Solving the Traveling Salesman Problem with the Quantum Approximate Optimization Algorithm

Jannik Eggert

29.08.2023
Matrikelnummer: 10002369

Gottfried Wilhelm Leibniz Universität Hannover
Fakultät für Mathematik und Physik

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science

unter Anleitung von Prof. Dr. Tobias J. Osborne

Contents

1 Introduction

The Travelling Salesman Problem (TSP) is a well-known problem in the fields of optimization and computer science. The problem is simple to describe: Given a list of cities, find the shortest possible route that visits each city exactly once and returns to the starting city. Despite its straightforward description, the TSP is a complex problem with many applications in science and industry.

The TSP has a long history, dating back to the 1800s. The term "Travelling Salesman Problem" became popular in the 1950s. [23] Over the years, the TSP has become a standard problem for testing optimization and computational methods. Many different mathematical, algorithmic, and heuristic strategies have been proposed to solve the TSP. However, finding the shortest route through a large number of cities remains a challenging task.

The TSP has many practical applications in industry and commerce, especially in logistics, transportation, and supply chain management. The growth of a need for efficient transportation have increased the demand for optimal routing solutions. Finding a better approach to solve the TSP can help minimize transportation costs and reduce the environmental impact of transportation activities.

In science, the TSP has applications in many fields such as bioinformatics, molecular dynamics simulations, chip design, and astrophysics. For example, the TSP is used in bioinformatics to help sequence DNA fragments and study protein folding. [3] [5]

From a computational standpoint, the TSP is classified as a problem in the complexity class NP. This means that the problem is hard to solve in a reasonable amount of time as the number of cities increases. Specifically, it is easy to check whether a given solution is correct, but it is hard to find the optimal solution in the first place.

Over the years, many algorithms have been proposed to solve the TSP. One of the earliest is the brute-force approach, which looks at all possible routes to find the shortest one. However, this method is impractical for large instances due to its factorial time complexity. Many other algorithms have been proposed, including exact algorithms like the Held-Karp algorithm and heuristic algorithms like the nearest neighbor algorithm. Exact algorithms guarantee the optimal solution but may need an unfeasible amount of time for large instances. Heuristic algorithms provide approximate solutions quickly but do not guarantee optimality.

Recent algorithms for the TSP use machine learning techniques. For example, reinforcement learning-based approaches and deep learning methods have shown promise in approximating TSP solutions. These methods learn the structure of the problem instances and make informed decisions during the routing process. However, they have limitations in terms of generalization, computational requirements, and the lack of optimality guarantees.

Recent advancements in quantum computing have opened up new avenues for solving optimization problems, including the TSP. Quantum algorithms leverage the principles of quantum mechanics to solve problems that classical computers struggle with. One such quantum algorithm is the Quantum Approximate Optimization Algorithm (QAOA)

The Quantum Approximate Optimization Algorithm is an algorithm that uses the

advantages of quantum computing to produce approximate solutions for combinatorial optimization problems. Its relative speed up in comparison to classical methods is an open question of research. [1]

In this thesis, I will examine if the Traveling Salesman Problem can be solved by the Quantum Approximate Optimization Algorithm (QAOA) defined in [4] on a simulated quantum computer without noise and if the QAOA is or will be able to outperform classical methods on this problem now or in the future. For that, I will first introduce commonly used classical methods to solve the problem with focus on their complexity with regards to the problem size and then examine the perfomance and the applicability of the QAOA on the Traveling Salesman Problem.

1.1 Traveling Salesman Problem

As mentioned, in the Traveling Salesman Problem (TSP), the objective is to find the shortest possible route that visits a set of cities exactly once and returns to the starting city. An instance of the problem can be modelled as a graph, which is a mathematical representation of a set of objects connected by links. In the context of the TSP, the objects are cities, and the links represent the paths between cities. Specifically, the TSP can be represented as an weighted graph. In this graph, the cities are the vertices (or nodes), the edges (or links) represent the paths between cities, and the weights assigned to the edges represent the distances between the cities.

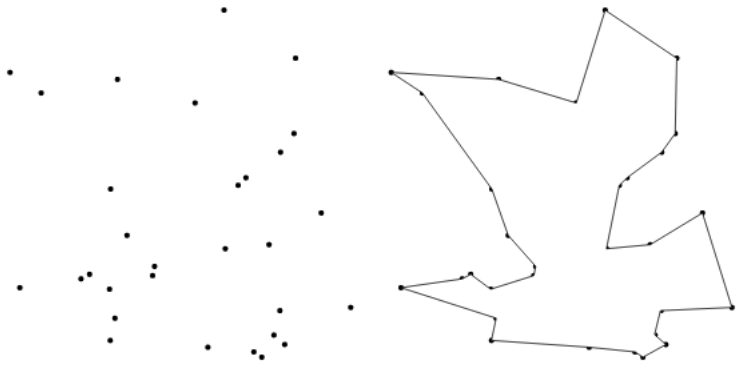


Figure 1: The Traveling Salesman Problem. Left: Distribution of cities. Right: The path that connects the cities in the shortest possible manner. [2]

Two main variants of the TSP exist: the symmetric TSP and the asymmetric TSP. In the symmetric TSP, the distance between any two cities is the same in both directions. This means that if we have two cities A and B, the distance from city A to city B is the same as the distance from city B to city A. This type of TSP forms an undirected graph.

In contrast, the asymmetric TSP allows for different distances between cities depending on the direction of travel. In this case, the distance from city A to city B may not be the same as the distance from city B to city A. This variant of the TSP forms a directed graph, where the edges have a direction, indicating wether the paths is ingoing or outgoing in regards to a city.

Both symmetric and asymmetric TSPs have their unique challenges and applications. For example, the asymmetric TSP can be used to model situations where travel in one direction may be affected by factors such as traffic, wind resistance, or elevation changes, making it different from travel in the opposite direction. In view of the graph representation, the asymmetric TSP forms a directed graph while the symmetric TSP forms an undirected graph. [5] [3]

1.2 Combinatorial optimization

Combinatorial optimization is a broad area of study within the field of optimization that focuses on finding the best solution from a finite set of possibilities. This involves selecting an object that optimizes a particular cost function or objective function. In the context of combinatorial optimization, the "objects" under consideration are usually arrangements, combinations, or sequences of elements from a given set. The primary challenge is to determine the best possible configuration that minimizes or maximizes a specified criterion.

Mathematically, a combinatorial optimization problem can be formalized as follows: given a set L of feasible solutions and a cost function $f : L \to \mathbb{R}$, the goal is to find an element $i \in L$ such that there is no other element $u \in L$ for which $f(u) < f(i)$. In other words, we aim to find a solution within the set L that minimizes (or maximizes) the value of the cost function f. The cost function f assigns a cost to each possible solution in L, and the goal of combinatorial optimization is to find the solution with the lowest (or highest) cost.

Combinatorial optimization problems often arise in practice, and they are characterized by their discrete nature, large search space, and complex constraints. Examples of such problems include, as mentioned, the Traveling Salesman Problem, the minimum spanning tree problem, and the knapsack problem. In the minimum spanning tree problem, the goal is to find the smallest possible connected subgraph of a given graph that includes all the vertices and has the smallest total edge weight. In the knapsack problem, the objective is to select a subset of items with given weights and values to maximize the total value without exceeding a specified weight limit.

Combinatorial optimization problems are often computationally challenging due to their large search space and the need to evaluate multiple solutions. They frequently fall into the category of NP-hard problems, which means that finding an optimal solution becomes exponentially difficult as the size of the problem grows.

2 Classical Methods to solve the Traveling Salesman Problem

2.1 Approximate Algorithms

For small instances of the TSP, it is possible to solve the problem exactly using deterministic algorithms that explore all possible routes and select the shortest one. However, as the number of cities grows, the search space expands exponentially, rendering exact solutions computationally infeasible even for moderate-sized instances. Consequently, researchers have shifted their focus towards the development and application of approximate algorithms, which provide reasonably accurate solutions within acceptable time frames.

Approximate algorithms offer a valuable compromise between solution accuracy and computational efficiency. These algorithms, also known as heuristic algorithms, attempt to find good solutions without exploring the entire search space. Instead, they use various strategies to intelligently guide the search towards promising regions of the solution space.

In practice, approximate algorithms have proven to be highly effective for solving large instances of the TSP, particularly when the exact solution is not required.

In the following sections, I will provide an overview of some of the most known and used approximate algorithms to solve the TSP. These approaches represent the state of the art in classical methods for tackling this computationally challenging problem.

2.1.1 Nearest Neighbour Algorithm

The Nearest Neighbour Algorithm is an intuitive approach for approximating the solution to the TSP. As a greedy algorithm, the nearest neighbour method operates by making locally optimal choices at each stage of the solution process, with the aim to arrive at a globally good-enough solution. In this context, the "greedy" nature of the algorithm refers to its strategy of selecting the best available option at each step, without considering the consequences of these choices.

In the Nearest Neighbour Algorithm, the traveling salesman begins at a starting city and proceeds to the nearest unvisited city. He does so until all cities have been visited once. The route concludes with a return to the original starting city.

The Nearest Neighbour Algorithm is favored for its computational efficiency and ease of implementation. Its runtime is polynomial with respect to the problem size, making it suitable for instances of the TSP where a quick approximation is needed. The simplicity of the algorithm is also an advantage, as it can be quickly implemented with minimal computational resources.

