://quantumcomputingreport.com/psiquantum-receives-940-million-aud-620m-usd-to-install-a-1-million-qubit-machine-in-australia-by-2027/.

[2] Susan Galer. *Who Is Investing In The $850 Billion Quantum Tech Market And Why*. Forbes. Nov. 2023. URL: https://www.forbes.com/sites/sap/2023/11/06/who-is-investing-in-the-850b-quantum-tech-market-and-why/.

[3] Yu. I. Manin. "Vychislimoe i nevychislimoe (Computable and Non-computable)". In: *Soviet Radio* (1980), pp. 13–15.

[4] Richard P. Feynman. "Simulating physics with computers". In: *International Journal of Theoretical Physics* 21.6 (June 1982), pp. 467–488. DOI: 10.1007/BF02650179.

[5] P.W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[6] John Clark and Frank K. Wilhelm. "Superconducting quantum bits". In: *Nature* 453 (June 2008), pp. 1031–1042. DOI: 10.1038/nature07128.

[7] I. Pogorelov et al. "Compact Ion-Trap Quantum Computing Demonstrator". In: *PRX Quantum* 2 (2 June 2021), p. 020343. DOI: 10.1103/PRXQuantum.2.020343. URL: https://link.aps.org/doi/10.1103/PRXQuantum.2.020343.

[8] H. Watzinger et al. "A germanium hole spin qubit". In: *Nature communications* (Sept. 2018). DOI: 10.1038/s41467-018-06418-4.

[9] Ingrid Fadelli. *Team demonstrates quantum advantage on optimization problems with a 5,000-qubit programmable spin glass*. Phys.org. May 2023. URL: https://phys.org/news/2023-05-team-quantum-advantage-optimization-problems.html.

[10] Davide Castelvecchi. *IBM releases first-ever 1,000-qubit quantum chip*. Nature. Dec. 2023. URL: https://www.scientificamerican.com/article/ibm-releases-first-ever-1-000-qubit-quantum-chip/.

[11] Michael Brooks. *IBM wants to build a 100,000-qubit quantum computer*. MIT Technology Review. May 2023. URL: https://www.technologyreview.com/2023/05/25/1073606/ibm-wants-to-build-a-100000-qubit-quantum-computer/.

[12] John Russel. *PASQAL Issues Roadmap to 10,000 Qubits in 2026 and Fault Tolerance in 2028*. HPC Wire. Mar. 2024. URL: https://www.hpcwire.com/2024/03/13/pasqal-issues-roadmap-to-10000-qubits-in-2026-and-fault-tolerance-in-2028/.

[13] D. Deutsch and R. Jozsa. "Rapid solution of problems by quantum computation". In: *Proceedings of the Royal Society A* 439 (1907 Dec. 1992), pp. 553–558. DOI: 10.1098/rspa.1992.0167.

[14] Daniel Garisto. *Light-Based Quantum Computer Exceeds Fastest Classical Supercomputers*. Scientific American. Dec. 2020. URL: https://www.scientificamerican.com/article/light-based-quantum-computer-exceeds-fastest-classical-supercomputers/.

[15] Michael Baczyk. *Realizing Quantum Potential: Study of Quantum Computing Use Cases Developed Worldwide*. Quantum Computing Report. May 2024. URL: https://quantumcomputingreport.com/realizing-quantum-potential-study-of-quantum-computing-use-cases-developed-worldwide/.

[16] Ben W. Reichardt. "The quantum adiabatic optimization algorithm and local minima". In: *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*. STOC '04. Chicago, IL, USA: Association for Computing Machinery, 2004, pp. 502–510. ISBN: 1581138520. DOI: 10.1145/1007352.1007428. URL: https://doi.org/10.1145/1007352.1007428.

[17] Kostas Blekos et al. "A review on Quantum Approximate Optimization Algorithm and its variants". In: *Physics Reports* 1068 (June 2024), pp. 1–66. ISSN: 0370-1573. DOI: 10.1016/j.physrep.2024.03.002. URL: http://dx.doi.org/10.1016/j.physrep.2024.03.002.

