SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Quantum Algorithms for Constraint Satisfaction Problems

Arda Nacar

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Quantum Algorithms for Constraint Satisfaction Problems

Quantenalgorithmen zur Lösung von Bedingungserfüllungsproblemen

Author:	Arda Nacar
Supervisor:	Prof. Christian Mendl
Advisor:	Irene López Gutierrez
Submission Date:	15.02.2023

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.02.2023 Arda Nacar

Abstract

The field of quantum computing is being explored since decades and to this day, it has proven proven its potential to outperform classical computing in certain problems. Many of these promising algorithms have been proposed to tackle search problems. In this thesis, our goal is to expend the field of use of these algorithms by modelling constraint satisfaction problems as search problems. Previous work has been done on how constraint satisfaction problems can benefit from quantum computing to accelerate filtering for "alldifferent" constraints [1] and backtracking[2]. However, our focus in this thesis will be on solving constraint satisfaction problems with arithmetic constraints quantum amplitude amplification.

Contents

1 Introduction

Computers have been assisting us in our daily lives to solve certain kind of problems. They are especially useful when we are faced with problems of large scale that would normally take a human hours or maybe even days to solve. There is a certain category of problems that comes up in various parts of our lives called constraint satisfaction problems. Even fitting all the your groceries you just bought into your fridge could be modelled as constraint satisfaction problems. While we might not need computer assistance for placing our groceries, some other constraint satisfaction problems like arranging the sitting plan for a thousand people wedding can prove to be quite tricky. For problems like this, letting a computer do the job seems to make sense. However, at large scales, even computers might have trouble solving these type of problems in a reasonable amount of time. So what is the solution? Should we come up with more efficient algorithms that our computers can run. Unfortunately, we find out that many of these problems can only be solved by naively trying out each possible option by our classical computers, which is not very efficient.

Luckily, this thesis does not deal with classical computers but with quantum computers. By exploiting the properties of quantum particles, e.g. superposition and entanglement, quantum algorithms have a significant speedup to offer against classical algorithms. In this thesis we are going to explore, how some of the previously proposed quantum algorithms can be reused to solve constraint satisfaction problems more efficiently than a classical computer can. For this purpose, will use the game of Kakuro as an example. We hope this to illustrate, how constraint satisfaction problems with arithmetic constraints could benefit from these algorithms.

1.1 Repository

In the following repository, you can find implementations of the algorithms that will be discussed during the course of this thesis:

https://github.com/ardanacar/quantum-arithmetic-cps.git

2 Constraint Satisfaction Problems

Constraint satisfaction problems (CPSs) make up a remarkable part of the problem set that we encounter in our businesses, social life or even in our daily lives. For example, planning university lecture schedules, furniture placement in a room or many games like sudoku can be modelled as constraint satisfaction problems, and since we encounter them so often, it is of our interest to solve them as efficiently as possible.

Constraint satisfaction is known to be an NP-complete problem, meaning that there is no deterministic classical computer that can solve such a problem in polynomial time. For example, let us consider the boolean satisfiability problem that tackles the question whether there is an assignment of a set of logical variables, so that a logical term defined over these variables evaluates to true. A naive approach to solving this problem would be to try out every possible assignment one after the other, which results in $O(n^2)$ complexity.

However, in many cases CSPs have a certain structure, and this structure can be used to optimize the complexity of an algorithm (a solver) that solves this problem. For example, the 3-SAT problem can be solved with $O(1,307^n)$ time complexity [3], whereas a naive solver would solve it in $O(2^n)$ time complexity.

Nevertheless, one can always think of problems, where it is not possible to exploit a given structure. An example for such a problem could be cracking a password, where any set of n characters from a given character set might be a valid password, so that there is no way to validate a sub-solution of length less then n. In cases like this, we cannot do better then $O(m^n)$ with a classical solver, where m is the number of characters in the given character set.

On the other hand, quantum computation offers a speedup even in cases, where we do not have a certain structure of the CSP. Like many other quantum algorithms, quantum algorithms that solve CSPs also benefit from quantum parallelity to achieve the speedups that they offer. In this thesis, we will talk about different quantum algorithms that have been proposed and discuss how they can be used to solve CSPs. More specifically, we are interested in how these algorithms could be utilized to implement a quantum circuit that solves a game of Kakuro.

Kakuro is a Sudoku-like game, where one tries to fill every box of the puzzle with a number, such that the consecutive numbers that form a row or a column add up to the given numbers on the edges. Much like in a Sudoku, only numbers from 1 to 9 are allowed and each number can only used once for each sum. But we will ignore these two rules to simplify the problem.

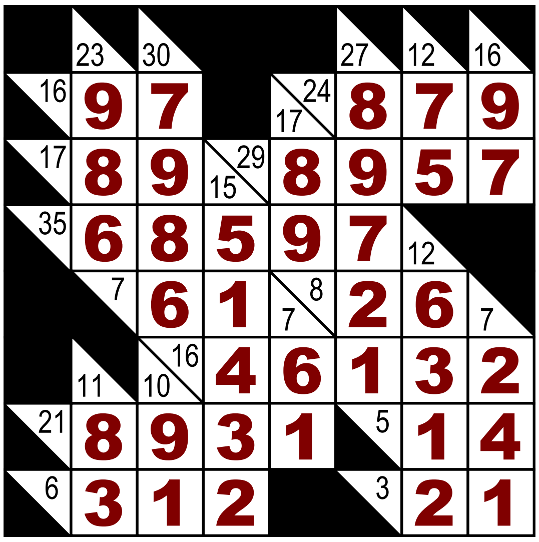

Figure 2.1: a solved Kakuro example[4]

The constraints of a Kakuro can be constructed as a set of i sums o f the form $\sum_{n=1}^{m} k_{ni} v_n = x_i$, where m is the number of variables in our Kakuro and k_n indicates, whether or not n-th variable takes part in the i-th sum. Using this notation, k_{ni} and x_i are the givens of our problem and we are trying to find n_v that satisfies the sums.

3 Existing Algorithms for Quantum Search Problems

We want to start by taking a look at quantum algorithms to solve quantum search problems. The reason they are interesting in the context of this thesis is because CSPs can be seen as search problems, where the state that we search is a state that satisfies the CSP.

One of the widely adopted approaches in quantum search algorithms is amplitude amplification. In this approach, one does not analytically calculate the target state, but amplify the probability of measuring it. This usually happens by using a modified version of quantum signal processing for multiple qubits, which is why it makes sense to first understand what it is. In sections 3.1 and 3.2, we will be revising the necessary knowledge from [5], while also giving intuitive explanations, where we think is necessary.

3.1 Quantum Signal processing

The goal of Quantum signal processing is to apply single-qubit rotations about two different axis to an input state, such that the transition probabilities of the input state when exposed to the given rotation sequence become arbitrary (related) polynomials over an experimentally controllable parameter. As explained in [5], we can take these rotations to be about the x-axis and the z-axis respectively. Now we want to consider a series of z-axis rotations, each followed by an x-axis rotation as follows[5].

$$U_{\vec{\phi}} = e^{i\phi_0 Z} \prod_{k=1}^{d} W(\theta) e^{i\phi_k Z} \tag{3.1}$$

Here, $W(\theta)$ is an x-axis rotation by a fixed angle θ, whereas the y-rotations ϕ_k are not necessarily of the same angle. Then we observe that such series of rotations yield interesting unitaries as in Fig.3.2 [5].

$$U_{\vec{\phi}} = \begin{pmatrix} P(a) & iQ(a)\sqrt{1-a^2} \\ iQ^*(a)\sqrt{1-a^2} & P^*(a) \end{pmatrix} \tag{3.2}$$

In the matrix above, $P(a)$ is a polynomial of a and denotes the matrix element $\langle 0| U_{\vec{\phi}} |0\rangle$. But the actual importance of such rotation series and resulting matrices is the fact that for any $P(a)$ one can find ϕ_k such that the resulting matrix encodes $P(a)$ in its $\langle 0| U |0\rangle$ element

[5]. This property will later come in handy for amplitude amplification.

One should also point out that so far the above mentioned transformation has only been shown to work on single qubit systems. Nevertheless, this can be generalised to multi-qubit systems as well, which will be shown in the following section.

3.2 Amplitude Amplification

For this section, we will first take a look at the procedure of amplitude amplification and then build an intuition about how it works, recognising parallelisms with the quantum signal processing section.

For the task of amplitude amplification, one is given the ability to perform rotations about an initial state and a target state. For an initial state $|I_0\rangle$ and a target state $|T_0\rangle$, these rotations can be represented as $e^{i\phi|I_0\rangle\langle I_0|}$ and $e^{i\phi|T_0\rangle\langle T_0|}$ respectively [5]. If we now consider a series of these rotation like in the quantum signal processing, it might not be yet clear what the effect of these rotation on an initial state should be. Because unlike in quantum signal processing, we have two rotation axis, which are not necessarily the x-axis and the z-axis. So we will try to rewrite it in such a way that it looks more like the quantum signal processing sequence. First of all, we define two more states: the $|I_\perp\rangle$ state with $\langle I_0|I_\perp\rangle = 0$ and the $|T_\perp\rangle$ state with $\langle T_0|T_\perp\rangle = 0$. An important realisation is that $|I_0\rangle$, $|I_\perp\rangle$ and $|T_0\rangle$, $|T_\perp\rangle$ can both be used to define a basis of a bloch sphere. This is a quite significant realization, because it also implies that we can consider multi-qubit systems as single qubit systems by defining the states $|I_0\rangle$, $|I_\perp\rangle$ and $|T_0\rangle$, $|T_\perp\rangle$ respectively. Doing so, we can solve an n-dimensional problem in a pair of 2-dimensional Hilbert spaces and we can use the results from the previous section.

Since we map our initial problem to a lower dimensional space, there is not a unique solution for the $|I_\perp\rangle$ and $|T_\perp\rangle$ states. So we define these states depending on some unitary U, sticking to the definitions in [5].

Suppose we are given a unitary U, such that when applied to $|I_0\rangle$ it yields a state, which has a nonzero overlap a with the $|T_0\rangle$ state. Then we define the $|T_\perp\rangle$ state as the $U|I_0\rangle$ state without its $|T_0\rangle$ portion (normalized to a unit vector). For example, if $U|I_0\rangle$ is an equal superposition over all the possible states of our n-qubit system, then $|T_\perp\rangle$ is en equal superposition over all the possible states, excluding the state $|T_0\rangle$. Similarly, we define $|I_\perp\rangle$ as follows [5]:

$$U|I_\perp\rangle = \sqrt{1-a^2}\,|T_0\rangle - a\,|T_\perp\rangle \tag{3.3}$$

The two qubit-spaces spanned by $|I_0\rangle, |I_\perp\rangle$ and $|T_0\rangle, |T_\perp\rangle$ (I space and T space respectively) are not equal to one another. This can easily be seen by choosing $|I_0\rangle$ to be $|0\rangle^{\otimes n}$ and $U|I_0\rangle$ again to be an equal superposition of all the possible states. In this case the ket $|0\rangle^{\otimes n}$ is clearly in the I space. However it is not in the T space, since it is perpendicular to both of its basis states, meaning that it cannot be written as a linear combination of its basis states. In general, this entails that we cannot reach a state in e.g. the T space by appying x-, y- or z-axis rotations to a state in the I space. On the other hand, by definition U returns a ket in the T space when given a ket in the I space and $U^\dagger$ returns a ket in the I space when given a ket in the T space, meaning that they map betweeen qubit-spaces. We can write U in matrix form as follows [5]:

$$U = \begin{pmatrix} a & \sqrt{1-a^2} \\ \sqrt{1-a^2} & -a \end{pmatrix} \tag{3.4}$$

Similar to what we did in the QSP section, we now analyse the following operator sequence [5]:

$$e^{i\phi_0 |T_0\rangle\langle T_0|} \left[\prod_{k=1}^{\frac{d-1}{2}} U e^{i\phi_{2k-1}|I_0\rangle\langle I_0|} U^\dagger e^{i\phi_{2k}|T_0\rangle\langle T_0|} \right] U \tag{3.5}$$

We realize that $e^{i\phi_0|I_0\rangle\langle I_0|}$ and $e^{i\phi_0|T_0\rangle\langle T_0|}$ can be simply represented as z-axis rotations, if we are in the respective qubit-space [5]. Even more strictly, in general these operators cannot be represented at all, if we are not in the right qubit-space. Because we would be trying to rotate around an axis, that might not be representable in the respective qubit-space. But our sequence is constructed such that we translate into the right qubit-space using the U and $U^\dagger$ operators before applying the rotation operators. So we can rewrite our sequence as follows[5]:

$$e^{i\phi_0 Z} \left[\prod_{k=1}^{\frac{d-1}{2}} U e^{i\phi_{2k-1} Z} U^\dagger e^{i\phi_{2k} Z} \right] U \tag{3.6}$$

Another important thing to realize is that the matrix form of the U operator has the structure of the operator $XR_y(\theta)$, where $R_y(\theta)$ is a y-axis rotation by $\theta = 2arcsin(a)$[5]. Despite this structural similarity, we know that U cannot be a rotation operator or a concatenation of multiple rotation operators. Because it maps between qubit-spaces and we explained above, that this cannot be done with rotation operators. This suggests that our U operator and the $XR_y(\theta)$ are syntactically the same but have different semantics. Nonetheless, as far as the algebra is concerned semantics are not relevant. So we can threat our U operator the same way as the $XR_y(\theta)$ operator.