However, the algorithm has limitations. Since it operates based on local, short-term decisions, it can often result in suboptimal tours. In the worst-case scenarios, the resulting tours may be significantly longer than the optimal solutions. Despite this, the Nearest Neighbour Algorithm can produce good approximations in certain instances.

The quality of the approximation can vary based on factors such as the distribution of cities and the distances between them.

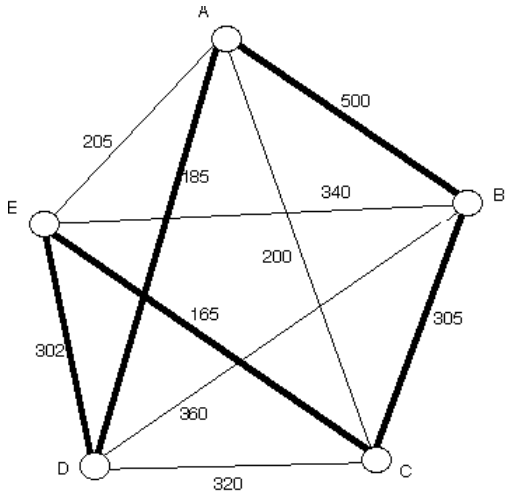


Figure 2: Nearest Neighbour Algorithm. Starting with city A, the next nearest neighbour is city D with a distance of S. The nearest unvisited neighbour then is city E with a distance of 302. This continues untill all cities are visted. Then path closes with traveling from the last visted city to the first. Here, from city B to city A. [25]

To evaluate the quality of the resulting tour, one approach is to compare the lengths of the initial and final stages of the tour. If the lengths of the last few stages are significantly greater than those of the first few stages, it may indicate that a better tour is possible. This is because the longer final stages may be a consequence of the greedy approach, which can lead to suboptimal choices in the latter part of the tour. By assessing the consistency of stage lengths through the tour, it is possible to gain insight into the quality of the approximation.

In conclusion, the Nearest Neighbour Algorithm offers a straightforward and computationally efficient approach to approximating the TSP. While it may not always yield the best solutions, it provides a useful starting point for further exploration and refinement of TSP solutions. [6] [3]

2.1.2 The Algorithm of Christofides and Serdyukov

The Algorithm of Christofides and Serdyukov, commonly referred to as the Christofides heuristic, is one of the best-known approximation algorithms for the Traveling Salesman Problem (TSP). The Christofides heuristic is notable for its theoretical guarantee of providing a solution that is at most $\frac{3}{2}$ times the length of the optimal tour.

The Christofides heuristic consists of several steps, each consisting of well known algorithms from graph theory and combinatorial optimization. The steps of the algorithm are as follows:

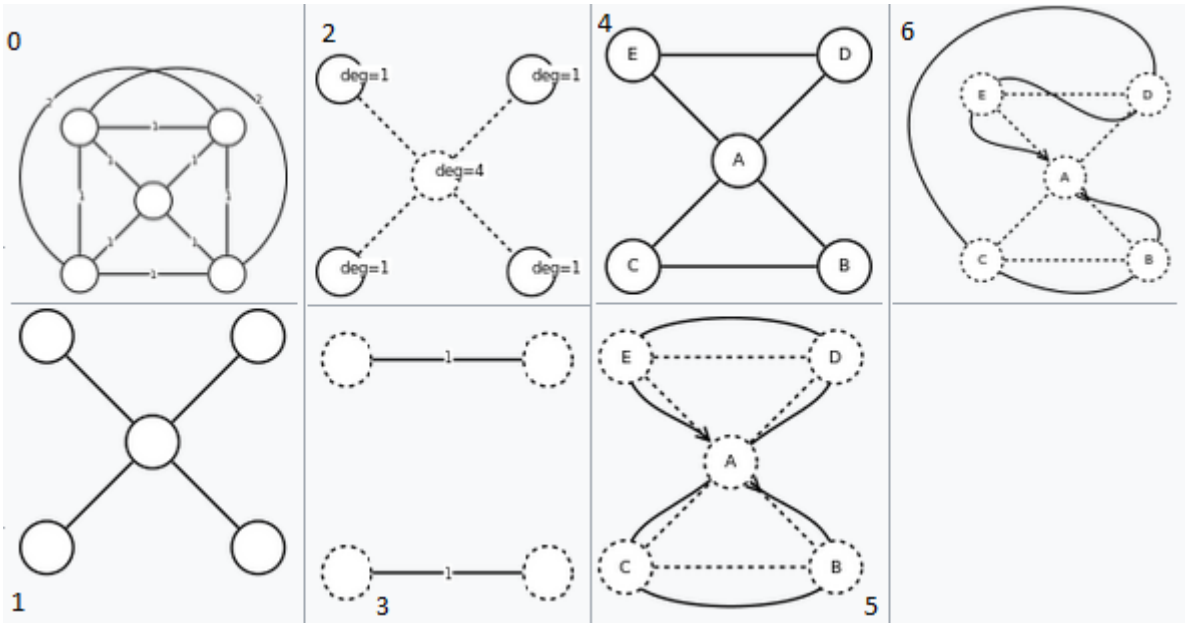


Figure 3: The Algorithm of Christofides and Serdyukov and its steps. [23]

1. Begin by finding a minimum spanning tree (MST) T for the underlying graph $G = (V, E)$ of the TSP instance. The MST connects all the vertices of the graph with a minimal total edge weight. Various algorithms, such as Kruskal's algorithm or Prim's algorithm, can be used for this. These algorithms are polynomial in time, making them computationally efficient for constructing the MST. [9]

2. Identify all vertices V' with an odd degree in the MST T. In graph theory, the degree of a vertex is the number of edges connected to it. Vertices with an odd degree are essential for the next steps of the algorithm.

3. Find a minimum weight perfect matching M on the subgraph induced by the vertices V'. A perfect matching is a set of edges in which each vertex is connected to exactly one edge from the set, and no two edges share a common vertex. The minimum weight perfect matching minimizes the sum of edge weights in the matching. The Edmonds' blossom algorithm, which runs in $\mathcal{O}(V^2E)$ time, is a standard approach for finding such a matching. [8]

4. Add the edges of the matching M to the MST T. The resulting graph $M \cup T$ now has only vertices with an even degree, a necessary condition for the next steps.

5. Find an Eulerian circuit for the graph $M \cup T$. An Eulerian circuit is a closed trail that visits every edge of the graph exactly once, but multiple visits for each vertex is allowed. Hierholzer's algorithm, a polynomial-time algorithm, can be used to find the Eulerian circuit efficiently.

6. Convert the Eulerian circuit into a Hamiltonian circuit, which is a closed path that visits each vertex of the graph exactly once. This is achieved by choosing an arbitrary starting vertex and following the Eulerian circuit, skipping any vertices that have already been visited. The resulting Hamiltonian circuit represents the tour for the TSP.

All steps of the Christofides heuristic are based on polynomial-time algorithms, and thus, the overall algorithm runs in polynomial time. [7]

2.1.3 K-Opt Heuristic and V-Opt Heuristic

The K-Opt algorithm is a refinement technique commonly used for enhancing existing solutions to the Traveling Salesman Problem (TSP). This heuristic, a variant of the Lin-Kernighan heuristic, iteratively improves suboptimal solutions by modifying the structure of the current solution. The most basic form of this method is the 2-Opt algorithm, where two edges are removed from the current solution, and the remaining vertices are reconnected in a different way. If the resulting tour is shorter than the original, it is kept; otherwise, the change is undone.

The K-Opt algorithm generalizes this approach, allowing for the removal and rearrangement of any number of edges k in each step. So, it removes k disjoint edges from the current solution and reassembles the remaining graph into a complete tour. Due to the combinatorial nature of the problem, there are $2k$ possible ways to reconnect the graph, which requires solving a $2k$-city TSP at each step. This subproblem is usually addressed using a brute-force algorithm.

This heuristic requires an initial tour, which is typically generated using a simple greedy algorithm such as the nearest neighbor method. However, it can also be used to enhance already good solutions, derived by for example the Christofides heuristic. Although the K-Opt algorithm can potentially produce near-optimal solutions in polynomial time for some TSP instances, there is no guarantee that it will do so.

In practice, the value of k is often allowed to vary during the optimization process. In this case, the method is called V-Opt heuristic. By varying k, the algorithm can explore a wider range of potential solutions, potentially escaping local optima that might trap a fixed-k algorithm. This variation of the K-Opt heuristic provides additional flexibility and adaptability in addressing the TSP.

One of the key advantages of the K-Opt and V-Opt heuristics is their ability to iteratively improve existing solutions, potentially leading to better results. This capability makes them useful for improving solutions obtained from other algorithms or methods.

Despite their benefits, it is important to note that the K-Opt and V-Opt heuristics, like all heuristic methods, have limitations. The quality of the final solution depends on the initial tour provided to the algorithm, as well as the specific instances of the TSP being addressed. But, while these heuristics can provide good solutions in many cases, there is no theoretical guarantee of optimality. Nonetheless, they are important techniques for tackling the TSP, especially in cases where the problem size or complexity makes exact methods unfeasible.[10]

2.1.4 Ant Colony Optimization

Ant Colony Optimization (ACO) is an optimization technique that is inspired by the behavior of real ant colonies in finding the shortest path between food sources and their nest. This algorithm models the process by which ants use pheromone trails to communicate and cooperate with one another to find efficient paths. This concept has been adapted to solve combinatorial optimization problems, including the TSP.