[18] Sören Wilkening et al. *A quantum algorithm for the solution of the 0-1 Knapsack problem*. 2023. arXiv: 2310.06623 [quant-ph].

[19] Lov K. Grover. "A fast quantum mechanical algorithm for database search". In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. STOC '96. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, pp. 212–219. ISBN: 0897917855. DOI: 10.1145/237814.237866. URL: https://doi.org/10.1145/237814.237866.

[20] Silvano Martello, David Pisinger, and Paolo Toth. "Dynamic Programming and Strong Bounds for the 0-1 Knapsack Problem". In: *Management Science* 45 (Mar. 1999), pp. 414–424. DOI: https://doi.org/10.1287/mnsc.45.3.414.

[21] *IBM® ILOG® CPLEX® Optimization Studio*. IBM. URL: https://www.ibm.com/products/ilog-cplex-optimization-studio.

[22] M. J. Magazine and Osman Oguz. "A heuristic algorithm for the multidimensional zero-one knapsack problem". In: *European Journal of Operational Research* 16.3 (June 1984), pp. 319–326.

[23] "An Improved Heuristic for Multidimensional 0-1 Knapsack Problems". In: *The Journal of the Operational Research Society* 41.10 (1990), pp. 963–970. ISSN: 01605682, 14769360. URL: http://www.jstor.org/stable/2583274 (visited on 05/29/2024).

[24] Yalçın Akçay, Haijun Li, and Susan Xu. "Greedy algorithm for the general multidimensional knapsack problem". In: *Annals of Operations Research* 150.1 (Mar. 2007), pp. 17–29. DOI: 10.1007/s10479-006-0150-4.

[25] P. C. Chu and J. E. Beasley. "A Genetic Algorithm for the Multidimensional Knapsack Problem". In: *Journal of Heuristics* 4.1 (June 1998), pp. 63–86. DOI: 10.1023/A:1009642405419.

[26] Md. Abul Kalam Azad, Ana Rocha, and Edite Fernandes. "Solving Large 0–1 Multidimensional Knapsack Problems by a New Simplified Binary Artificial Fish Swarm Algorithm". In: *Journal of Mathematical Modelling and Algorithms in Operations Research* 14 (Aug. 2015). DOI: 10.1007/s10852-015-9275-2.

[27] Bezalel Gavish and Hasan Pirkul. "Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality". In: *Mathematical Programming* 31 (1985), pp. 78–105.

[28] M.Grazia Speranza and Renata Mansini. "CORAL: An Exact Algorithm for the Multidimensional Knapsack Problem". In: *Informs Journal on Computing* 24 (July 2012), pp. 399–415. DOI: 10.1287/ijoc.1110.0460.

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

[30] Thomas G. Draper. *Addition on a Quantum Computer*. 2000. arXiv: quant-ph/0008033 [quant-ph].

[31] Gilles Brassard et al. *Quantum amplitude amplification and estimation*. 2002. DOI: 10.1090/conm/305/05215. URL: http://dx.doi.org/10.1090/conm/305/05215.

[32] Christoph Durr and Peter Hoyer. *A Quantum Algorithm for Finding the Minimum*. 1999. arXiv: quant-ph/9607014 [quant-ph].

[33] *Discrete optimization in the IBM® ILOG® CPLEX® Optimization Studio Documentation*. IBM. URL: https://www.ibm.com/docs/en/icos/22.1.1?topic=cplex-discrete-optimization.

[34] Arnaud Fréville and Gérard Plateau. "An Efficient Preprocessing Procedure for the Multidimensional 0- 1 Knapsack Problem". In: *Discret. Appl. Math.* 49 (1994), pp. 189–212. URL: https://api.semanticscholar.org/CorpusID:35099523.

[35] Peter Palotas and Juan Jesus Alcolea Picazo. *Memory Supervision System*. URL: https://libmss.sourceforge.net/index.php.