Furthermore, in the 3-D space, in which our bloch sphere lives, we can represent a y-axis rotation as a combination of x-rotations and z-rotations as follows[5]:

$$R_y(\theta) = e^{-i\frac{\pi}{4}Z} W(-\theta) e^{i\frac{\pi}{4}Z} \tag{3.7}$$

When we substitute U in our operator sequence using (3.7) and rewrite the X operator as an x-axis rotation by π, we get

$$e^{i\phi_0 Z}\left[\prod_{k=1}^{\frac{d-1}{2}} W(\pi)e^{i\frac{\pi}{4}Z}W(-\theta)e^{-i\frac{\pi}{4}Z}e^{i\phi_{2k-1}Z}W(\pi)e^{i\frac{\pi}{4}Z}W(-\theta)e^{-i\frac{\pi}{4}Z}e^{i\phi_{2k}Z}\right]U \tag{3.8}$$

Nevertheless, this sequence can be simplified by using the following rules:

$$Xe^{-ix} = e^{ix}X \tag{3.9}$$

$$W(-\theta) = e^{-i\frac{\pi}{2}}W(\theta)e^{i\frac{\pi}{2}} \tag{3.10}$$

Applying these two rules and joining the consecutive x- and z-rotations together leaves us with the sequence[5]

$$S = e^{i\phi_0' Z}\left[\prod_{k=1}^{\frac{d-1}{2}} W(\theta+2\pi)e^{i\phi_{2k-1}' Z}W(\theta)e^{i\phi_{2k}' Z}\right]U = e^{i\phi_0' Z}\left[\prod_{k=1}^{\frac{d-1}{2}} W(\theta)e^{i\phi_{2k-1}' Z}W(\theta)e^{i\phi_{2k}' Z}\right]U \tag{3.11}$$

As it can now be recognised, this is exactly the sequence 3.1 that we had in section 3.1. This implies that amplitude amplification is also built on the foundation of quantum signal processing and therefore transforms the matrix element $\langle T_0| U |I_0\rangle$ according to some polynomial $P(a)$ that depends on the choice of ϕ_k, where $a = sin(\frac{\theta}{2})$. It is also important to notice that a describes, how far the initial state is from the target state. Under this realisation, the polynomial $P(a) = \langle T_0| S |I_0\rangle$ takes on an important meaning, because it shows us how high the probability of measuring a target state is, depending on how close the two states $U|I_0\rangle$ and $|T_0\rangle$ are.

This is where the choice of the polynomial $P(a)$ becomes critical. In cases, where we know the value of $a = \langle T_0| U |I_0\rangle$, it makes sense to choose a polynomial that reaches values close to 1 for this known a. Nonetheless, we do not always know the value of a, e.g. because of unknown number of solutions. In that case, it makes sense to choose a polynomial that is close to 1 for the most of the interval $a \in [-1, 1]$, e.g. the polynomial approximation of the sign function[5].

3.3 Grover's Search Algorithm

Probably the best known algorithm that utilizes amplitude amplification is the Grover's search algorithm [6]. It is a special case of amplitude amplification. In this algorithm $U|I_0\rangle$ is the equal superposition of 2^n basis states of an n-qubit system. Choosing $U|I_0\rangle$ as such implies that no matter what the target state is, it holds that $a = \sqrt{1/N}$ with N being the number of possible target states and assuming that there is only one target state [5]. In general $a = \sqrt{M/N}$, where M is the number of target states. Furthermore, in Grover's search algorithm all ϕ_k are chosen to to be equal to π for $k \neq 0$ and ϕ_0 is equal to 0 [5].

Even though using Grover's algorithm saves us the effort of determining the suitable rotation sequence, it also has some down sides. Because in Grover's algorithm $\langle T_0| S |I_0\rangle$ oscillates with increasing number of iterations and the right number of iterations depends on M. M is however not always known. Our Kakuro example is a good illustration of this issue, since the number of a, b that satisfy the constraint $a + b = c$ depends on c and therefore not fix. This also holds for more complex Kakuros, which will be shown and exemplified later. This fact prevents us from constructing a one size fits all circuit that finds solutions to "sum-constraints" using Grover's algorithm with close to 100% probability.

Next we will derive the optimal number of iterations for Grover's search algorithm by viewing it as a QSP sequence. A single Grover iteration can be written as follows:

$$U_g = W(\pi)e^{i\frac{\pi}{4}Z}W(-\theta)e^{-i\frac{\pi}{4}Z}e^{-i\frac{\pi}{2}Z}W(\pi)e^{i\frac{\pi}{4}Z}W(-\theta)e^{-i\frac{\pi}{4}Z}e^{-i\frac{\pi}{2}Z} \tag{3.12}$$

Using the rule 3.9 and joiining the the consecutive rotations about the same axis together, we can rewrite this equation into

$$U_g = e^{-i\frac{\pi}{4}Z}W(\pi-\theta)e^{-i\pi Z}W(\pi-\theta)e^{-i\frac{3\pi}{4}Z} = e^{-i\frac{\pi}{4}Z}W(-2\theta)e^{-i\frac{3\pi}{4}Z}$$

Then we multiply the matrix representations of the operators.

$$U_g = \begin{pmatrix} -cos(\theta) & -sin(\theta) \\ sin(\theta) & -cos(\theta) \end{pmatrix}$$

This is equivalent to the matrix below, up to a global phase.

$$\begin{pmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{pmatrix}$$

Since we will (most likely) not be using a single iteration, we need to determine which form U_g^d takes, where d is the number of iterations. We can ignore any global phase here as well.

$$U_g^d = \begin{pmatrix} cos(d\theta) & (-1)^{d+1}sin(d\theta) \\ (-1)^d sin(d\theta) & cos(d\theta) \end{pmatrix} \tag{3.13}$$

To fully represent our circuit C_g, we also need to post multiply U_g^d with the initialization unitary, which corresponds to the U unitary in 3.11.

$$\begin{aligned} C_g &= \begin{pmatrix} cos(d\theta) & (-1)^{d+1}sin(d\theta) \\ (-1)^d sin(d\theta) & cos(d\theta) \end{pmatrix} \begin{pmatrix} sin(\frac{\theta}{2}) & cos(\frac{\theta}{2}) \\ cos(\frac{\theta}{2}) & -sin(\frac{\theta}{2}) \end{pmatrix} \\ &= \begin{pmatrix} (-1)^{d+1}sin(\frac{2d+1}{2}\theta) & cos(\frac{2d+1}{2}\theta) \\ cos(\frac{2d+1}{2}\theta) & (-1)^d sin(\frac{2d+1}{2}\theta) \end{pmatrix} \end{aligned} \tag{3.14}$$

The success probability of our circuit is modelled by the squared matrix element

$$\left| \langle T_0 | C_g | 0 \rangle^{\otimes 2n} \right|^2 = sin^2(\frac{2d+1}{2}\theta) \tag{3.15}$$

So, we need to choose d such that $sin(\frac{2d+1}{2}\theta) = 1$ in order to maximize the success probability. This implies $\frac{2d+1}{2}\theta = \frac{\pi}{2}$. By using

$$sin(\frac{\theta}{2}) = \sqrt{\frac{T_0}{N_0}} \tag{3.16}$$

and solving for d we get:

$$d = \frac{\pi}{4sin^{-1}(\sqrt{\frac{T_0}{N_0}})} - \frac{1}{2} \approx \frac{\pi}{4}\sqrt{\frac{N_0}{T_0}} - \frac{1}{2} \tag{3.17}$$

As one can see in Fig.3.1, the approximation of the iteration number differs from the exact formula by at most $0,3$. Also remember that the iteration number has to be an integer. So if we round down the results from the both formulas, the approximation either returns the same answer as the exact formula or returns one more iteration than the exact formula does.

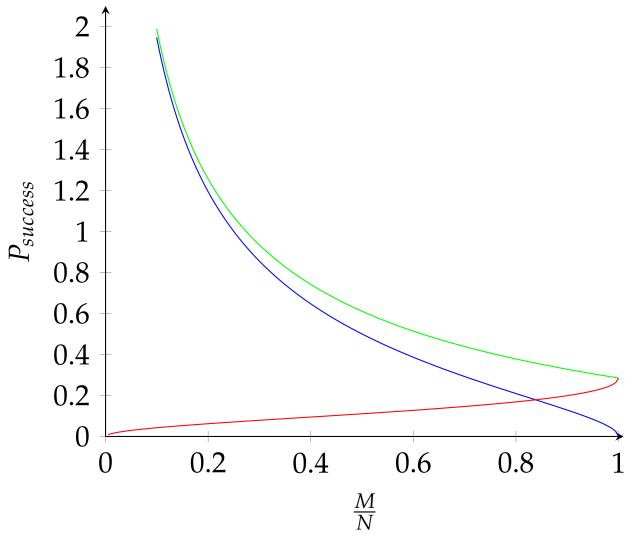

Figure 3.1: exact number of needed iterations(blue), approximation of needed number of iterations(green), approximation error(red)

3.4 Amplitude Amplification with Partial Diffusion

A slightly different approach was proposed in [7]. We will refer to this algorithm as the partial diffusion algorithm. Before explaining how it works, we first want to give a motivation why it might be useful in our case. We already said that $\langle T_0 | S | I_0 \rangle$ oscillates with changing number of iterations. But what is also interesting for us is that the success probability $| \langle T_0 | S | I_0 \rangle |^2$

also oscillates for a fix number of iterations and a varying $\frac{M}{N} \in [0, 1]$. the importance of this behaviour comes from the fact that in the end we will have to choose a fixed number of iterations, while not knowing the value of $\frac{M}{N}$. This will give us a statistical result, depending on how $|\langle T_0| S |I_0\rangle|^2$ reacts to varying $\frac{M}{N}$. For example, by solving 3.16 for θ and inserting it in 3.15 we can see that Grover's search algorithm transforms the initial success probability $|\langle T_0| U |I_0\rangle|^2$ to the post processing success probability $|\langle T_0| S |I_0\rangle|^2$ according to the function

$$p_d(x) = \sin^2\left((d + \frac{1}{2})2\arcsin\left(\sqrt{x}\right)\right) \tag{3.18}$$

where $x = \frac{M}{N}$ and d is the number of iterations. If we plot this function, we see that it oscillates between 1 and 0. On the other hand, the partial diffusion algorithm achieves a probability transformation function that still oscillates, but does not return to zero probability for any $x > 0$ [7].

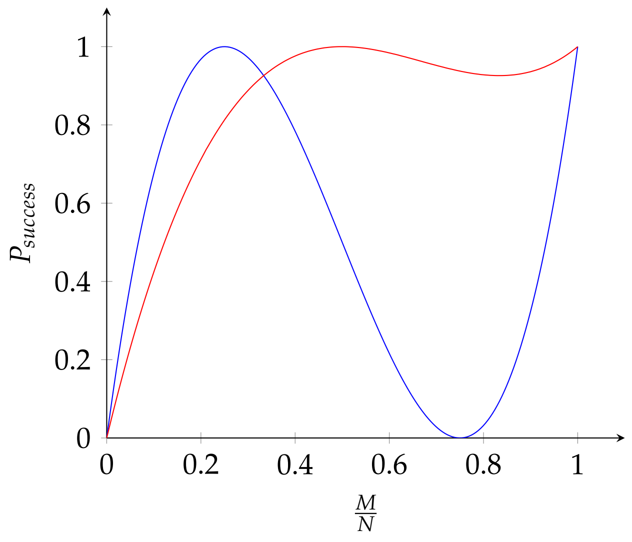

Figure 3.2: probability transformation of Grover's algorithm after one iteration(blue), probability transformation of the partial diffusion algorithm after one iteration(red)

As one can see in Fig. 3.2, if we were to integrate these two functions in an interval I close to $\frac{M}{N} = 1$, the partial diffusion algorithm yield a much higher value compared to Grover's search algorithm. This means that if a Kakuro is equally likely to have $m * N$ solutions for $m \in I$, than in average the partial diffusion operation succeeds with a higher probability. Even though the we will set the interval I of likely initial probabilities to be relatively small and close to 0 in the implementation section, it might still be worth comparing how well it performs against Grover's search algorithm.