In ACO, a large population of virtual ants is generated, and each ant is allowed to travel through the cities of the TSP, forming complete tours. The decision-making of the ants is influenced by two factors: the concentration of pheromones on the edges connecting cities and the length of these edges. Ants stochastically tend to choose paths with higher pheromone concentrations and shorter lengths, similar to the behavior observed in real ants.

To explore different paths, each virtual ant has a memory to keep track of visited cities, ensuring that no city is visited more than once in a single tour. After an ant completes a tour, it spreads pheromones on the edges of its path. The amount of pheromone deposited is inverse proportional to the total length of the path, with shorter paths receiving more pheromones. This process helps guide other ants towards more efficient paths.

As more ants travel through the graph and deposit pheromones, the pheromone concentrations on the edges evolve, eventually stabilizing and reaching an equilibrium. The path with the highest pheromone concentration is then considered as a potential solution to the TSP.

The complexity and convergence of the ACO algorithm in solving the TSP are topics of ongoing research. The algorithm's decision-making process is probabilistic, which introduces some variability in the solutions obtained. The feedback mechanism, through pheromone deposition and evaporation, allows the algorithm to explore different solutions and avoid getting stuck in local optima. The performance of the algorithm can be adjusted by modifying parameters such as the number of ants, the rate of pheromone evaporation, and the relative importance of pheromone concentration and edge length in the ants decisions.

In summary, the Ant Colony Optimization algorithm provides an approach to solving the Traveling Salesman Problem by simulating the natural behavior of ants. The algorithm uses the emergent behavior and adaptability of ants to explore different solutions and reinforce efficient paths. The exploration of various paths and the updating of pheromone trails contribute to the identification of potential solutions to the TSP. Research continues in this area to further understand and optimize the performance and convergence of the algorithm. [11] [12]

Having an overview of the best-known approximate algorithms to solve the TSP, I will now introduce the best-known exact alogrithms.

2.2 Exact Algorithms

To solve the TSP exactly using classical algorithms, one might instinctively consider an exhaustive search of all possible paths, storing the best one as the solution. However, this approach becomes impractical even for a comparably small number of cities due to the rapid growth of the number of possible paths. Specifically, the number of paths, p, that need to be checked is given by the formula

$$p = \frac{(n-1)!}{2} \tag{1}$$

where n represents the number of cities in the TSP. This factorial growth of the number of paths with respect to the number of cities makes an exhaustive search approach already infeasible for problem instances with less then 100 cities. Even with modern computing power, examining all possible paths quickly becomes unfeasible as the number of cities increases.

Fortunately, there are alternative deterministic algorithms that can solve the TSP exactly more efficiently. One such class of algorithms includes the Branch-and-Cut and Branch-and-Bound methods. They use strategies to explore the solution space selectively, avoiding the need to examine every single path. By pruning parts of the search space that are guaranteed not to contain optimal solutions, these methods significantly reduce the number of paths that need to be considered, making it feasible to solve TSP instances of relevant sizes.

A crucial mathematical tool underlying these methods is integer linear programming (ILP), which is a general framework for solving optimization problems with integer variables and linear constraints. By formulating the TSP as an ILP problem, these algorithms can leverage optimization techniques and solvers to find the exact solution efficiently. To provide a deeper understanding of how the branch-and-bound and branch-and-cut algorithms work, I will give a brief introduction to integer linear programming and its application to the Traveling Salesman Problem in the next section.

2.3 Integer linear programming

Integer Linear Programming (ILP) is an optimization technique that uses the concept of Linear Programming (LP) by introducing integrality constraints on the decision variables. Before addressing into the specifics of ILP, I will first explain the foundational concept of Linear Programming.

Linear Programming (LP) is an optimization technique that aims to find the maximum value of a linear objective function subject to a set of linear constraints. An LP problem can be mathematically described as:

$$\max_{x_i} f(x_i) = \sum_i c_i x_i \tag{2}$$

where the function $f(x_i)$ is the objective function to be optimized, and c_i are the coefficients of the decision variables x_i. This maximization problem is subjected to

constraints of the form:

$$\begin{aligned} \forall i : \sum_j a_{ij} x_j \leq b_i \\ \forall i : x_i \geq 0, \end{aligned} \tag{3}$$

where a_{ij} and b_{ij} are coefficients and constants, from the constraint equations. All these parameters, c_{ij}, a_{ij} and b_{ij}, belong to the set of real numbers $\mathbb{R}$.

Solving an LP problem involves finding values for the decision variables x_i that satisfy the constraints and optimize the objective function. LP problems can be solved efficiently using algorithms such as the Simplex algorithm, which is known to be polynomial in time. The LP framework can also be used for minimization problems by considering the negation of the objective function:

$$\min_{x_i} f(x) = \max_{x_i} -f(x). \tag{4}$$

Integer Linear Programming (ILP) on the other hand, is a specialized form of LP where the decision variables are constrained to integer values. An ILP problem can be described as an LP problem with an added integrality constraint, $x_i \in \{0, 1\}$. ILP problems cannot be solved directly using the Simplex algorithm. A common approach is to perform a linear programming relaxation (LP-Relaxation) followed by the application of the cutting plane method.

The LP-Relaxation of an ILP problem means to relax the integrality constraint by allowing the decision variables to take any value within the range $[0, 1]$:

$$0 \leq x_i \leq 1. \tag{5}$$

This relaxation transforms the ILP problem into an LP problem, which can be solved using standard LP algorithms. To ensure integrality again, one now uses the cutting plane method.

The cutting plane method is an iterative technique to find feasible integer solutions by solving the relaxed version of an ILP problem and adding cutting planes if needed. A cutting plane is a constraint of the form:

$$\sum_i a_i x_i \leq b \tag{6}$$

with a_i,$b \in \mathbb{R}$.

The cutting plane method works as follows: First, solve the relaxed ILP problem and check if the obtained optimum is a feasible integer solution. If the solution is feasible, the ILP problem is solved. If not, add a cutting plane to the relaxed ILP problem that excludes the prior non-integer optimum from the set of feasible solutions. This process is repeated until a feasible integer solution is found. [26]

In order to apply ILP to the TSP, one must first derive an ILP representation of the problem. In this context, two formulations are best-known: the Miller-Tucker-Zemlin (MTZ) formulation and the Dantzig-Fulkerson-Johnson (DFJ) formulation.

To establish the ILP representation of the TSP, we consider a set of cities indexed by the labels $1, \ldots, n$, and denote the distance between the cities i and j as c_{ij}. The decision variables for the problem are defined through the binary variable

$$x_{ij} = \begin{cases} 1 & \text{if the path includes the edge from city } i \text{ to } j \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

The objective is to minimize the cost function defined by the total distance of the selected tour:

$$c(x) = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} c_{ij} x_{ij} \tag{8}$$

Additional constraints are needed to ensure that each city has exactly one incoming and one outgoing edge:

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = 1 \quad \forall j \tag{9}$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij} = 1 \quad \forall i \tag{10}$$

And one has to add one constraint, that ensures that the resulting tour is exactly one tour and not a set of subtours. For that, one needs to define a set of dummy variables u_i for $i = 2, ..., n$ that represents the order of the cities. u_i thus represents the order of the i-th city. With that, the constraint can be formulated as

$$u_j \geq u_i + 1 \quad \text{if} \quad x_{ij} = 1. \tag{11}$$

or as

$$u_j + (n-2) \geq u_i + (n-1)x_{ij} \tag{12}$$

$$\Leftrightarrow \quad u_i - u_j + (n-1)x_{ij} \leq n - 2, \tag{13}$$

for all $2 \leq i \neq j \leq n$. These $(n-1)(n-2)$ equations are easier to implement in most cases and algorithmically equivalent. The whole linear programming problem of the TSP in the MTZ formulation is thus

$$\min \sum_{i=1}^{n} \sum_{\substack{j=1 \\ i \neq j}} c_{ij} x_{ij} : \tag{14}$$

$$x_{ij} \in \{0,1\} \qquad i,j = 1,...,n; \tag{15}$$

$$u_i \in \mathbb{Z} \qquad i = 2,...,n; \tag{16}$$

$$\sum_{\substack{i=1 \\ i \neq j}} x_{ij} = 1 \qquad j = 1,...,n; \tag{17}$$

$$\sum_{\substack{j=1 \\ j \neq i}} x_{ij} = 1 \qquad i = 1,...,n; \tag{18}$$

$$u_i - u_j + (n-1)x_{ij} \leq n-2 \quad 2 \leq i \neq j \leq n; \tag{19}$$

$$1 \leq u_i \leq n-1 \qquad 2 \leq i \leq n; \tag{20}$$

The other common formulation of the problem, the Dantzig Fulkerson Johnson formulation (DFJ), only differs to the MTZ in the constraint that ensures that a set of closed subtours is not a feasible solution. In the DFJ formulation, the constraint is given by:

$$\sum_{i \in Q} \sum_{\substack{j \neq i \\ j \in Q}} x_{ij} \leq |Q| - 1 \tag{21}$$

with Q being any feasible subset of $\{1,...,n\}$ with more than two elements. This constraint technically encapsulates an exponential number of constraints. So, a common technique is to add single cuts derived from this constraint if a closed subtour occures.

It is important to note that both formulations can potentially require an exponential number of constraints. In the MTZ formulation, the initial constraints are polynomial, but additional constraints (cuts) may be necessary during the solution process with the cutting plane method to obtain an integer solution with only one tour.