Now we want to move on to the technical part of the algorithm. Similar to the Grover's algorithm, the partial diffusion algorithm consists of iterations of two operators. But unlike in the Grover's algorithm, the first unitary U_f flips the state of an ancillary qubit if the working register(all the qubits except for the ancilla qubit) is a target state, instead of flipping the phase of these states[7].

$$U_f \left|I\right\rangle \left|0\right\rangle = \sqrt{\frac{M}{N}} \left|T\right\rangle \left|1\right\rangle + \sqrt{1 - \frac{M}{N}} \left|T_\perp\right\rangle \left|0\right\rangle \tag{3.19}$$

After this step, the second operator Y performs an inversion of the amplitudes like Grover's algorithm also does, but as the name suggests, it only performs this operation on the subspace where the ancilla qubit is in the state $\left|0\right\rangle$[7].

$$Y = H^{\otimes n} \otimes \mathbb{I} \left(2\left|0\right\rangle\left\langle 0\right| - \mathbb{I}\right) H^{\otimes n} \otimes \mathbb{I} \tag{3.20}$$

Even if both of these operations seem to be quite similar to Grover's algorithm, we can see the differences when we examine how it evolves the target state probability. In Grover's search an iteration of the algorithm amplifies each solution state equally. But this is not the case with the partial diffusion algorithm. Notice that adding an ancilla qubit to our system doubles the number of basis states and with that also the number of target states, so that for each solution state $\left|t_m\right\rangle$ our new solution states become $\left|t_m\right\rangle\left|0\right\rangle$ and $\left|t_m\right\rangle\left|1\right\rangle$. Nevertheless, an iteration of the partial diffusion algorithm only equally amplifies the solution states in the subspace where the ancilla qubit is in the $\left|0\right\rangle$ state and than switches their amplitudes with the solution states in the subspace where the ancilla is in the $\left|1\right\rangle$ state. This behaviour is explained in more detail in [7] and it makes it quite hard to model this as a QSP sequence. Because it results in a recursive definition of the amplitudes. Nevertheless, one can still derive their closed forms as it is done in [7]. As a result, the amplitudes a, b and c of the states $\left|t_{\perp n}\right\rangle \otimes \left|0\right\rangle$, $\left|t_m\right\rangle \otimes \left|0\right\rangle$ and $\left|t_m\right\rangle \otimes \left|1\right\rangle$ after d iterations can be written as follows ($m \in [1, M]$ and $n \in [1, N-M]$ are indexes of target/non-target states):

$$a_d = \frac{1}{\sqrt{N}} \left(\frac{\sin\left((d+1)\theta\right) - \sin\left(d\theta\right)}{\sin\left(\theta\right)}\right) \tag{3.21}$$

$$b_d = \frac{1}{\sqrt{N}} \left(\frac{\sin\left((d+1)\theta\right)}{\sin\left(\theta\right)}\right) \tag{3.22}$$

$$c_d = -\frac{1}{\sqrt{N}} \left(\frac{\sin\left(d\theta\right)}{\sin\left(\theta\right)}\right) \tag{3.23}$$

where $\frac{M}{N} = 1 - \cos(\theta)$ [7]. Since both $\left|T\right\rangle \otimes \left|0\right\rangle$ and $\left|T\right\rangle \otimes \left|1\right\rangle$ are equal superpositions of M states, the success probability after d iterations becomes ([7]):

$$P_d = M\left(b_d^2 + c_d^2\right) \tag{3.24}$$

This results in the function plotted with red in Fig. 3.2. Next we want to plot b and c as a function of number of iterations d for the case that $\frac{M}{N} = 0.2$ (we assume d to be continuous).

The point where the two functions in Fig. 3.3 intersect corresponds to the optimal number of iterations for the case $\frac{M}{N} = 0,2$ and the probability of measuring a target state in this point

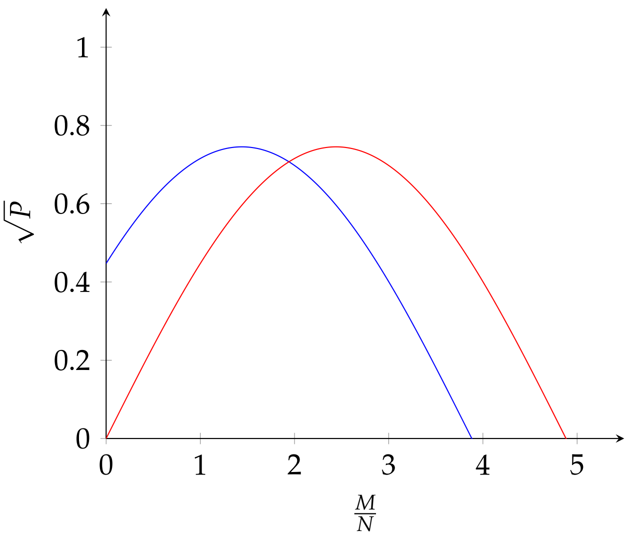

Figure 3.3: amplitudes of the states $|T_0\rangle\,|0\rangle$(blue) and $|T_0\rangle\,|1\rangle$(red) for $\frac{M}{N} = 0,2$

is 1. On the other hand, if we iterate too few times, we observe that measuring a target state where the ancilla qubit is in the $|0\rangle$ state is higher than measuring a target state, where the ancilla qubit is in the $|1\rangle$ state. In this case these two probabilities do not add up to 1, since we iterated a suboptimal number of times. This affect will also be visible in our Kakuro example, since in general the optimal number of iterations will not be an integer and we will not have exact information about the number of solutions of the Kakuro. Even though this affect will not have an impact on the performance of the algorithm, we thought that it might be interesting to discuss this from an experimental point of view.

3.5 $\frac{\pi}{3}$ Algorithm

Another algorithm that we thought is worth mentioning is the $\frac{\pi}{3}$ algorithm, which was also proposed by Lov K. Grover [8]. This algorithm is very similar to the Grover's search algorithm as well. It only differs in the amount of phase shifts $e^{i\phi_k |T_0\rangle\langle T_0|}$ and $e^{i\phi_l |I_0\rangle\langle I_0|}$ applied in each iteration. For the Grover's algorithm all of the shifts were by π radians, whereas the $\frac{\pi}{3}$ algorithm predictably chooses them to be all by $\frac{\pi}{3}$ radians. However this small change affects the resulting QSP polynomial drastically. Specifically, one can observe that after one iteration of this algorithm, the failure probability becomes [8]

$$1 - |\langle T_0|\, U e^{i\frac{\pi}{3}|I_0\rangle\langle I_0|} U^{\dagger} e^{i\frac{\pi}{3}|T_0\rangle\langle T_0|} U\, |I_0\rangle\,|^2 = \left(1 - |\langle T_0|\, U\, |I_0\rangle\,|^2\right)^3 \tag{3.25}$$

This implies that with growing number of iterations the failure probability decreases exponentially rather than oscillating between 0 and 1. With this property we overcome the so called "*soufflé*" problem [8], which means that we cannot overshoot the the optimal iteration number, since there is no optimum iteration number. The more we iterate, the better results

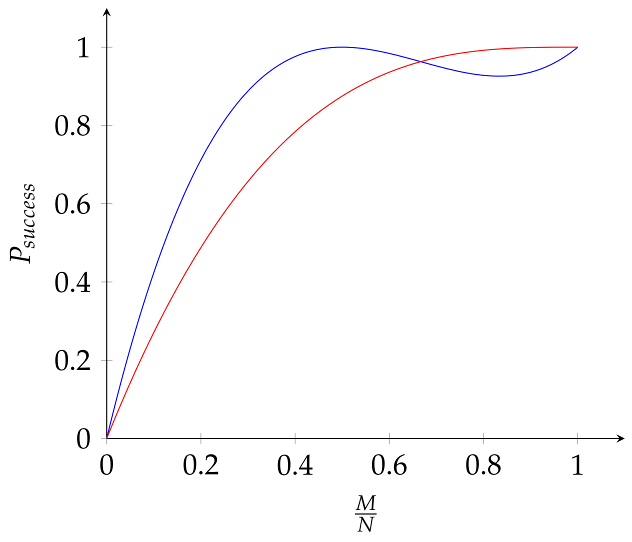

Figure 3.4: probability transformation functions of the partial diffusion algorithm(blue) and the $\frac{\pi}{3}$ algorithm after one iteration

we get. This behaviour results in a much better performance compared to the aforementioned algorithms in certain cases [8].

As it can be seen in Fig. 3.4 the $\frac{\pi}{3}$ algorithm performs much better if we assume $\frac{M}{N}$ to be equally distributed in an interval close to 1 for our problem. If we iterate this algorithm often enough, we can enlarge the interval $[x, 1]$ that this algorithms mean performance is superior to the algorithms that we previously mentioned above. However, as we also mentioned in the previous section, in our Kakuro example $\frac{M}{N}$ is expected to be relatively small and bound to a rather small interval. In this case the convergence speed of the algorithm becomes important, where the $\frac{\pi}{3}$ algorithm is easily outperformed by the other algorithms, and the property of convergence itself plays less of a role. Note that here the convergence speed refers to how fast the first peak is reached for the algorithms that implement an oscillating QSP polynomial. For instance, it takes 49 iterations for Grover's algorithm to amplify an initial success probability of $\frac{1}{4096}$ to 0.999, whereas it would take 9841 queries for the $\frac{\pi}{3}$ algorithm to amplify it over 0.9. Also notice that we use the term queries for the $\frac{\pi}{3}$ algorithm instead of iterations, since we define an iteration of this algorithm as the step that decreases the failure probability ϵ to ϵ^3. However, this results in a recursive definition of the operator sequence. To better understand this let us consider the sequence

$$U_0 = Ue^{i\frac{\pi}{3}|I_0\rangle\langle I_0|}U^\dagger e^{i\frac{\pi}{3}|T_0\rangle\langle T_0|}U \tag{3.26}$$

whose $\langle T_0| U_0 |I_0\rangle$ element equals to ϵ and resembles a single iteration of the $\frac{\pi}{3}$ algorithm applied to the unitary U with $|T_0\rangle$ as the target state and $|I_0\rangle$ as the initial state [8]. We know that each iteration of this algorithm exponentially decreases the failure probability. So if we want to further decrease ϵ to ϵ^3, we have to consider U_0 as our new U operator and apply a single iteration of the $\frac{\pi}{3}$ algorithm. In general the d-th iteration of the algorithm takes the form [8]:

$$U_d = U_{d-1} e^{i\frac{\pi}{3}|I_0\rangle\langle I_0|} U_{d-1}^\dagger e^{i\frac{\pi}{3}|T_0\rangle\langle T_0|} U_{d-1} \tag{3.27}$$

After we decompose U_d into U, $e^{i\frac{\pi}{3}|T_0\rangle\langle T_0|}$ and $e^{i\frac{\pi}{3}|I_0\rangle\langle I_0|}$ unitaries we regroup these unitaries into groups of four unitaries. Now each group of unitaries represents a query, comparable to an iteration of Grover's algorithm. However, due to the recursive nature of this algorithm, not all queries look identical. Also, in order to increase the number of iteration from m to $m+1$ we have to add 3^m more queries to our operator sequence, which makes clear why we need to differentiate between an iteration and a query for the $\frac{\pi}{3}$ algorithm. So in the example above, 9841 queries would actually correspond to 9 iterations of the algorithm [8].

Nevertheless, it is the query complexity that matters, since queries are the units of the $\frac{\pi}{3}$ that are time complexity-wise comparable to the iterations of the algorithms we previously discussed. As it should have been clear, this algorithm is not suitable for our Kakuro example and therefore will not be implemented in the next chapter.

3.6 Mizel's critically damped quantum search algorithm

The final algorithm that we are going to discuss is Mizel's critically damped quantum search algorithm [9]. This algorithm is fundamentally different compared to the previously discussed algorithms, because it does not require knowing the number of target states but unlike the $\frac{\pi}{3}$ algorithm it returns the target state with certainty while maintaining a quadratic speed up against classical search. And unlike the other algorithms, it also does not require determining the number of iterations ahead of runtime. In a sense, it runs as long as it needs to. In this thesis, we will give an intuitional explanation of this algorithm and we refer to [9] for an in depth explanation.

To begin with, an ancilla qubit is introduced to the quantum system. This ancilla will function as an indicator of how close to the target state our system is, as mentioned in [9]. We initialize the system to the state $|\psi\rangle|1\rangle$, where $|\psi\rangle$ is an equal superposition of n qubits, n being the number of qubits ignoring the ancilla qubit. Then we apply a y-rotation by ϕ to the ancilla qubit, if the rest of the n qubits are in a target state. We will call the operator that achieves this U_y.

$$|\Psi_1\rangle = U_y|\psi\rangle|1\rangle = \alpha_0\left(\sin(\frac{\phi}{2})|T_0\rangle|0\rangle + \cos(\frac{\phi}{2})|T_0\rangle|1\rangle\right) + \beta_0|T_\perp\rangle|1\rangle \tag{3.28}$$

$$\alpha_0 = \frac{M}{N},\ \beta_0 = \frac{N-M}{N}$$

Next we apply a Grover iteration to the first n qubits, if the ancilla qubit is in the $|1\rangle$ state. To better understand how this evolves the state $U_y|\psi\rangle|1\rangle$ state, we will first analyze how the both $e^{i\pi|T_0\rangle\langle T_0|}$ and $e^{i\pi|I_0\rangle\langle I_0|}$ operators that make up a Grover iteration act on the $|T_0\rangle$ and $|T_\perp\rangle$ states. Since $e^{i\pi|T_0\rangle\langle T_0|}$ acts as a Pauli Z operator in the $|T_0\rangle$, $|T_\perp\rangle$ space, it is easy to see that both of these states are eigenstates of this operator with the eigenvalues 1 and -1. We refer to the ancilla controlled $e^{i\pi|T_0\rangle\langle T_0|}$ operator as U_{ct}.

$$|\Psi_2\rangle = U_{ct}|\Psi\rangle = \alpha_0\left(\sin(\frac{\phi}{2})|T_0\rangle|0\rangle - \cos(\frac{\phi}{2})|T_0\rangle|1\rangle\right) + \beta_0|T_\perp\rangle|1\rangle \tag{3.29}$$

When it comes to the $e^{i\pi|I_0\rangle\langle I_0|}$ operator, we know that this is the diffusion operator that does an inversion of amplitudes about the mean value of the amplitudes. Moreover, since this operation is controlled by the ancilla qubit, we can realize that this operation is actually the partial diffusion operator that we know from the partial diffusion algorithm. Using the results from [7], we see that the state $|\Psi_2\rangle$ is evolved as follows by the partial diffusion operator:

$$\begin{aligned}|\Psi_3\rangle = U_{ct}|\Psi_2\rangle = -\alpha_0\sin(\frac{\phi}{2})|T_0\rangle|0\rangle &+ \left(2\langle\mu\rangle - \frac{\alpha_0\cos(\frac{\phi}{2})}{\sqrt{M}}\right)|T_0\rangle|1\rangle \\ &+ \left(2\langle\mu\rangle - \frac{\beta_0}{\sqrt{N-M}}\right)|T_\perp\rangle|1\rangle\end{aligned} \tag{3.30}$$

$$\langle\mu\rangle = \frac{\alpha_0\cos(\frac{\phi}{2})\sqrt{M} + \beta_0\sqrt{N-M}}{N}$$

Once we prepare the $|\Psi_3\rangle$ state, we measure the ancillary qubit. If it is measured to be in the $|0\rangle$ state, we know that the first n qubits must be in the target state. On the other hand, if it is measured in the $|1\rangle$ state, the wave function of the quantum system collapses to

$$|\Psi_4\rangle = \alpha_1|T_0\rangle|1\rangle + \beta_1|T_\perp\rangle|1\rangle,\ \alpha_1 = \frac{2\langle\mu\rangle - \frac{\alpha\cos(\frac{\phi}{2})}{\sqrt{M}}}{1-\frac{M}{N}},\ \beta_1 = \frac{2\langle\mu\rangle - \frac{1}{\sqrt{N-M}}}{1-\frac{M}{N}} \tag{3.31}$$

As explained in [9], if we let ϕ change over iterations according to the rule $\phi_0 = \pi/2$ and $\phi_n = (1-\sin(\pi/2n))\ /\ (1-\sin(\pi/2n))$ for n > 0, then the expected number of iterations before the target state is reached is in $\mathcal{O}\left(1.5\sqrt{\frac{M}{N}}\right)$. This was the last algorithm that we want to consider in this thesis. In the next chapter, we will discuss how one can implement these algorithms to solve a simple Kakuro.

4 Implementation

Even though we previously mentioned that the Grover′ algorithm is not suited for our Kakuro problem, we will first start by implementing it for the illustration of the problem and then compare its performance with the other previously mentioned algorithms. As a reminder, the Grover′s algorithm relies on two rotations operators: A rotation by π around the target state and a rotation by π around the initial state. In the following, an implementation for the rotation around the target state will be proposed.

In order to perform a rotation around the target state, we first want to create a circuit that can detect the states that satisfy our constraint. Since our constraint is of the form $a + b = c$ or $c - (a + b) = 0$, it is clear that we will need to perform quantum arithmetic operations. So we will start by exploring two different ways of performing addition/subtraction of quantum states.

4.1 Classical Inspired Approach

Our first approach is inspired by classical subtraction. The essential steps are:

1. starts from the least significant qubit of the subtrahend
2. subtract it from the minuend and update all the necessary qubits
3. repeat for each qubit of the subtrahend

Classically, one could perform these steps for the numbers a (consisting of a_1 and a_0) and b (consisting of b_1 and b_0) using AND, XOR and NOT operations as follows:

1. $a_1 := a_1 \oplus (b_0 \wedge \neg a_0)$ (check for carry during $a - b_0$)
2. $a_0 := a_0 \oplus b_0$ (subtraction of b_0 from a_0)
3. $a_1 := a_1 \oplus b_1$ (subtraction of b_1 from a_1)

Analogously, one can perform the quantum version of these operations using multi-controlled-Tofolli-gates and controlled X-gates:
Since we want to implement a solver as efficiently as possible, the complexity of this circuit is of our interest. In the following, we will be using the results from [10] to find a decomposition for our circuit and evaluate its complexity. But first we need to generalise the

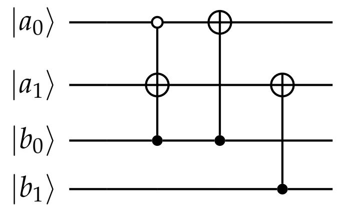

Figure 4.1: a quantum circuit that subtracts the number $|b_1 b_0\rangle$ from the number $|a_1 a_0\rangle$

above circuit to numbers with arbitrary (but equal) many bits.

To find a generalisation for our subtraction circuit, we start by determining the necessary gates for the subtraction of a single qubit from the minuend. Implementing the subtraction of equally significant qubits is trivial and can always be implemented as $a_n := a_n \oplus b_n$. On the other hand, determining the effect of subtracting b_m on a_n for the case that $n > m$ requires analysing when a carry from a_n happens. We use the expression

$$(\wedge_{x=m}^{n} \neg a_x) \wedge b_m \tag{4.1}$$

to determine if a_n should be flipped as the result of subtracting b_m from a. So the resulting state of a_n is $a_n \oplus ((\wedge_{x=m}^{n} \neg a_x) \wedge b_m)$. Moreover, we make the realization that the qubit b_m can only affect the states of the qubits a_n with $n \in [m, n_{max}]$, n_{max} being the index of the most significant qubit of a. This means that for each b_m, $n_{max} - m$ qubits need to be checked for a state flip caused by carry and each of these "checks" depend on $n - m$ qubits (indices start from 0). One should also keep in mind that for each b_m, these checks should be performed starting from the most significant qubit of a and going to the least significant. Otherwise our checks could be falsified by the updates that b_m might have caused on the less significant qubits. Generalising the example circuit above with $n_{max} = 1$, one can see that each of these checks can be implemented as an $(n - m + 1)$-controlled-Tofolli-gate, where all the controll-qubits belonging to the number a are negated with X-gates. But multi controlled-Tofolli-gates cannot be constructed with constant circuit depth. However, using the result from [10], we know that they can be implemented using linear amount of gates in the number of control-qubits m, given that the total number of qubits in the circuit n_{tot} is greater than 4 and is at least $m + 2$. Since we only need multi controlled-Tofolli-gates for subtractions with numbers with more than 3 qubits, the given constraints on n_{tot} can always be satisfied.

Now that we know how our subtraction circuit can be constructed, we can express its complexity as

$$\sum_{m=1}^{n^{max}+1} k(m) * C(m) \tag{4.2}$$

where $k(m) = n_{max} - m + 2$ is the number of m-controlled-Tofolli-gates and $C(m)$ is the complexity of implementing an m-controlled-Tofolli-gate. We have dicussed that $C(m)$ is linear in m. This means that the whole circuit has a complexity of $\mathcal{O}\left(n_{max}^3\right)$, which can be exemplified by choosing $C(m) = m$:

$$C(U) = \sum_{m=1}^{n_{max}+1} (n_{max} - m + 2) * m = \frac{1}{6}(1 + n_{max})(2 + n_{max})(3 + n_{max}) \tag{4.3}$$

Before we move on to the next possibility to implement an arithmetic operator, we want to reconsider our choice of arithmetic operator. More specifically, we will analyse whether using a subtraction operator results in an optimal spacial and computational complexity for this problem. To be able to decide this, we first need to analyse the feasibility and complexity of using an addition operator instead.

One can see that, formulating the constraints as $c - a - b = 0$ implies that we need three different registers for a, b and c, so that we can compare the final state in $|c\rangle$ to $|0\rangle$:

$$U\,|abc\rangle = U_b U_a\,|abc\rangle\,, U_a\,|abc\rangle = |ab(c-a)\rangle\,, U_b\,|abc\rangle = |ab(c-b)\rangle \tag{4.4}$$

However, if we reformulate our constraint as $a + b = c$ we can use an addition circuit to construct the sum in either of the registers reserved for a and b and compare the resulting state in this register to $|c\rangle$:

$$U\,|ab\rangle = |(a+b)b)\rangle \tag{4.5}$$

This means that we only need two registers, if we use this formulation of constraints. This is a direct result of the formulation of the problem and holds for any implementation of the addition operation, as long as the operator itself does not require any auxiliary registers. This is a feasible requirement for the classical inspired approach, as we realize that each gate used in the implementation of the subtraction operator is unitary and we can simply reverse the order of the gates in the subtraction circuit and take their inverse to convert it into an addition circuit. Nevertheless, using this implementation does not change the time complexity of our arithmetic operation circuit. Next we will try another approach to implement an addition circuit.

4.2 QFT Approach

In our next approach, we want to take advantage of the quantum fourier transformation. This algorithm was proposed in [11]. Before we discuss its benefits, we first want to illustrate, how addition works in the frequency domain. To add the two numbers represented by $|a\rangle$ and $|b\rangle$ in the frequency domain, we apply a phase rotation by $e^{\pi i 2^{-(n_{max}+1-n-m)}}$ to the n-th qubit of $|a\rangle$ to increment $|a\rangle$ by 2^m (corresponds to adding the m-th qubit of $|b\rangle$ to $|a\rangle$), where n_{max} is the index of the most significant qubit of $|b\rangle$ and $n \in [0, n_{max}+1]$, $m \in [0, n_{max}]$.

When it comes to analysing the complexity of the QFT-adder, it is important to note that it only uses single controlled gates (including the QFT operation), which can be implemented in constant time.

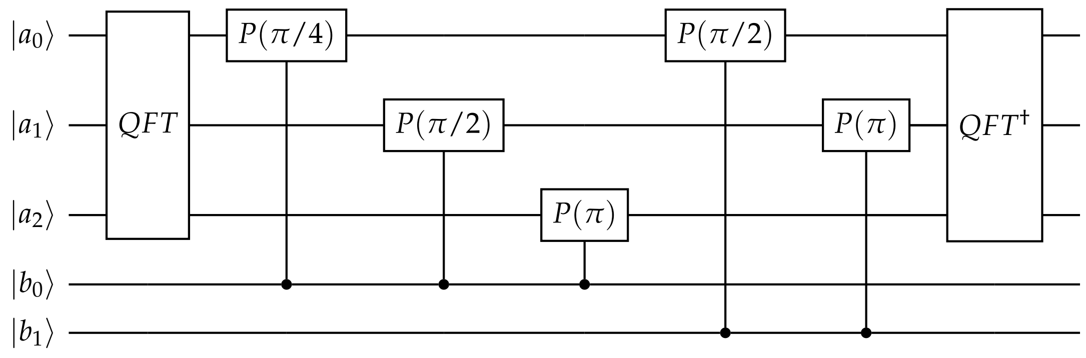

Figure 4.2: QFT adder circuit

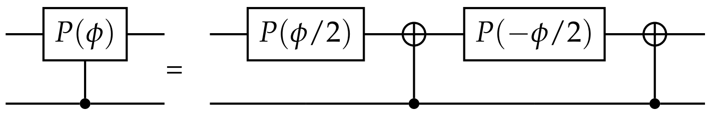

Figure 4.3: decomposition of a controlled phase gate

Given that the effect of each $|b_m\rangle$ ($m \in [0, n_{max}]$) on $|a_n\rangle$ ($n \in [0, n_{max}+1]$) is implemented via a controlled-phase gate, it is clear that the complexity of implementing the addition of two k-qubit numbers is in $\mathcal{O}(k^2)$. The QFT and $QFT^\dagger$ operations at the beginning and at the end both have a very similar construction to the actual addition part in the middle and also have a complexity of $\mathcal{O}(k^2)$. So the complexity of the whole circuit remains $\mathcal{O}(k^2)$. Even if it does not change the complexity of the circuit, it is important to note that for $n, m \in [0, n_{max}]$, if $n + m > n_{max} + 1$ holds, the term $e^{\pi i 2^{-(n_{max}+1-n-m)}}$, which gives us the applied rotation to $|a_n\rangle$ by $|b_m\rangle$, evaluates to an integer multiple of $e^{2\pi i}$. This means that in these cases we only apply full rotations to the target qubit $|a_n\rangle$. Since full rotations do not affect the state our quantum system is in, we can simply leave them out, which is already done in 4.2. The same thing applies to QFT and $QFT^\dagger$ as well.