In the DFJ formulation, the subtour elimination constraint itself can result in an exponential number of constraints in addition to the cuts that may be needed for an integer solution. This exponential growth comes from the need to potentially consider all possible subsets of cities that could form subtours.

Consequently, in both formulations, the cutting plane method can potentially introduce an exponential number of constraints. [27]

2.4 Branch and Bound

The Branch and Bound algorithm is an efficient approach to minimize an objective function $f(x)$ over a set of feasible solutions S, offering optimization advantages over the cutting plane method. Here, S represents the feasible region or search space. The algorithm works by recursively dividing the search space S into smaller regions, determining the optimal value of f in these regions, and maintaining a record of the best solution

found so far. In a brute-force approach, the search space would be divided until the subregions are indivisible. The Branch and Bound algorithm improves this by calculating lower bounds for f in the subdivided regions of the search space. The algorithm can then prune subproblems where the lower bound exceeds the current upper bound, which is the current best solution. This rules out the need to further explore of those regions. This pruning process can significantly reduce computational effort compared to a brute-force approach, as a considerable portion of the search space may be excluded, eliminating the need to check many potential solutions. Even though the algorithm's runtime may still be of exponential order in $dim(S)$, and thus to the number of cities in the TSP, the Branch and Bound approach may result in substantial time savings.

The size of the solution subspace that can be pruned and where f does not need to be calculated, is dependent on the quality of the heuristic used to estimate the lower bounds.

The Branch and Bound algorithm can as well be used to maximize functions. This is done by using heuristics calculate upper bounds for subdivided regions and comparing it to the current lower bound of the optimal solution, the best solution so far. In the following, I will only consider the minimization algorithm, as only this is needed for the TSP.

To formulate an algorithm one must also introduce the concept of instances I of the search space S. An instance represents a subset of the search space. One can branch the instance, which results in multiple mutually disjoint subsets of the instance

$$I_i = \text{branch}(I), \tag{22}$$

one can calculate a lower and upperbound for an instance with a given heuristic,

$$\begin{aligned} l &= \text{lower\_bound}(I) \\ u &= \text{upper\_bound}(I), \end{aligned} \tag{23}$$

Additionally, it is possible to determine whether an instance represents only a single solution candidate:

$$\text{solution}(I) = \begin{cases} x_{ij} & \text{if I only represents one specific solution } x_{ij} \\ \text{false} & \text{otherwise.} \end{cases} \tag{24}$$

In the context of the Traveling Salesman Problem using the MTZ formulation, branching involves dividing an instance into two, setting a specific x_{ij} to 1 in one instance and 0 in the other. For the lower bound calculation, a heuristic can be used, such as the minimum spanning tree (MST) of the graph representing the problem, derived efficiently using algorithms like Kruskal's.

While this is a useful heuristic, it may not always provide the tightest lower bound, and there are other heuristics available. As the performance of the Branch and Bound algorithm highly depends on the quality of the lower bound heuristic, a careful selection is crucial.

To have a good initial upper bound of the solution one can, for example, use the nearest neighbour algorithm.

With that, a pseudocode for the Branch and Bound algorithm for the Traveling Salesman Problem can be formulated as such.

Algorithm 1: Branch and Bound for the Traveling Salesman Problem

Input:

$\mathbf{S}$ as the search space
$\mathbf{c_{ij}}$ as the distance between city i and city j
$\mathbf{c(x)} = \sum_{\mathbf{i=1}}^{\mathbf{n}} \sum_{\substack{j=1 \\ j \neq i}}^{\mathbf{n}} \mathbf{c_{ij} x_{ij}}$ as the path length to be minimized

Output:

$\mathbf{x}$ with $\min_{\mathbf{y}} \mathbf{c(y)} = \mathbf{c(x)}$

```
1  x ← solution of upper_bound(S)
2  B ← upper_bound(S)
3  initialize a queue Q to hold instances of the search space
4  store S in the queue Q
5  while Q not empty do
6      take an instance I from Q
7      if solution(I) then
8          if c(I) < B then
9              B ← c(I)                    // I is the best solution so far
10             x = I
11     else
12         I_i ← branch(I)
13         for i=1,2 do
14             if B > lower_bound(I_i) then
15                 Store I_i in Q
16 return x
```

In this algorithm, a queue Q is used to store instances of the search space, which are processed one after another. The algorithm explores the search space by branching instances, calculating bounds, and pruning regions as appropriate.

Several types of queues can be used for Q. One option is to use a priority queue that sorts instances based on their lower bounds. Another option is a LIFO (last in, first out) queue. When a priority queue is used, the algorithm is called a best-first

branch and bound. This approach is recommended when a good heuristic is available to produce lower bounds, as it tends to find good solutions early and results in more efficient pruning. If a suitable heuristic is not available for the initial solution and upper bound, it is advisable to use a LIFO queue, as this approach produces full solutions and upper bounds more quickly. This variant is called a depth-first branch and bound. [14]

2.5 Branch and Cut

The best known algorithm today to solve the Traveling Salesman Problem and many other combinatorial optimization problems that can be formulated as an interger linear problem (ILP) is the branch and cut algorithm.

The Branch and Cut algorithm shares many similarities with the Branch and Bound algorithm, particularly in its approach to generating lower bounds for instances and branching its instances until a solution is found or the lower bound exceeds the previously found minimum. However, the Branch and Cut algorithm incorporates additional features that make it a more powerful approach for solving ILPs. Specifically, the algorithm uses the linear programming (LP) relaxation of its instances and solves them with the simplex algorithm, but without using cutting planes.

The procedure of the Branch and Cut algorithm follows a systematic process. First, the algorithm checks if the solution obtained from the LP relaxation has a cost greater than the current minimum. If it does, the branch will be discarded and not considered further. If the cost is less than the current minimum, the algorithm then checks if the solution is feasible and integral. If it is, the instance is completed, and a new minimum is found. If the solution is not integral, the algorithm searches for violated cutting planes and adds them to the LP, minimizing it once more. This process is repeated until a feasible, integral solution is obtained, the cost exeeeds to current mimimum or until all cutting planes are satisfied. If the latter occurs and the result is still not a feasible, integral solution, the algorithm branches the result and applies the same steps to the new branches.

For the TSP it is mandatory to use the MTZ, not the DFJ for the Branch and Cut algorithm as the MTZ-Formulation itself does not have a number of cutting planes, that grows exponentially with the number of cities.

Just like the Branch and Bound algorithm, the Branch and Cut algorithm prunes branches by comparing their lower bounds to the current optimal solution. The key difference is that the Branch and Cut algorithm uses the non-integral solutions of the LP relaxation as lower bounds. This is a valid approach because the cost can only increase with the addition of more constraints, which are necessary to obtain an integral solution.

The Branch and Cut algorithm offers a wide range of methods for branching and selecting variables to branch on. While the most common approaches involve branching on a single variable, there are variations in how this can be achieved. One possibility is to set one decision variable in one branch to 0 and in the other to 1. Alternatively, two new constraints can be added:

Algorithm 2: Branch and Cut for the Traveling Salesman Problem

Input:

```
L, c                          // The TSP problem as ISP in MTZ formulation
```

Output:

```
x                                                  // The solution to the TSP
```

```
1  initialize a queue Q to hold instances of integer linear programs or linear
     programs
2  store L in the queue Q without cutting planes
3  v* ← ∞, x* ← null
4  while Q not empty do
5      take an ILP I from Q
6      I ← LP-Relaxation(I)
7      x ← simplex(I)
8      v ← c(x)
9      if v < v* then
10         if x is a feasible solution then
11             v* ← v
12             x* ← x
13         else
14             if x violates a cutting plane then
15                 add said cutting plane to I
16                 jump to 7.
17             else
18                 I_i ← branch(I)
19                 for i=1,2 do
20                     store I_i in Q
21 return x
```

$$\begin{aligned} x_{ij} &\leq x_{ij}^* \\ x_{ij} &\geq x_{ij}^* \end{aligned} \tag{25}$$

where x_{ij}^* is the value of the decision variable x_{ij} on which one wants to branch. A common approach for choosing which variable to branch on is the so-called most

infeasible branching, where the variable closest to 0.5 is selected. A pseudocode version of the Branch and Cut algorithm can be found in Algorithm [2] and more detailed information about the Branch and Bound or Branch and Cut algorithms can be found in the references [13] or [14].

2.6 Complexity and Results

In aboves exploration of classical methods for solving the Traveling Salesman Problem (TSP), it is worth noting that the best known exact algorithm for this task, the Branch and Cut algorithm, still exhibits exponential complexity. This is shown in figure [4], where a benchmark test was conducted using random instances of the TSP. It is important to note that the y-axis of this figure is scaled logarithmically. Additionally, these tests were performed without any prior calculations to approximate and prune parts of the solution space.

The results shown in the figure indicate a wide distribution of runtimes. For more challenging instances of the TSP, specifically those with up to 26 cities, the algorithm's runtime was found to vary significantly. Some instances took more than a thousand seconds to be solved, while others were completed in less than one second. The disparity in these runtimes highlights the variable nature of the algorithm's performance.

It is essential to mention that one can observe a significant improvement when incorporating heuristics into the TSP-solving process. By using an appropriate heuristic to determine an upper bound for the path length prior to using the Branch and Bound or Branch and Cut algorithm, the runtime can be significantly reduced.