Also we want to draw attention to the fact that $|a\rangle$ consists of $k+1$ qubits instead of k qubits. Because the addition of two k-qubit numbers results in a $k+1$-qubit number in general. So in the example above, $|a_2\rangle$ is used as a carry qubit.

An Attempt to Improve the QFT Adder

As a result of the above explained addition rules in the frequency domain, we see that incrementing a number in the frequency domain by one rotates the least significant qubit around its z-axis by $e^{\pi i 2^{-(n_{max}+1)}}$. So it completes a full rotation after $2^{n_{max}+1}$ incrementations. This implies that for each number that can be uniquely represented using $n_{max}+1$ qubits, the least significant qubit also has a unique phase. This brings the question to mind, whether it is necessary to apply phase rotations to each of the qubits or it is enough to only rotate the least significant qubit. If we can eliminate all the phase rotations on the qubits other than the least significant one, we could improve the complexity of our adder to $\mathcal{O}(n_{max}^2)$. It is true that we can encode the whole information about a multi-qubit number in a single qubit, but the actual question that need to be answered is whether this information can be efficiently used. As It will be shown in the next section, we need to apply some operation on an ancillary qubit controlled by the outcome of the arithmetic operation to implement a rotation around the target state for Grovers's search. Nonetheless, we cannot trivially implement a phase-controlled unitary. One could try to map the desired phase angle to the $|1\rangle$ state to then use it for controlling the unitary. This could e.g. be done with the operator $HR_z(\theta)$, where θ is the phase angle of the least significant qubit corresponding to the desired number. This would however map most of the non-target states to a superposition of $|0\rangle$ and $|1\rangle$, meaning that the non-target states also have a non-zero probability of being amplified. What we need instead is a unitary that maps the desired phase angle to the $|1\rangle$ state and every other state to the $|0\rangle$ state. However, this operation would not be bijective. Therefore such an operation cannot be unitary.

We could try to bypass this problem by mapping each phase angle that our qubit might have to a unique multi-qubit number and using all of the qubits of the resulting number to control the unitary. However we can quickly realize that using these resulting numbers correspond to using the actual sums that would come out, if we were to use the QFT adder directly. Moreover, deriving these number from the phase of a single qubit would require employing quantum phase estimation, which has $\mathcal{O}(2^{n_{max}+1})$ computational complexity and requires $\mathcal{O}(n)$ auxiliary qubits [12].

As a result, even if we have not shown that QFT adder has optimal complexity, we have shown that one cannot make use of the information encoded in a single qubit efficiently using unitary operators. Although we know that one can also implement non-unitary quantum operators, e.g. LCUs [13], they will not be further investigated for implementing an arithmetic operator in this thesis.

4.3 Implementing the Grover's Search

Now that we have discussed two ways of implementing quantum addition/subtraction, we now want to implement the two crucial operations for the Grover's search, which are the inversion about the target state and the inversion about the initial state. At first we will consider a single constraint over two numbers. Then we will discuss the combination of constraints.

4.3.1 Grover's Search with single constraint over two numbers

We want to start with the implementation of the inversion about the target state. For our Kakuro case, the target is nearly always a superposition, since we have multiple target states, except for the case when the sum is 0. First we want to try an intuitive approach and evaluate its results. The QFT-adder circuit that we discussed will be of use for this part. On a mathematical level, we will use the addition unitary U_{add}, a phase unitary U_p and a unitary U_s.

$$U_{add}\,|ab\rangle = \sum_{n=0}^{2^{k_a}-1}\sum_{m=0}^{2^{k_b}-1} \langle nm|ab\rangle\,|n+m\rangle \otimes |m\rangle \tag{4.6}$$

$$U_s\,|ab\rangle = |(a + (2_a^k - s - 1))\rangle \otimes |b\rangle \tag{4.7}$$

$$U_p\,|ab\rangle = \langle 1^{\otimes k_a}|a\rangle\,(-\,|ab\rangle) + (1 - \langle 1^{\otimes k_a}|a\rangle^2)\,|ab\rangle \tag{4.8}$$

In the formulation above, s in the ket $|sb\rangle$ stands for the desired sum, $|n\rangle$ and $|m\rangle$ are the basis vectors of $|a\rangle$ and $|b\rangle$ and finally of k_a and k_b are the number of qubits in $|a\rangle$ and $|b\rangle$ respectively. With the intuitive assumption that our target unitary U_{-t} inverts the phase of every solution state and leaves the non-solution states as they are, one can construct U_{-t} as a combination of the unitaries above:

$$U_{-t} = U_{add}^{\dagger} U_s^{\dagger} U_p U_s U_{add} \tag{4.9}$$

The unitary U_s can be trivially constructed by applying an x-gate to $|a_n\rangle$, if the n-th bit is a 0 in the binary representation of the given sum s. For the unitary U_p however, we use an ancillary qubit to perform the phase rotation.

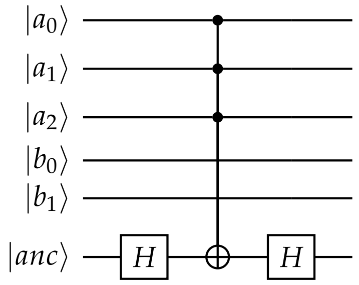

Figure 4.4: U_p unitary

So the unitary that we implement performs the transformation

$$U_p = |1^{\otimes k_a}\rangle\langle 1^{\otimes k_a}| \otimes \mathbb{1}_{k_b} \otimes Z + \sum_{[a]\setminus 2^{k_a}-1} |n\rangle \otimes \mathbb{1}_{k_b} \otimes \mathbb{1}_1 \tag{4.10}$$

where $[a]$ is the set of all possible values of a. At first, it might seem like one should be able to implement the circuit without the ancillary qubit by applying the multi controlled z-gate to one of the qubits of $|b\rangle$. But if we further investigate the effect of the circuit, we see that only the states experiencce a negation of their amplitude, which have the target qubit of the multi controlled z-gate in the $|1\rangle$ state. However, for a chosen target qubit in $|b\rangle$ we cannot guarantee that this target qubit will be in the $|1\rangle$ state for all of the solution(target) states, which means that we could unintentionally leave some of the target states out.

For the next part, we want to implement the "inversion about the mean" operator of the Grover's search algorithm. This unitary operation commonly performs the transformation $U_m = e^{i\pi|\phi\rangle\langle\phi|}$ where $|\phi\rangle = \frac{1}{\sqrt{N}}\sum_{i\in S}|i\rangle$ is the equal superposition, S denoting the set of basis states of the system and $N = |S|$. This implementation assumes that the initial state is $|\phi\rangle$. For the implementation of this operator, we start by using a scaled version of the circuit in [14]. This circuit can be decomposed into five parts as follows, in order to better understand

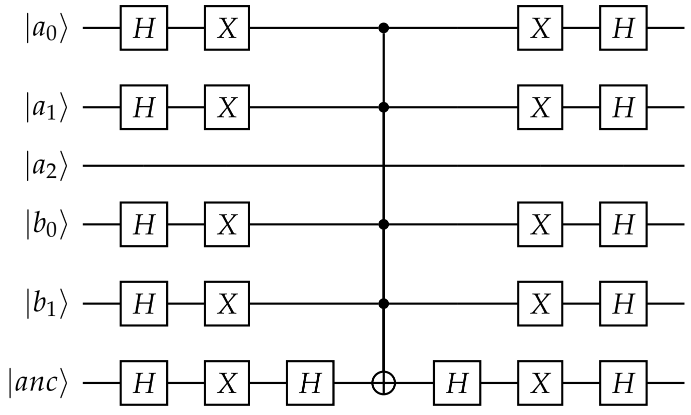

Figure 4.5: the U_m operator

what it does:

$$U_m = H'X'U_{p'}X'H' \tag{4.11}$$

$$H' = H^{\otimes n^*} \otimes I \otimes H^{\otimes n^*+1},\ X' = X^{\otimes n^*} \otimes I \otimes X^{\otimes n^*+1}$$

Notice that $|a_{n_{max}}\rangle$ is ignored for U_m, since it is only an auxiliary qubit used to temporarily hold the value of the most significant bit of the sum during the $|U_{-t}\rangle$ operation. It starts in the $|0\rangle$ state and stays in the $|0\rangle$ state after the aplication of the U_{-t} operator.

Since it is not trivial to apply a phase flip to an equal superposition, we first want to map this state onto another space, where the mapped state is simpler to apply a phase flip to. This is done by the first set of Hadamard-gates, which map the state $|\phi\rangle$ to the state $|0\rangle^{\otimes n_{tot}}$. Even if we also do not have a trivial circuit to implement a unitary that flips the phase of the $|0\rangle^{\otimes n_{tot}}$ state, we can slightly modify the circuit of the U_p unitary above to create a circuit that implements a phase flip of the state $|1\rangle^{\otimes n^*}|.\rangle|1\rangle^{\otimes n^*}$ and acts as the identity on all the other states ($|.\rangle$ indicates that the state of the respective qubit is ignored). This is done by making the multi controlled x-gate also depend on the qubits of the state $|b\rangle$. Then all we need to do before applying this new unitary $U_{p'}$ is to map the state $|0\rangle^{\otimes n_{tot}}$ to the state $|1\rangle^{\otimes n^*}|0\rangle|1\rangle^{\otimes n^*}$, which is easily done by applying a set of X-gates. After we apply the phase flip to the state $|1\rangle^{\otimes n^*}|0\rangle|1\rangle^{\otimes n^*}$, we use another set of x-gates followed by Hadamard-gates to map our state back onto the space, where we initially started. Finally the Grover iterate can be constructed by simply stacking these two unitaries.

$$U_g = U_m U_{-t} \tag{4.12}$$

Now we have implementations for both of the unitaries that are needed to perform Grover's search. The only thing left is the preparation of the initial state. In Grover's search, the initial state is chosen to be an equal superposition, which can easily by reached using a set of Hadamard-gates. This choice has the significance that we can guarantee the projection $\langle T|\phi\rangle$ to be equal to $\sqrt{\frac{1}{N}}$, T being the only target state. Knowing the amplitude of this projection has great importance from the QSP point of view. Because Grover's search algorithm, much like the other quantum amplitude amplification algorithms, applies quantum signal processing to the initial state to amplify the probability of the target state. Since the amplitude of the target state after the QSP process becomes a polynomial of the overlap between the target state and the initial state, knowing the initial overlap lets us choose the right QSP polynomial.

It is however important to keep in mind that the number of solutions to our constraints is mostly greater than one and more importantly variable. So with the Grover's algorithm we cannot expect an optimal solution for all the constraints. Despite this fact, we will proceed with the implementation and then analyse how well it performs.

This is a good time to realize the connections to the QSP sequence that we have previously introduced. First of all, notice that we have shown the initial overlap with the target state as $\langle\phi|T\rangle$ instead of $\langle\phi| U |T\rangle$. The reason for this is the fact that our initial state $|\phi\rangle$ already lives in the target qubit-space. So the unitary U that was supposed to map our initial state to some state in the target qubit-space corresponds to the identity operator and can be ignored. In that case, the U_m operator corresponds to the $e^{i\phi_n|I_0\rangle\langle I_0|}$ operator in 3.5 and the U_{-t} operator corresponds to the $e^{i\phi_m|T_0\rangle\langle T_0|}$ operator analogously, where all the rotation angles ϕ_n and ϕ_m are equal to π(not to be confused with our initial state $|\phi\rangle$). However, one could also interpret the initial state to be $|0\rangle^{\otimes 2n}$ and $U|I_0\rangle$ to be $|\phi\rangle$. This would mean that $e^{i\phi_n|I_0\rangle\langle I_0|}$ corresponds to the middle part of the U_m unitary without the Hadamard-gate layers, whereas each Hadamard-gate layer in our circuit would correspond to an application of the unitary U in 3.5. With that we see that whatever interpretation we choose, one can find a one-to-one correspondence between our circuit and the QSP sequence that we introduced in the previous section.

Next, we want to reconsider the implementation of the U_p unitary, since we have not considered the initialization of the ancillary qubit. As we have shown in 3.4, for amplifying the target states amplitude optimally, one needs $\frac{\pi}{4}\sqrt{\frac{N}{T}} - 0,5$ Grover iterations, T being the number of basis states that satisfy our constraint and N being the total number of basis states of the system. In the following we will analyse how this iteration number can be optimized by slightly altering the U_p unitary.