A prime example of the impact of heuristics on TSP solving is demonstrated by the Concorde TSP solver, as referenced in [18] and [19]. This solver employs a variety of heuristic strategies and has shown the ability to solve TSP instances of up to 500 cities in less than a minute on an Apple M1 processor. The Concorde TSP solver was also used to set a world record in 2006 by solving the largest TSP instance ever solved, known as the pla85900, which has 85,900 vertices. More details about this can be found in [20].

Although these are great results, it is essential to acknowledge that classical methods have limitations when it comes to solving TSP instances several magnitudes larger, particularly if one needs an exact solution. In light of this, it is worth to consider the potential of emerging technologies such as the Quantum Approximate Optimization Algorithm (QAOA). As a promising candidate for tackling other combinatorial optimization problems, the QAOA may be good candidate for the TSP as well.

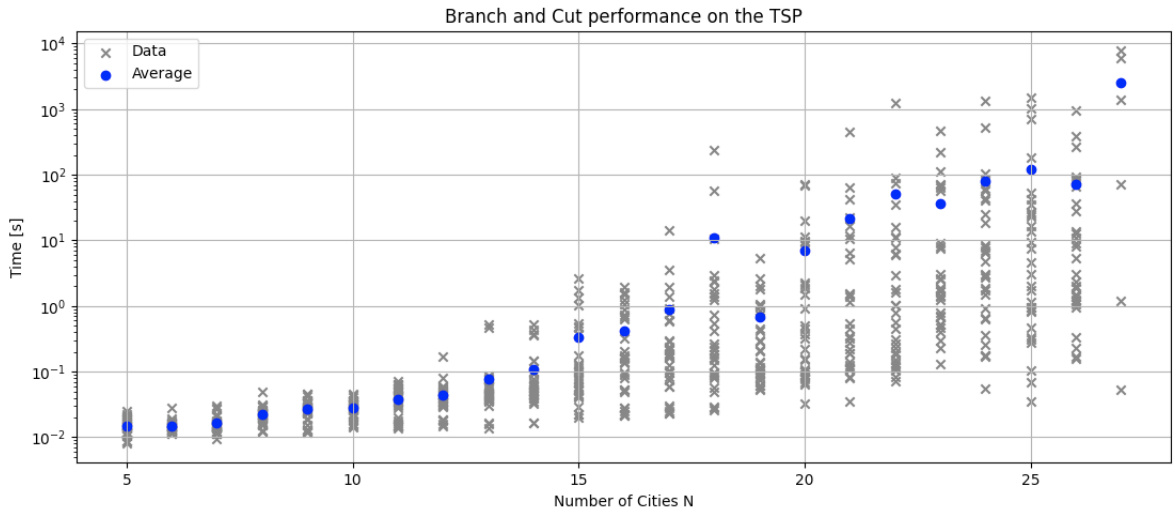


Figure 4: Performance measurements of the Branch and Cut algorithm on random instances of the TSP without a prior calculation of an approximated solution. Measured on an Apple M1 processor with the Python CPLEX API docplex.

3 Quantum Approximate Optimization Algorithm

First introduced by Edward Farhi and Jeffrey Goldstone, as cited in [6], the Quantum Approximate Optimization Algorithm (QAOA) serves as a method for approximately solving combinatorial problems. A distinctive feature of QAOA is that both the depth of its quantum circuit and the precision of its approximations are influenced by an integer parameter p, where $p \geq 1$.

The circuit depth scales, at most linearly dependent on the product of p and the number of constraints in the problem.

The algorithm uses a quantum circuit composed of unitary gates. The locality of these unitary gates is constrained by the locality of the objective function, which encapsulates both the problem and its optimal solution.

This objective function is expressed as $C(z)$, where z is a string of n bits, denoted as $z = z_1 z_2 ... z_n$. Typically, this function can be split into m clauses, encapsulated in the equation

$$C(z) = \sum_{\alpha=0}^{m} C_\alpha(z). \tag{26}$$

Each clause C_α acts as a constraint that focuses on a particular subset of bits, with $C_\alpha = 1$ if unsatisfied and $C_\alpha = 1$ if satisfied.

On a quantum computer, this bit string z is converted into computational basis vectors $|z\rangle$ within a Hilbert space of 2^N dimensions. The objective function itself takes on the form of a unitary operation, $U(C, \gamma)$, where γ is an angle that lies within the range $[0, 2\pi]$.

Before going into the details of the QAOA's functioning, it is necessary to introduce two more operators, B and $U(B, \beta)$. The operator B consists of a summation of Pauli-X matrices σ_j^x, and $U(B, \beta)$, with β in the interval $[0, \pi]$.

$$B = \sum_{j=1}^{n} \sigma_j^x, \tag{27}$$

and

$$U(B, \beta) = e^{-i\beta B} = \prod_{j=1}^{n} e^{-i\beta\sigma_j^x} \tag{28}$$

It is often called the mixer Hamiltonian in the QAOA.

In the algorithm, the system starts in a uniform superposition in the computational basis.

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_{z} |z\rangle \tag{29}$$

Then, for any $p \geq 1$, the unitary operations are being applied alternatingly with different angles. This defines the angle dependent quantum state:

$$|\gamma, \beta\rangle = U(B, \beta_p)U(C, \gamma_p)...U(B, \beta_1)U(C, \gamma_1) |s\rangle \tag{30}$$

where γ, β are two p-dimensional angle vectors. It is notable that, to produce this state, a quantum circuit with a depth of at most $mp + p$ is necessary. Finding the optimum of the cost function can then be redefined as finding the maximum of the cost expectation value of $|\gamma, \beta\rangle$ in dependence of γ, β.

$$M_p = \max_{\gamma,\beta} F_p(\gamma, \beta) = \max_{\gamma,\beta} \langle\gamma, \beta| C |\gamma, \beta\rangle \tag{31}$$

Further, it can be shown that

$$M_p \geq M_{p-1} \tag{32}$$

and

$$\lim_{p\to\infty} M_p = \max_{z} C(z). \tag{33}$$

So, to find the bit string z that maximizes $C(z)$ approximately, one picks a p and search for the maximum of $F_p(\gamma, \beta)$ in dependence of $(\gamma, \beta) \in [0, 2\pi] \times [0, \pi]$ with a classical optimizer. Each time, the value of $F_p(\gamma, \beta)$ needs to be calculated for the classical optimizer, a quantum computer is invoked to produce the state $|\gamma, \beta\rangle$ and to measure

it in the computational basis to get z. The production of the state and measurement must be repeated often enough with the same angles to obtain a z, such that $C(z)$ is near or greater than $F_p(\gamma, \beta)$. The obtained z can then be used to calculate the value of $C(z)$, which thus approximates $F_p(\gamma, \beta)$. The classical optimization of $F_p(\gamma, \beta)$ can be done efficiently since the partial derivatives of $F_p(\gamma, \beta)$ are bounded. In the end, one can increase p to increase the quality of the approximation of the maximum of $C(z)$.

3.1 The Traveling Salesman Problem as a quadratic unconstrainted binary optimization problem

In this section, I will introduce the simplest technique for using the QAOA to solve the TSP. For that, a quadratic unconstrained binary optimization (QUBO) problem formulation of the Traveling Salesman Problem is needed. Subsequently, I will use the formulation as described in [18]. As in the classical attempts to solve the Traveling Salesman Problem, one again has to abstract the problem in terms of graphs. With the cities serving as nodes or vertices, paths acting as edges, and the distances being the weights assigned to these edges, the TSP is about the shortest Hamiltonian cycle that goes through each of the vertices.

The Hamiltonian cycle is then again described by N^2 variables $x_{i,p}$, where i represents the node and p its order in the cycle, such that $x_{i,p}$ is one if the resulting path goes through node or city i at time p and zero if not. Additionally, it is necessary to include two additional constraints that resemble the initial two constraints in the ILP formulation. Firstly, each node is limited to a single occurrence within the cycle. Secondly, at any given moment, there must be one node that is visited. The latter can mathematically be formulated as such

$$\sum_i x_{i,p} = 1 \quad \forall p. \tag{34}$$

This requires that all $x_{i,p}$ are zero for all i for a given p except for one. Which encodes the requirement, that only one city is visited at a time. And

$$\sum_p x_{i,p} = 1 \quad \forall i \tag{35}$$

to require that all $x_{i,p}$ are zero for all p for a given i except for one. This constraint encodes the requirement that each city is visited only once in the cycle. The distance, which needs to be minimized, can be represented as

$$C(x) = \sum_{i,j} w_{ij} \sum_p x_{i,p} x_{j,p+1} \tag{36}$$

with w_{ij} being the weight of the edge (i, j) and therewith the distance between city i and j. Reformulating the constraints as quadratic functions and adding a penalty weight A leads to an overall cost function for the problem

$$C(x) = \sum_{i,j} w_{ij} \sum_p x_{i,p} x_{j,p+1} + A \sum_p (1 - \sum_i x_{i,p})^2 + A \sum_i (1 - \sum_p x_{i,p})^2. \tag{37}$$

To ensure that the constraints are adhered to during the optimization, A must be chosen to be sufficiently large. A typical approach for this selection is to set $A > max(w_{i,j})$.

3.1.1 QAOA Circuit for the QUBO TSP

Now that a proper cost function is derived one has to encode it as an operator. This can be achieved by mapping

$$x_{i,p} \rightarrow \widehat{x_{i,p}} = \frac{1 - \widehat{Z_{i,p}}}{2} \tag{38}$$

with $\widehat{Z_{i,p}}$ being the pauli-z operator acting on the qubit boolean that denotes if the path goes through city i at time p

$$\widehat{Z_{i,p}} = \mathbb{I} \otimes ... \otimes \mathbb{I} \otimes \widehat{Z} \otimes \mathbb{I} \otimes ... \otimes \mathbb{I}. \tag{39}$$

Note that the cost function fullfills the requirements stated in [26] because it is separable in sums of smaller cost functions which act only on a subset of all qubits. With that, the distance part of the cost function becomes

$$\begin{aligned} C_{distance}(x) &= \sum_{i,j} w_{i,j} \sum_p x_{i,p} x_{j,p+1} \\ &= \sum_{i,j} \sum_p w_{i,j} x_{i,p} x_{j,p+1} \\ &= \sum_{i,j} \sum_p w_{i,j} \frac{1 - \widehat{Z_{i,p}}}{2} \frac{1 - \widehat{Z_{j,p+1}}}{2} \end{aligned} \tag{40}$$

and the objective function unitary for the distance part thus becomes

$$\begin{aligned} U_{distance}(C_{distance}, \gamma) &= e^{-i\gamma C} \\ &= e^{-i\gamma \sum_{i,j} \sum_p w_{i,j} \frac{1-\widehat{Z_{i,p}}}{2} \frac{1-\widehat{Z_{j,p+1}}}{2}} \\ &= \prod_{i,j} \prod_p e^{-i\gamma w_{i,j} \frac{1-\widehat{Z_{i,p}}}{2} \frac{1-\widehat{Z_{j,p+1}}}{2}} \\ &= \prod_{i,j} \prod_p \text{CPhase}(-\gamma w_{i,j})_{i,p,j,p+1} \end{aligned} \tag{41}$$

where $\text{CPhase}(x)_{i,p,j,t}$ is a controlled phase gate with parameter x controlled by bit (i,j) and acting on bit $(j, p+1)$. For an example with distance matrix

$$D_3 = \begin{bmatrix} 0 & 3 & 5 \\ 3 & 0 & 4 \\ 5 & 4 & 0 \end{bmatrix} \tag{42}$$

and γ set to $1/2$, this results in the circuit displayed in figure 5.

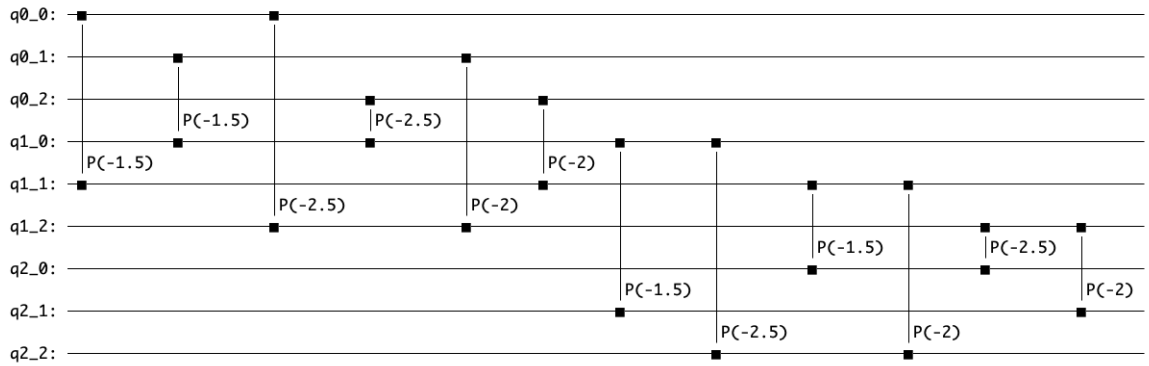


Figure 5: Path length cost unitary circuit for the QUBO TSP QAOA for 3 cities with distance matrix D_3 and $\gamma = 1/2$

The first part of the penalty part of the cost function, that penalizes solutions where one city occurs multiple times in a path, becomes

$$\begin{aligned} C_{penalty_1}(x) &= A\sum_p (1 - \sum_i x_{i,p})^2 \\ &= \sum_p A(1 - \sum_i x_{i,p})(1 - \sum_i x_{i,p}) \\ &= \sum_p A(1 - \sum_i x_{i,p} + \sum_{i,j} x_{i,p}x_{j,p}) \\ &= \sum_p A - \sum_p \sum_i Ax_{i,p} + \sum_p \sum_{i,j} Ax_{i,p}x_{j,p} \end{aligned} \tag{43}$$

and the resulting part of the objective function unitary becomes

$$\begin{aligned} U_{penalty_1}(C_{penalty_1}, \gamma) &= e^{-\sum_p \sum_i \gamma A x_{i,p} + \sum_p \sum_{i,j} \gamma A x_{i,p} x_{j,p}} \\ &= e^{-\sum_p \sum_i \gamma A x_{i,p}} e^{\sum_p \sum_{i,j} \gamma A x_{i,p} x_{j,p}} \\ &= \prod_{p,i} e^{-\gamma A x_{i,p}} \prod_{p,i,j} e^{\gamma A x_{i,p} x_{j,p}} \\ &= \prod_{p,i} \mathrm{Phase}(-\gamma A)_{i,p} \prod_{p,i,j} \mathrm{CPhase}(\gamma A)_{i,p,j,p}, \end{aligned} \tag{44}$$

where $\text{Phase}(x)_{i,p}$ is a phase gate with parameter x acting on qubit (i,p). Analogously, the second part of the penalty part of the cost function, that penalizes solutions where multiple cities occur on the path at the same point in time, becomes

$$C_{penalty_2}(x) = \sum_i A - \sum_i \sum_p Ax_{i,p} + \sum_i \sum_{p,t} Ax_{i,p}x_{i,t} \tag{45}$$

and its part of the objective function, thus becomes

$$U_{penalty_2}(C_{penalty_2}, \gamma) = \prod_{i,p} \text{Phase}(-\gamma A)_{i,p} \prod_{i,p,t} \text{CPhase}(\gamma A)_{i,p,i,t}. \tag{46}$$

For A set to 10 and $\gamma = 1/2$, both penalty parts of the cost function result in the circuit in figure 6.

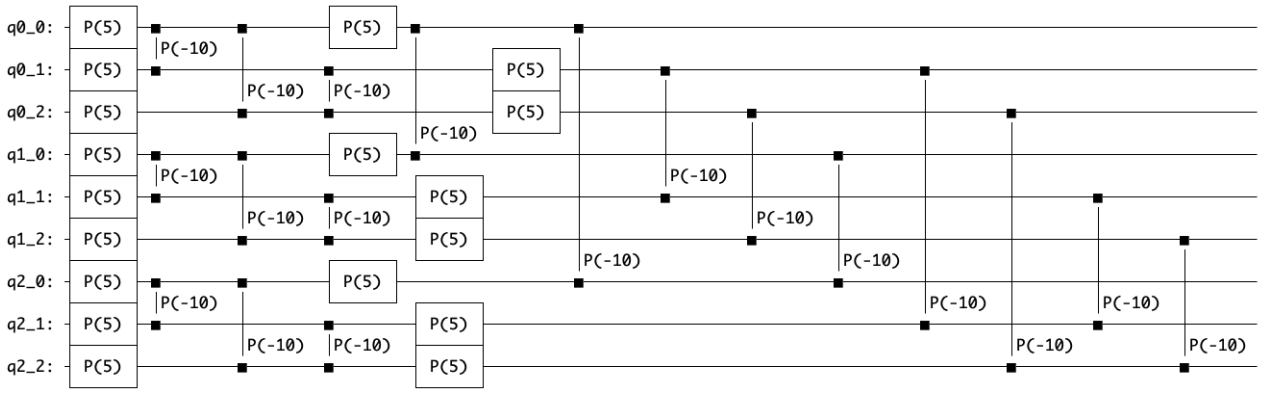


Figure 6: Panalty cost unitary circuit for the QUBO TSP QAOA for 3 cities with $A = 10$ and $\gamma = 1/2$

The other circuit, needed for the QAOA is the mixer unitary

$$U(B,\beta) = e^{-i\beta B} = \prod_{j=1}^{n} e^{-i\beta\sigma_j^x}, \tag{47}$$

which can be realized with single-qubit rotations about the X-axis - RXGates:

$$\text{RXGate}(\phi)_j = exp(-i\frac{\phi}{2}X_j) \Rightarrow U(B,\beta) = \prod_{j=1}^{n} \text{RXGate}(2\beta)_j. \tag{48}$$

To produce a superposition over all states in the beginning, one can initialize the n^2 qubits and apply a single-qubit Hadamard gate on each of them.

3.1.2 Results

The circuit described above can be programmed using various languages specifically designed for quantum computing. For this discussion, I'll focus on using the Qiskit module for Python, which was developed by IBM (see [18]). This module allows you to both simulate quantum circuits and the outcomes of measurements in those circuits, as well as run experiments remotely on actual quantum computers.

Unfortunately, there are only 5 qubit quantum computers available at the time of writing and for the TSP this would mean, an instance of only two cities could be solved, which is trivial. The simulation of a quantum circuit requires an exponential computational overhead with regards to the number of qubits. The system I am working on is able to simulate up to 23 qubits, which is enough for small non-trivial instances. Another benefit of using simulation over a real quantum device is the ability to toggle noise on and off. This feature is valuable for studying the algorithm. However, it is important to remember that we must account for noise when thinking about real-world applications.