Suppose that the target sum can be constructed in T_0 different ways using $|a\rangle$ and $|b\rangle$, which each have n qubits (ignoring the auxiliary qubit used tho store the most significant bit of the sum). The number of basis states N_0 of this 2n-qubit system can then be written as 2^{2n}. We will denote the set of basis states of this system with $S_0 = \{|x\rangle \,|x \in [0, 2^{2n} - 1]\}$. Further, we denote the constraint satisfying set of states $|ab\rangle$ S_{T_0}, which is a subset of S_0. When we add an ancillary qubit to this system, the number of the basis states doubles. We denote the new set of basis states as $S = \{|x\rangle \,|x \in [0, 2^{2n+1} - 1]\}$ On the other hand, by the current definition of the unitary U_p, U_{-t} adds a negative phase to the states $|ab\rangle \otimes |1\rangle$, where the numbers represented by the state $|ab\rangle$ satisfies the condition $a + b = c$ and acts as the identity on all the other states. We call the set of the phase flipped states S_{T_1}, which is our new set of target states by the definition of U_{-t}. Here, one can see that $|S_{T_1}| = |S_{T_0}| = T_0$. This means that the number of Grover iterations that we need to optimally amplify the target states is $\frac{\pi}{4}\sqrt{\frac{2N_0}{T_0}} - 0,5$.

Now we consider a slightly modified version of U_p, which first maps the $|+\rangle$ state of the ancillary qubit to the $|1\rangle$ state, then applies the multi-controlled-z gate and finally maps the ancillary-qubit back into its initial state, much like we did for the U_m unitary. The circuit for the modified unitary $U_p^{'}$ would look like the following:

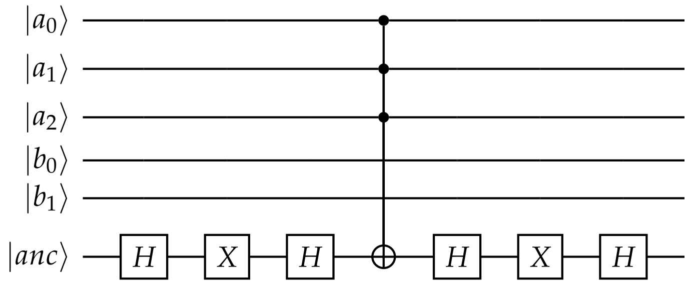

Figure 4.6: the $U_p^{'} operator$

This time we choose our set of basis states as $S^{'} = \{|x\rangle \otimes |y\rangle \,|x \in S_0, y \in \{+, -\}\}$. Notice that our initial state is no longer an equal superposition over the basis states $S^{'}$, since the ancillary qubit is in the state $|+\rangle$ with certainty. Our new unitary $U_p^{'}$ adds a negative phase to the states $|ab\rangle \otimes |+\rangle$ and acts as the identity on all the other states. We call the set of new target states as $S_{T_1}^{'}$. Also in this case we find that $|S_{T_1}^{'}| = T_0$. An important point here is that the amplitude of every state, where the ancillary-qubit is in the $|-\rangle$ state remains zero throughout the operators $U_{-t}^{'}$ and U_m, where $U_{-t}^{'}$ is the U_{-t} initary that uses the $U_p^{'}$ unitary instead of U_p. So we can actually consider our quantum system to be defined only over the basis states $\{|x\rangle \otimes |+\rangle \,|x \in S_0\}$, which has carnality N_0. With this realization we conclude that for this case the optimal number of Grover iterations is $\frac{\pi}{4}\sqrt{\frac{N_0}{T_0}} - 0,5$, which indicates an asymptotical speedup by a factor of $\frac{1}{\sqrt{2}}$.

When we repetitively apply the iteration $U_{-t}^{'}U_m$, we realize that many X-gates and Hadamard-gates applied to the ancillary-qubit cancel each other out. If we then analyse the resulting circuit, we see that it corresponds to initialising the ancillary-qubit to the $|1\rangle$ state and using the U_{-t} operator and the $U_m^{'}$ operator instead of $U_{-t}^{'}$ and U_m, where

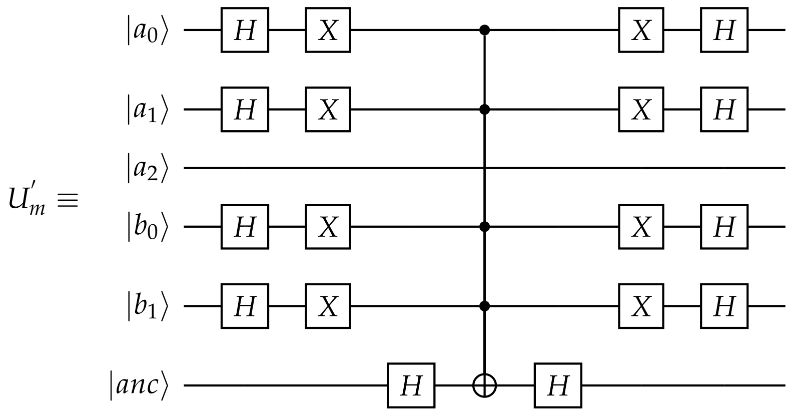

Figure 4.7: the $U_m^{'}$ operator

rotates around the new initial state $|+\rangle^{\otimes n} |.\rangle \, |+\rangle^{\otimes n} \otimes |1\rangle$. So we choose to initialise our circuit as follows, contradicting with the commonly used initialisation for the Grover's algorithm.

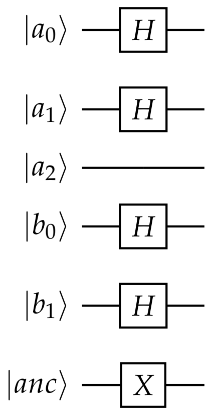

Figure 4.8: initialization of the circuit

However, even in a single-constrained Kakuro with two variables, we cannot calculate the exact number of iterations using this formula. Because even for this minimal case multiple solutions might exist. So what we want to calculate instead is the number of iterations r_g that yields the minimum expected squared error for n qubits, assuming that the number of target states M is equally distributed in some interval I.

$$r_g(I,n) = \min_r \sum_{m \in I} \left(1 - \langle T_m| \, G^r \, |I_0\rangle\right)^2 \tag{4.13}$$

Here $|T_m\rangle$ represents a target state consisting of an equal superposition of m unique solutions and n is the total number of qubits, excluding any ancilla/auxiliary qubits. How n comes into play will be clear later, when we use the expended version of this formula on a specific case. But before we make use of this formula, we first want to explore how Kakuros with multiple constraints can be implemented and than evaluate the performance of different algorithms in case of a two-by-two Kakuro.

4.3.2 Implementing Kakuros with multiple constraints

In this section, we will discuss the combination of multiple constraints. One can represent a Kakuro as an AND concatenated set of sums. Remember that in the case with a single constraint, we realized the inversion of the phases of the target states with an n-controlled x-gate, controlled by the state $|a + b + (2^{2n} - s - 1)\rangle$ and we built this state on the qubits $|a_m\rangle$ $m \in [0, n-1]$. If we want to add more constraints to our Kakuro, we have to form the states that represent each of the constraints (sums) simultaneously, so that we can realize the phase flips with (k*n)-controlled x-gates, k being the number of constraints of our Kakuro. Without using any more ancillary registers, this means that our implementation can only solve Kakuros that have at most so many constraints as they have variables. However, in the following we will prove that a Kakuro can never have more constraints than it has variables.

Lemma 1. *In a Kakuro, each set of variables with size > 1 can participate in a maximum of one constraint.*

Proof. In a Kakuro, a constraint is formed by consecutive variables iff. they are all on the same axis (see example in the first chapter). With this in mind, we assume that our Kakuro only has the constraint a + b = x and that the variables are on the vertical axis. Then out Kakuro has to look like the following:

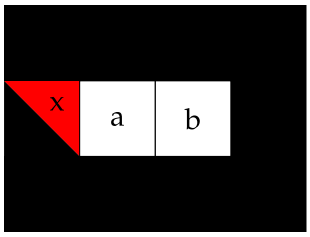

Figure 4.9: a Kakuro with a single constraint over two variables

As it can be seen, the red triangle is reserved for the sum x. Now we assume that our Kakuro also has the constraint a + b + c = y. This would imply that our Kakuro looks like the following:

As one can see, the y constraint leaves us with no possibility of representing the x constraint on our Kakuro, since the spot marked with the red triangle now has to refer to the sum of all the consecutive variables to its right. □

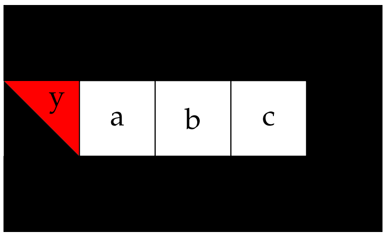

Figure 4.10: a Kakuro with a single constraint ove three variables

Another important realization results from the rule that sums can only be formed by variables that are all on the same vertical axis or by the variables that are all on the same horizontal axis. Since there is only a single vertical axis and a single horizontal axis going through each variable, this implies that each variable can only participate in a maximum of two constraints.

For the next step, we will introduce a notation for representing the constraints. In this notation each constraint is represented by a k-dimensional vector, where k is the number of constraints in our Kakuro. The n-th element of the vector indicates weather or not the n-th variable of the Kakuro participates in the constraint. Then we can stack the transposed constraint vectors to form a matrix.

Because we want to prove a statement about the maximum possible number of constraints, we will assume that all the constraints of our Kakuro contain two variables, since with lemma 1 we showed that using longer constraints eliminates the possibility to add certain shorter constraints, resulting in a fewer total number of constraints. For example, adding the constraint $(1,1,1)^T$ eliminates the chance to add the two constraints $(1,1,0)^T$ and $(0,1,1)^T$.

If we want to use each variable as many times as possible, we can define the constraints e.g. according to the following function:

$$c_n(x) = \begin{cases} 1 & n - x \mod n < 2 \\ 0 & \text{otherwise} \end{cases}, \; x \in [0, N_v - 1], \; n \in [1, N_v] \tag{4.14}$$

Here, the function c_n indicates the x-th elementh of the n-th constraint and N_v is the number of variables in our Kakuro. Note that these assignments assign each variable $v_x = 1$ twice and thus give us a maximal constraint set. It is also clear that the number of constraints is equal to the number of the functions c_n, which is N_v. With that, we have shown that a Kakuro can never have more constraints than it has variables. This means that we will always have enough registers to store the desired sums, which can then be used to control the x gate of the U_{-t} unitary. On the other hand, it still needs to be shown that one can find a series of gates that efficiently bring the initial state to the state $|s_1 s_2 ... s_k\rangle$, which is the state of desired sums.

To prove that the state $|s_1 s_2 ... s_k\rangle$ can always be prepared, starting from the initial state

$|v_1 v_2 ....v_N\rangle$, we want to take take advantage of the above mentioned constraints matrix notation. Moreover, we want to use the fact that the same notation can be used to represent the states of our registers as well. In this notation, the initial state would be equivalent to the constraint matrix $\mathbb{1}$, which is the identity matrix. So what we are looking for is a way to transform the identity matrix into the desired constraint matrix.

Now, instead of the transformation from $\mathbb{1}$ to the constraint matrix C, we want to consider the inverse transformation from C to $\mathbb{1}$. We can easily realize that this transformation is equivalent to solving a linear system of equations. So, if we can find an efficient algorithm for this transformation, we can then reverse each operation to transform $\mathbb{1}$ into C.

For our purpose, we can use Gaussian elimination, which is known to have the complexity $\mathcal{O}(N_v^3)$. However, taking the structure of our Kakuro into account, we can slightly modify this complexity. Let us consider any $N_v x N_v$ rectangular matrix with full rank. To solve a system of liner equations of this form, we need to add multiples of $\mathcal{O}(N_v)$ different rows to find the value of each variable. Given that we have N_v variables and each row addition also requires addition of $\mathcal{O}(v)$ variable, it is easy to see that this algorithm should run in $\mathcal{O}(N_v^3)$ time. On the other hand, for our Kakuro problem each row represents the state of a register/variable. In this case adding two rows takes only a constant time. So if we have our registers in a state that is represented by some constraint matrix C, we need $\mathcal{O}(N_v^2)$ steps to bring every register in a state such that each register represents a single variable. But this also implies that starting from a state where each register holds the value of a respective variable, we need $\mathcal{O}(N_v^2)$ steps to create any desired constraint matrix.

We must remember that the above mentioned algorithm will be used to realize the U_{-t} unitary, which will be used $\mathcal{O}(\sqrt{\frac{a^{N_v}}{b}})$ times for the Grover's search algorithm, a representing the number of values each variable can take and b representing the number of solutions of our Kakuro. This yields an overall complexity of $(N_v^2\sqrt{\frac{a^{N_v}}{b}})$ for the whole search.