To begin with the simplest problem, one should have a look at the results of a three city TSP instance with supressed noise with $p = 1$. To minimize the function in the angle-space I will use a fine grid search algorithm, because it is a reliable global optimizer and almost neglectable as an error source.

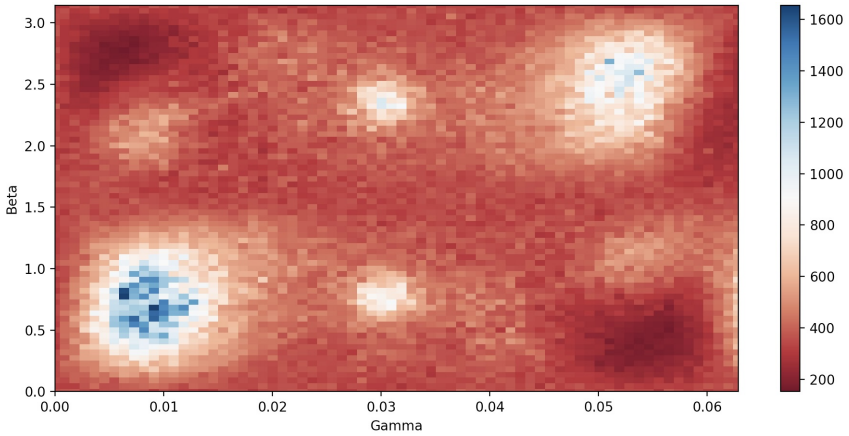


Figure 7: Average cost value of the result of a $p = 1$ QAOA circuit with aboves distance matrix D_3 with 100 shots per parameter pair and $A = 50$ and a minimum at $(\gamma, \beta) = ((0.00546), (2.64075))$

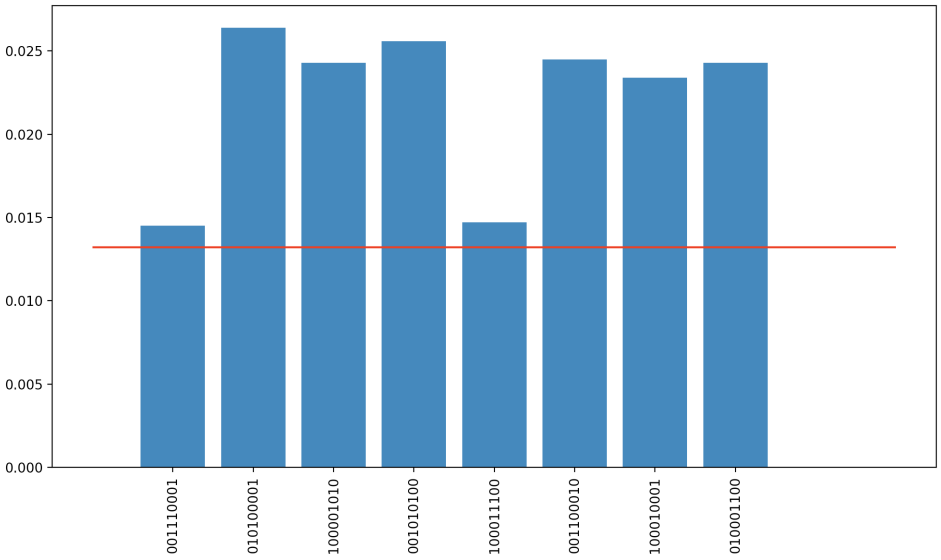


Figure 8: Histogram of the results of a $p = 1$ QAOA circuit with distance matrix D_3, $\gamma = 0.0054$ and $\beta = 2.731$ with 10000 shots per parameter pair and $A = 50$ filtered by $p_i > \overline{p}$. The six most often results are exactly the three feasible results.

In figure 7, we see the average cost values for the outcomes of a Quantum Approximate Optimization Algorithm (QAOA) applied to the Traveling Salesman Problem (TSP) over 100 shots. The curve appears to have four local minima when dealing with three cities ($n = 3$). figure 8 displays that when the cost function reaches its minimum based on the selected parameters, the outcomes across multiple experiments are relatively evenly distributed, with a few maxima that have less than a 3% probability in a sample size of $10,000$.

With a closer look one can see, that the six most probable outcomes are exactly the six feasible solutions to the $n = 3$ TSP in the encoding of the QUBO notation $100010001, 100001010, 001010100, 001100010, 010100001, 010001100$, which are equivalent to the orders $123, 132, 321, 312, 213, 231$.

To better understand the results from the QAOA experiments with set angles, further analysis is necessary. Initially, it is helpful to filter out bitstrings that do not satisfy the TSP constraints, as described in equations (34) and (35). Also, one can filter out circular equivalent tours and show only one representative tour with the summed-up probability of all identical tours. Likewise, it is good to account for directional equivalence, since this thesis does not focus on directed TSP, tours that are the same but in opposite directions should be considered identical.

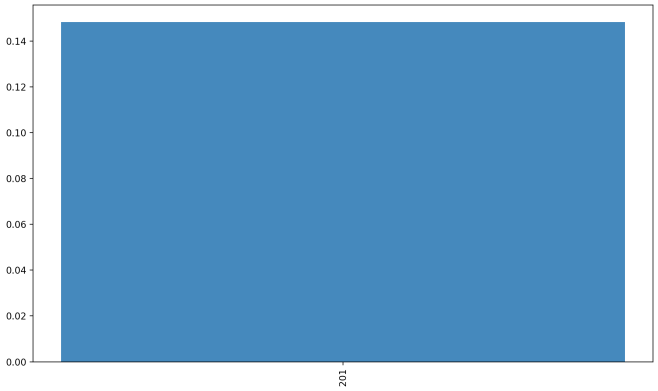


Figure 9: Filtered histogram of the results of a $p = 1$ QAOA circuit with distance matrix D_3, $\gamma = 0.0054$ and $\beta = 2.731$ with 10000 shots and $A = 50$.

The outcome of the experiment with $n = 3, p = 1, A = 50, (\gamma, \beta) = ((0.00546), (2.7318))$, 1000 Shots and distance matrix D_3 would then look like in figure [9].

Notably, all valid tours fall under the same class of undirected tours, and each tour has roughly a $p \approx 15\%$ chance of being measured. This is not surprising, as a TSP involving only three cities essentially has just one valid solution if one does not count tours in opposite directions as different. For more significant results, one must increase the problem size. For four cities, $p = 1$, $A = 50$ and distance matrix

$$D_4 = \begin{bmatrix} 0 & 0.3181 & 0.2272 & 0.0909 \\ 0.3181 & 0 & 0.1818 & 0.0454 \\ 0.2272 & 0.1818 & 0 & 1 \\ 0.0909 & 0.0454 & 1 & 0 \end{bmatrix} \tag{49}$$

the cost function looks as in figure [10].

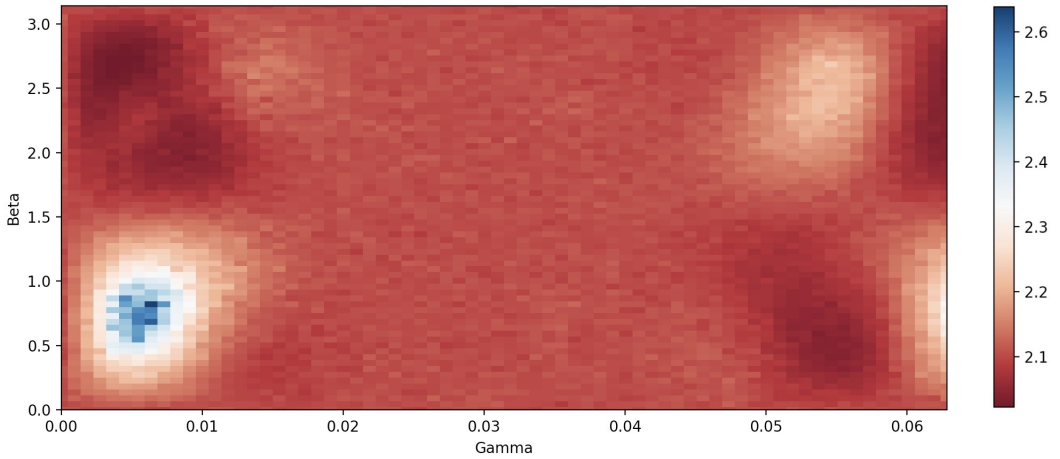


Figure 10: Average cost value of the result of a $p = 1$ QAOA circuit with distance matrix D_4 with 100 shots per parameter pair, $A = 50$ and a minimum at $\gamma = 0.00364$, $\beta = 2.641$.

One can see that the cost function is very similar shaped as the cost function for the $n = 3$ TSP.