Now we want to compare this to the complexity of a classical Kakuro solver. As a classical solver, we can again use the Gaussian elimination. But this time we actually want to transform the constraint matrix C concatenated with the vector s of desired sums. In this case, one might think that running Gaussian elimination on this matrix takes $\mathcal{O}(N_v^3)$ steps, since this time adding two rows takes $\mathcal{O}(N_v)$ time. On the other hand, our complexity matrix always consists of only zeroes and ones and is defined by the shape of our Kakuro. So once we determine the needed sequence of row additions, we can just omit the matrix C and only apply the sequence to the vector s for any vector s. This means that classically we can solve a Kakuro in $\mathcal{O}(N_v^2)$ steps, if its constraint matrix has full rank. If this is the case, we cannot improve the performance of a classical solver with our quantum solver, because in each Grover iterate we need as many steps as we would need to completely solve the Kakuro classically. However, at this point we would like to remind that while calculating these costs we are ignoring the rule that each sum may include each number only once. We also allow assigning 0 to variables.

Taking these rules into account would completely change the calculated costs.

So far, we have shown that Grover's search can be implemented without using any extra registers to calculate the sums, based on the assumption that we have a Kakuro with N_v linearly independent constraint vectors, which can be concluded from the fact that we used an $N_v x N_v$ constraint matrix with full rank for the proof of the complexity of our algorithm. However we have not improved over the performance of a classical Kakuro solver for this case. Nevertheless, most of the time Kakuros have constraints matrices with rank less than N_v. Luckily implementing the U_{-t} unitary without any auxiliary registers is possible, even if the Kakuro is not fully-constrained, which will be proven in the following.

Kakuros with less than N_v constraints

Before we start with the proof, note that a Kakuro might still have a single solution, even if it has less than N_v constraints. Because, what we mean by constraint is a sum over some variable of the Kakuro that has to be met. However our Kakuros are constrainted with more than the desired sums. Because the variables cannot take any integer value, since they all have a certain bit-depth. For example, if we have a Kakuro consisting only of two 3-qubit variables and we want their sum to be equal to 14, then the only possible valuation of these two variables is that they are both equal to 7. Meanwhile, even if we have N_v constraints (N_v sums) in our Kakuro, it might have multiple solutions. Because some of the sums in our Kakuro might be uniquely determined by the other sums. To exemplify this case, we will use the following Kakuro shape and naming convention.

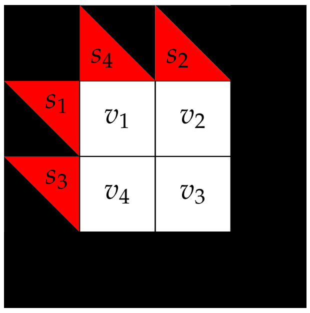

Figure 4.11: a 2-by-2 Kakuro

Now we consider a Kakuro, where $s_1 = s_2 = s_3 = s_4 = 3$. This Kakuro has the two solutions $v_1 = v_4 = 1$; $v_2 = v_3 = 2$ and $v_1 = v_4 = 2$; $v_2 = v_3 = 1$. The fact that this Kakuro has multiple solutions implies that its constraint matrix has a lower rank than 4. The constraint matrix of this Kakuro can be written as follows:

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} \tag{4.15}$$

The last row of this constraint matrix can be written as a linear combination of the other three rows by adding up the first and the third row and subtracting the second row. Therefore this matrix does not have full rank and we can omit the last row. A notable statement that derives from this Gaussian eliminated matrix is that the above mentioned Kakuro is equivalent to the one below. This means that the following proof applies to any Kakuro with a constraint matrix with rank less than N_v even if it seems to have N_v constraints.

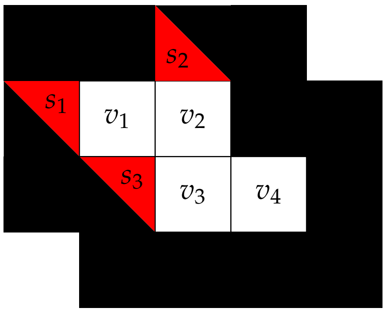

Figure 4.12: simplified version of a 2-by-2 Kakuro

Moving on with the actual proof for Kakuros with less than N_v constraints, let us consider some constraint matrix C with some rank $r < N_v$. We start by eliminating the extra rows of C if necessary. Then we rearrange the columns such that the upper left rxr matrix has full rank. This step corresponds to renaming the variables of our Kakuro. We need to store only r sums, in order to implement the U_{-t} unitary. We assuming that all of the r sums are stored in the first r registers and the rest of the $N_v - r$ registers hold the value of the respective variables $v_{N_v - r+1}$ to v_{N_v}. Now if we were to visualise this state of our Kakuro in a constraint-matrix-like notation, it would look like the following:

$$\begin{pmatrix} & C & \\ 0_{N_v - rxr} & & \mathbb{1}_{N_v - rxN_v - r} \end{pmatrix} \tag{4.16}$$

Here $0_{(N_v - r)\text{x}r}$ represents a $(N_v - r)$ xr zero-matrix and $\mathbb{1}_{(N_v - r)\text{x}(N_v - r)}$ represents the $(N_v - r)$ x $(N_v - r)$ identity matrix. Also note that the part of the matrix below C does not represent any constraint, it only represents the state of the Kakuro. Now we can add and subtract the rows as we did previously and this would correspond to adding and subtracting the registers from one another.

Next we subtract the row r_x for $x \in]N_v - r, N_v]$ from each of the previous row, which has the x-th element set to 1. After this step, we end up with the matrix

$$\begin{pmatrix} C_m & 0_{rx(N_v-r)} \\ 0_{(N_v-r)xr} & \mathbb{1}_{(N_v-r)x(N_v-r)} \end{pmatrix} \tag{4.17}$$

where, C_m is the left rxr part of the C matrix. We know that this matrix has rank r, since it was left untouched by the last step. At this point, we can use Gaussian elimination to bring it to a diagonal form. This further implies that we can reverse each operation that we made to the initial matrix, just like we did with the Kakuros with full-ranked constraint matrices, to transform an N_vxN_v identity matrix into the matrix that we started with, which block encodes the C matrix.

With that, we have shown that U_{-t} unitary can be implemented with N_v registers for any Kakuro. The complexity for the for the case $r < N_v$ can be calculated in two parts. For the first part, we need $\mathcal{O}(r^2)$ steps to form the partial sums in the first r register. For the second part, we need to add the $N_v - r$ missing values on top of the sums in the first r registers. This step takes $\mathcal{O}(2\,(N_v - r))$ steps, since each of the last $N_v - r$ variables can take part in a maximum of 2 constraints. Assuming that r is close to or at least linear in N_v, we can conclude that we can implement the U_{-t} unitary in $\mathcal{O}(N_v^2)$ steps and find a solution for the Kakuro in $\mathcal{O}\left(N_v^2\sqrt{\frac{a^{N_v}}{b}}\right)$ steps for any Kakuro.

Now it remains to calculate the complexity of solving a Kakuro classically. For the case that its constraint matrix has a lower rank than r. This case differs from the case with rank N constraint matrices, because we can no longer use Gaussian elimination or any other analytical approach to find a particular solution for the Kakuro. At best, we can reduce the number of constraints in the Kakuro. But then, the problem degenerates into a search in a tree with backtracking, with the help of all the constraints in our Kakuro (both the ones in the constraint matrix and the ones that result from the allowed set of values for each variable). This implies that the worst-case complexity of a classical Kakuro-solver is $\mathcal{O}(\frac{a^{N_v}}{b})$, a and b having the same meaning as before.

We have already proven that our implementation of Grover's search has asymptotically a quadratic speedup against a classical Kakuro-solver. Even though this is the best asymptotic speedup that we can achieve, this is not yet the most time efficient implementation that we can achieve. There is still a trade-off between time and space complexity that we can do. Namely, if we choose to use r auxiliary registers, we can use these to each hold one of the needed sums. Since each variable can participate in a maximum of two sums, we need $\mathcal{O}(2N_v)$ steps to create the sums.

We choose the space efficient version with no auxiliary registers for our implementation. Nevertheless, we must remember that adding multiple n-qubit numbers in general results in a number that is represented by more than n qubits. So if we want to use the registers that are reserved for the variables to form the sums on, we need to append an auxiliary qubit to each of them as we did to $|a\rangle$ in the example with two variables and a single constraint.

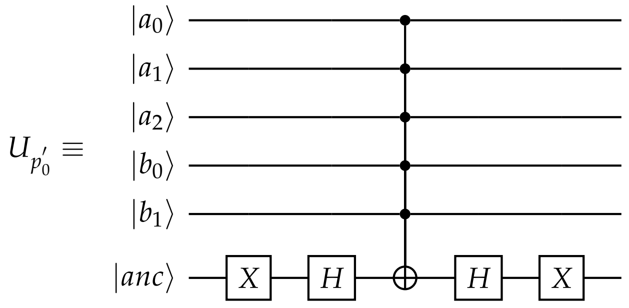

Figure 4.13: implementation of the $U_p^{'}$ unitary that flips the phase of the $|1^{\otimes n}\rangle\,|0\rangle$

Theoretically, appending only a single auxiliary qubit to each variable could still result in an overflow while calculating the sums. Because in a Kakuro a constraint could consist of up to N_v variables instead of just two. However appending a single auxiliary qubit to each variable will suffice for the case that we will investigate.

4.4 Implementing the Partial Diffusion Algorithm

Next we want to implement the partial diffusion algorithm for the two-by-two Kakuro case. Having already talked about the similarities between this algorithm and the Grover's algorithm, one can realize that Grover's algorithm can be transformed into the partial diffusion algorithm by making only three changes to the circuit:

1. adapt the initialization of the system
2. remove the Hadamard gates on the ancilla qubit surrounding the multi-controlled X-gate in the U_{-t} operator (we call the resulting operator U_{t1})
3. swap the $U_p^{'}$ unitary in $U_m^{'}$ with the unitary $U_{p_0^{'}}$ as it is implemented in figure 4.13 (we call the resulting operator U_{m0})

The first change is trivially done by removing the X-gate on the ancilla qubit during the initialisation, since the partial diffusion algorithm was defined as the ancilla qubit starting in the $|0\rangle$ state.

For the second point, removing the Hadamard gates on the ancilla qubit means that we can no longer consider U_{t1} to be a z-rotation in the $|T_0\rangle$, $|T_\perp\rangle$ space, but it rather serves the entanglement of the ancillary qubit with the number registers.

Finally in the third point, changing $U_p^{'}$ with $U_{p_0^{'}}$ has the effect that the U_{m0} operator now flips the phase of a state if the qubits representing the numbers are in an equal superposition and the ancilla qubit is in the $|0\rangle$ state, whereas in the implementation of the Grover's algorithm the ancilla qubit was required to be in the $|1\rangle$ state. Now the only question that might arise is how this change transforms the diffusion operator into a partial diffusion operator. Because if this new operator is a partial diffusion operator that causes an inversion of the amplitudes only in the subspace, where the ancilla qubit is in the $|0\rangle$ state (0-subspace), the old operator should also do the same, only this time in the other subspace, where the ancilla qubit is in the $|1\rangle$ state (1-subspace). So in our implementation of the Grover's algorithm seems to be performing a partial diffusion as well! In fact this is true. However we must keep in mind that the ancillary qubit in our Grover's search implementation has zero probability along the $|0\rangle$ axis after each iteration. So effectively this quantum system exists in a 2^n-dimensional Hilbert space, n being the number of qubits in our circuit ignoring the ancilla qubit, whereas the partial diffusion algorithm uses the whole 2^{n+1}-dimensional Hilbert space. This means that in our Grover's algorithm implementation we only need to consider the effect of the U_m unitary in the 1-subspace, which is that of a (non-partial) diffusion operator. In other words, it is how the whole algorithm uses/evolves the ancilla qubit that makes the difference between the diffusion operator and the partial diffusion operator.

As the last step of the implementation, we need to determine the optimal number of iteration. But just like in Grover's search algorithm, this number depends on the number of solutions states M. So instead, we will use the number of iterations that minimizes the following error function, analogously to the Grover's search algorithm.

$$r_{pd}(I,n) = \min_{r} \sum_{m \in I} \left(1 - \langle T_m| \left(U_{t1} U_{m0}\right)^r |I_0\rangle\right)^2 \tag{4.18}$$

4.5 Implementing Mizel's Critically Damped Search Algorithm

Once again we will implement this algorithm using the unitaries to which we have already given the implementations above. To quickly revise Mizel's algorithm, remember that it has the two major steps "$|T_0\rangle$-controlled y-rotation of the ancillary qubit" (U_{ty} operator) and the "ancillary-controlled Grover iteration" (U_{cg} operator). At first it might not be clear how many ancillary qubits these operators require, since we need a controlled-U_g operator U_{cg}, whereas the implementation of the U_g operator itself also requires an ancilla qubit. To better investigate this let us first decompose U_{cg} into a controlled-U_{-t} operator U_{ct} and a controlled-$U_m^{'}$ operator U_{cm}. In section 3.6 we have already explained that U_{cm} is equivalent to the partial diffusion operator, which does not require a second ancillary qubit. So the question becomes whether or not we need a second ancilla qubit to implement U_{ct}. By definition of U_{ct} there should be an ancillary qubit such that this operator flip the pahse of the $|T_0\rangle$ state iff. this ancilla is in the state $|1\rangle$. Nevertheless we realize that this is exactly the effect of the U_{-t} operator. We only do not experience the full effect of it, since the ancilla qubit in our Grover's algorithm implementation always has zero probability along the $|0\rangle$ axis as we previously mentioned. All in all, this means that we do not need a second ancilla qubit for implementing

Mizel's algorithm. The two operators that we need can be written as such:

$$U_{ty}(\phi) = |T_0\rangle\langle T_0| \otimes e^{-i\frac{\phi}{2}Y} + |T_\perp\rangle\langle T_\perp| \otimes I \tag{4.19}$$

$$U_{cg} = U_g,\ U_g = U_{-t}U_m^{'} \tag{4.20}$$

Notice that we use the U_g operator as it is, without altering the $U_m^{'}$ unitary like we did for the partial diffusion algorithm. This is because for Mizel's algorithm the inversion of the amplitudes about the mean should be performed in the 1-space, and not in the 0-space as it was for the partial diffusion algorithm.

The next thing that we should clarify is the implementation of the $U_{ty}(\phi)$ operator. For this purpose we use the decomposition explained in [10] to rewrite $U_{ty}(\phi)$ in terms of single qubit rotations and U_{-t} operator:

$$U_{ty}(\phi) = I \otimes e^{-i\frac{\phi}{4}Y}U_{-t}I \otimes e^{i\frac{\phi}{4}Y}U_{-t}$$

Finally we measure the ancilla qubit and either terminate the algorithm or keep iterating depending on the outcome of the measurement. However we cannot dynamically modify our circuit during the runtime to add iterations. As proposed in [9], we predetermine a "*judicious*" number of iterations R to solve this issue. We than iterate the algorithm R times, conditioning each of the iterations on the measurement outcome of the ancilla qubit except for the first one. If the ancilla has flipped after R iterations, we measure a target state with certainty. If it has not flipped, the final state of the system still has some probability to be in the $|T_0\rangle|1\rangle$ state. So in the case that the ancillary qubit has not flipped, we make a call to the U_{t1} unitary to mark the target state with the ancilla being in the $|0\rangle$ state. In the end, we measure the system. If the ancilla is in the $|0\rangle$ state, we measure a target state. If not, the algorithm has failed and we start over. However, in our implementation we will simply let the algorithm fail, if we do not measure a target state after the decided number of iterations. In [9], it is also explained that the overhead that this approach requires is a factor of 1,5, which we will use to chose the number of iterations.

Also note that while iterating the algorithm one does not need to condition each gate on the measurement outcome. It suffices to only condition the multi-controlled-Tofolli-gates on the measurement outcome, since for the case that these gates are not effective, all the other gates add up to the identity operator.

4.6 Comparing the Performances

Now that we have implementations for each of the algorithms, we can now test and compare their performances on a specific Kakuro. For our tests, the following Kakuro will be used.

For a small example like this, one can quite easily determine the number of solutions. However for the sake of generality we will ignore this information and assume that the

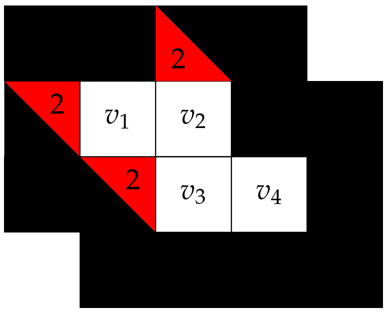

Figure 4.14: test Kakuro

probability of this Kakuro having M solutions is equally distributed for $M \in [1,7]$. For this assumption we calculate optimal number of iterations that yield the minimum expected squared error according to 4.13 and 4.18.

$$r_g([1,7],12) = \min_r \frac{1}{7} \sum_{m\in[1,7]} \left(1 - \langle T_m| G^r |I_0\rangle\right)^2$$
$$= \min_r \frac{1}{7} \sum_{m\in[1,7]} \left(1 - \sin^2\left((r+0,5)\, 2\arcsin\left(\sqrt{\frac{m}{2^{12}}}\right)\right)\right)^2 \approx 24,6$$

$$r_{pd}([1,7],12) = \min_r \frac{1}{7} \sum_{m\in[1,7]} \left(1 - \langle T_m| (U_{t1}U_{m0})^r |I_0\rangle\right)^2$$
$$= \min_r \frac{1}{7} \sum_{m\in[1,7]} \left(1 - \frac{m}{2^{12}} \frac{\sin^2\left((r+1)\arccos\left(1-\frac{m}{2^{12}}\right)\right) + \sin^2\left(r\arccos\left(1-\frac{m}{2^{12}}\right)\right)}{\sin^2\left(\arccos\left(1-\frac{m}{2^{12}}\right)\right)}\right)^2 \approx 35,1$$

Since the iteration number has to be an integer, we round these numbers to the nearest smaller integer. Using these iteration counts results in the following mean success probabilities.

$$P_g(24,[1,7],12) = \frac{1}{7} \sum_{m\in[1,7]} \sin^2\left((24,5)\, 2\arcsin\left(\sqrt{\frac{m}{2^{12}}}\right)\right) \approx 0,842$$

$$P_{pd}(35,[1,7],12) = \frac{1}{7} \sum_{m\in[1,7]} \frac{m}{2^{12}} \frac{\sin^2\left((36)\arccos\left(1-\frac{m}{2^{12}}\right)\right) + \sin^2\left(35\arccos\left(1-\frac{m}{2^{12}}\right)\right)}{\sin^2\left(\arccos\left(1-\frac{m}{2^{12}}\right)\right)} \approx 0,839$$

As one can see, partial diffusion algorithm provides a lower expected success probability compared to Grover's algorithm for yhis assumption of I. Next we simulate the Kakuro in Fig. 4.14 using the "ibmq_qasm_simulator" [15] and the following results are received:

$$\text{Grover's search: } \frac{\#success}{\#trials} = \frac{1934}{2048} \approx 94,4\%$$
$$\text{partial diffusion: } \frac{\#success}{\#trials} = \frac{1957}{2048} \approx 95,6\%$$

Another metric that might be useful for comparing the performances of the both algorithms is the worst case success probability. Using the above given values, the worst case for both of the algorithms arises when $M = 1$. In this case the success probability for Gorver's search is aprox. 48% whereas it is aprox. 49,9% for partial diffusion algorithm. So the partial diffusion algorithm gains a slight advantage against Grover's search in the worst case for the given I.

Before moving on to the results of the Mizel's algorithm, we want to evaluate these two algorithms also for the case $I = [3000, 3030]$, since we would expect partial diffusion algorithm to perform significantly better against the Grover's algorithm in this interval, as we discussed in 3.4.

$$P_g(1, [3000, 3030], 12) = \frac{1}{31} \sum_{m \in [3000,3030]} \sin^2\left((1,5)\, 2 \arcsin\left(\sqrt{\frac{m}{2^{12}}}\right)\right) \approx 0,002$$

$$P_{pd}(1, [3000, 3030], 12) = \frac{1}{31} \sum_{m \in [3000,3030]} \frac{m}{2^{12}} \frac{\sin^2\left((2) \arccos\left(1 - \frac{m}{2^{12}}\right)\right) + \sin^2\left(1 \arccos\left(1 - \frac{m}{2^{12}}\right)\right)}{\sin^2\left(\arccos\left(1 - \frac{m}{2^{12}}\right)\right)}$$
$$\approx 0,941$$

Note that in this example we round up the error minimizing number of iterations, since it is smaller than 1 for both of the algorithms. As it can be seen, partial diffusion algorithm easily outperforms Grover's algorithm's expected success probability for this case. Notably, Grover's search provides a result that is also far worse compared to a naive classical search, whereas partial diffusion still outperforms classical search. Nevertheless, this example is only given for illustration purposes and the maximum amount of solutions a Kakuro with the given structure in Fig. 4.14 is actually 7.

Finally, we will analyse how Mizel's algorithm performs on the given Kakuro. For this approach we determine the number of iterations pessimistically and assume that the Kakuro has a single solution, which results in the highest number of needed iterations. Doing so not only results in a high success probability for the case $M = 1$ but also for the cases that M is higher. Because this is a fixed-point algorithm, meaning that the success probability converges to 1 with increasing number of iterations.

$$r_m \approx 1,5 r_g([1], 12) = 1,5 \frac{\pi}{4 \arcsin\left(\sqrt{\frac{1}{2^12}}\right)} \approx 74,6$$

Running our implementation of the algorithm with 74 iterations results in the following results, once again using the "ibmq_qasm_simulator".

$$\text{Mizel's algorithm: } \frac{\#success}{\#trials} = \frac{2035}{2048} \approx 99,4\%$$

5 Conclusion

The purpose of this thesis was to explore, how existing quantum algorithms from the fields of quantum search and quantum amplitude amplification could be used to accelerate CSPs with arithmetic constraints. For this purpose we focused on the Grover's search algorithm, the partial diffusion algorithm and Mizel's critically damped quantum search algorithm. We expected the first two of these algorithms to perform worse for our Kakuro example, since the exact number of needed iterations cannot be calculated with certainty for these two algorithms. On the other hand we expected Mizel's algorithm to perform significantly better, because of its fix-point nature.

Even if Mizel's algorithm has yielded better results compared to the other two algorithms, Grover's search algorithm and the partial diffusion algorithm also performed remarkably, both yielding around 95% percent success rate for our small scale Kakuro example. This has shown us that all of these algorithms could be used to speed up arithmetic CSPs, given some information about the interval, in which the number of solutions of the CSP lies, while Mizel's algorithm also has scaling potential due to its convergent behaviour.

Speaking for the specific case of Kakuros, in the next step one could combine our work with the previous research in the quantum CSP field that deals with accelerating filtering for "alldifferent" constraints ([1]) and in doing so implement a full Kakuro solver.

List of Figures

Bibliography

[1] K. E. Booth, B. O'Gorman, J. Marshall, S. Hadfield, and E. Rieffel. "Quantum-accelerated constraint programming". In: *Quantum* 5 (2021).

[2] E. Campbell, A. Khurana, and A. Montanaro. "Applying quantum algorithms to constraint satisfaction problems". In: *Quantum* 3 (2019).

[3] T. D. Hansen, H. Kaplan, O. Zamir, and U. Zwick. "Faster k-sat algorithms using biased-ppsz". In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. 2019.

[4] W. Commons. *File:Kakuro black box solution.svg — Wikimedia Commons, the free media repository*. [Online; accessed 13-February-2023]. 2020. URL: `https://commons.wikimedia.org/w/index.php?title=File:Kakuro_black_box_solution.svg&oldid=448276307`.

[5] J. M. Martyn, Z. M. Rossi, A. K. Tan, and I. L. Chuang. "Grand unification of quantum algorithms". In: *PRX Quantum* 2.4 (2021), pp. 3–6, 11.

[6] L. K. Grover. "A fast quantum mechanical algorithm for database search". In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 1996.

[7] A. Younes, J. Rowe, and J. Miller. "Quantum search algorithm with more reliable behaviour using partial diffusion". In: *AIP Conference Proceedings*. Vol. 734. 1. American Institute of Physics. 2004, pp. 7, 13–15.

[8] L. K. Grover. "A different kind of quantum search". In: *arXiv preprint quant-ph/0503205* (2005), pp. 1–8.

[9] A. Mizel. "Critically damped quantum search". In: *Physical review letters* 102.15 (2009), pp. 1–3.

[10] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. "Elementary gates for quantum computation". In: *Physical review A* 52.5 (1995), pp. 18–20.

[11] T. G. Draper. "Addition on a quantum computer". In: *arXiv preprint quant-ph/0008033* (2000).

[12] T. Q. Team. *Quantum phase estimation*. Nov. 2022. URL: `https://qiskit.org/textbook/ch-algorithms/quantum-phase-estimation.html#references`.

[13] S.-J. Wei and G.-L. Long. "Duality quantum computer and the efficient quantum simulations". In: *Quantum Information Processing* 15 (2016).

[14] T. Q. Team. *Grover's algorithm*. Nov. 2022. URL: `https://qiskit.org/textbook/ch-algorithms/grover.html`.

[15] *IBM quantum*. URL: `https://quantum-computing.ibm.com/services/resources`.