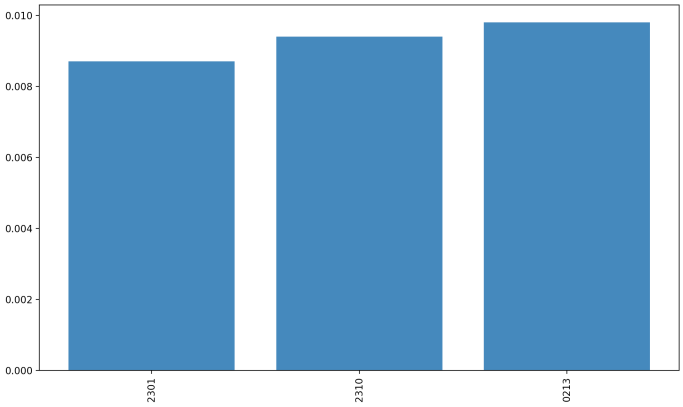


Figure 11: Filtered histogram of the results of a $p = 1$ QAOA circuit with distance matrix D_4, in the cost minimum with regards to β and γ at $(\gamma, \beta) = ((0.003642), (2.64075))$, 10000 shots and $A = 50$.

Tour	Probability	Path length
0213	0.0098	0.5454
2310	0.0094	1.5909
2301	0.0087	1.5909

Table 1: Filtered results of a $p = 1$ QAOA circuit with distance matrix D_4, in the cost minimum at $\gamma = 0.00364$, $\beta = 2.641$, 10000 shots and $A = 50$ and their path length.

If one has a look at figure [11] and table [1], one can see, now that more than one potential feasible solutions of the problem exists, more potential tours occur with different probabilities in the measurements. Remarkable is, that the algorithm already found the optimal tour with $\hat{p} \approx 1\%$, which is a larger probability then the other potential pathes. Therefore, the QAOA successfully solved this instance of a TSP with $n = 4$. But this results might be taken with a grain of salt, because if one has a further look at the distance matrix, one can see that this TSP instance might be easier to solve then others because the path length difference between the best tour and the second best tour being disproportional big.

3.1.3 Fine-tuning

To investigate the optimization potential of the Quantum Approximate Optimization Algorithm (QAOA) with respect to its hyperparameters and potential sources of error, I will now exermine the influence of varying parameters and other influencing factors on the quality of the obtained results. For that, suitable metric for quantitatively evaluating the results' quality must be derived.

The two main characteristics of measurements results are: the probability of obtaining the correct or shortest solution and its ratio to the probability of finding other pathes that are as well suitable but not the shortest ones.

For the latter, one can use the ratio of the probability to measure the shortst path and the probability to measure the most probable path excluding the shortest path. For the former, one can use the probability of the shortest path. In formular:

$$\rho_1 = \hat{p}_{best\ path} \tag{50}$$

$$\rho_2 = \frac{\hat{p}_{\text{shortest path}}}{\max(\{\hat{p}_i\} \backslash \{\hat{p}_{\text{shortest path}}\})} \tag{51}$$

With these metrics, ρ_1 and ρ_2, a series of measurements can be conducted to understand their dependency on multiple variables. These variables include A, the depth of the quantum circuit p, the inherent difficulty of the problem, a scaling factor applied to the distance matrix and finally, the hyperparameters of the classical optimizer such as the number of shots used to compute the cost value in the angle space and the grid size employed in a fine grid search for optimization.

The problem's difficulty, dd, can be quantified using the formula:

$$d = \frac{1}{\frac{L_{\text{second best path}}}{L_{\text{best path}}} - 1} \tag{52}$$

With L_i being the path length of path i. It is worth noting, that this investigation does not account the tendancy of the algorithm to get stuck in a local minima of the cost function in the angle space. But given that no logical local minima exist in the Traveling Salesman Problem (TSP) for $n = 4$, and considering the limitation of not being able to

test TSP instances with a greater number of cities, this factor remains unmeasurable within the scope of this study.

Before one tests the dependence of the problems factors on the result quality, one should test if the hyperparameters of the classical optimization of the angles are appropiately fine-tuned such that one can be sure the global minimum will be found up to an appropiate accuracy and one can exclude the optimizer as a potential source of errors.

Since the output of the QAOA for each angle set is statisticly distributed and a too small sample size per measurement could therefore be one of the main error sources for the optimizer, one should first test what the minimum sample size is such that the distribution of the measurement results and cost function becomes neglectable in the optimization process.

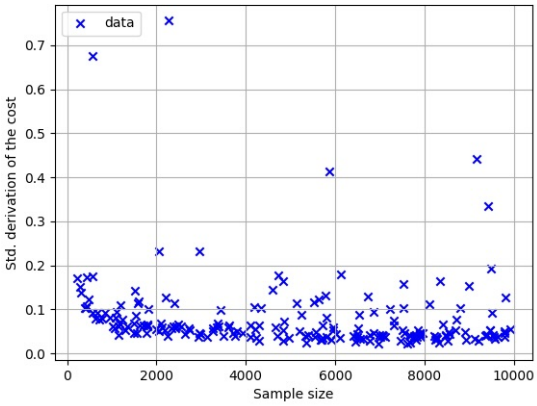


Figure 12: Empirical standard deviation $\bar{\sigma}(cost)$ of the cost function in a measurement sample in dependence of the sample size.

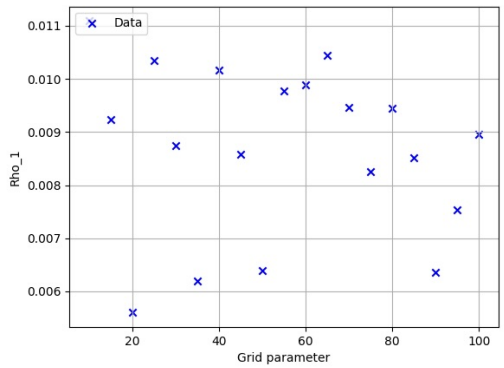
With having a look at figure [12], it becomes evident that the standard deviation of the cost function within a particular sample approaches zero in a statistical sense as the sample size increases. Furthermore, beyond a sample size of 2000, the rate of decrease in the standard deviation shows a disproportionate slowing down. So, a sample size of 2000 appears to be sufficient for most intents. However, it remains crucial to acknowledge that the cost values measured are still subject to statistical fluctuations around the actual cost values.

Another parameter of a fine grid search algorithm that needs to be tested is the grid parameter g. The grid parameter determines the grid of angles for which the optimizer calculates the cost and searches the minimum in:

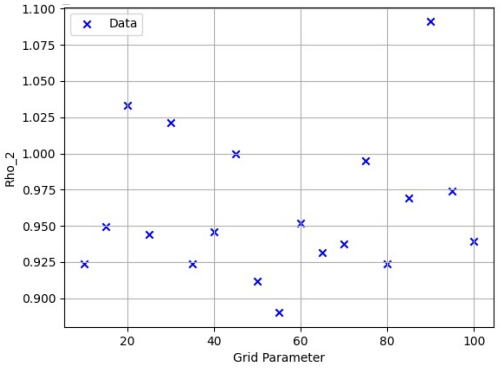
$$\mathcal{S} = \{\beta = it : 0 < i < g, t = \frac{2\pi}{g} \ : i \in \mathcal{N}\} \otimes \{\gamma = it : 0 < i < g, t = \frac{\pi}{g} \ : i \in \mathcal{N}\}, \quad (53)$$

with $\mathcal{S}$ being the search space of angles. To make an informed choice of g, one can

plot ρ_1 and ρ_2 after the optimization process, as a function of g, as demonstrated in figure [13].



(a) ρ_1 in dependence of the grid parameter g with distance matrix D_4 and $p = 1$ with the QUBO cost function. No correlation is recognizable.



(b) ρ_2 in dependence of the grid parameter g with distance matrix D_4 and $p = 1$ with the QUBO cost function. No correlation is recognizable.

Figure 13: $\rho_{1,2}$ in dependence of the grid parameter g for the n=4 QUBO TSP QAOA with the QUBO cost function

Upon examination of these plots, it becomes clear that there is no visible correlation between the grid parameter and the quality of the outcomes. A potential reason for that might be a weak correlation between the cost function and the metrics $\rho_{1,2}$.

To examine this error source one could randomly choose angles and plot ρ_1 in dependence of the cost function. This plot can be seen in figure [14].

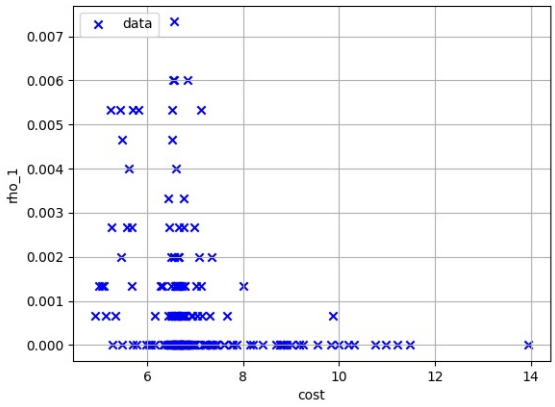


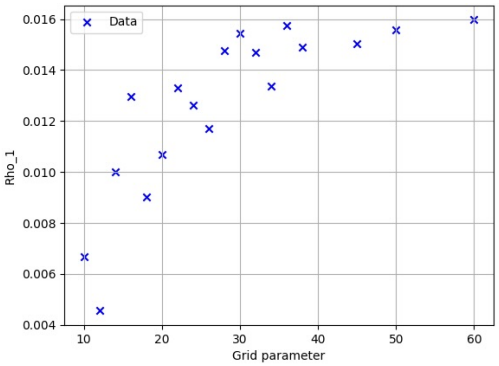
Figure 14: Derivation of 200 measurements of $(\rho_1, Cost)$ with a sample size of 2000 for randomly choosen angles.

It becomes visible that for samples in the angle space in higher cost regions, ρ_1 and

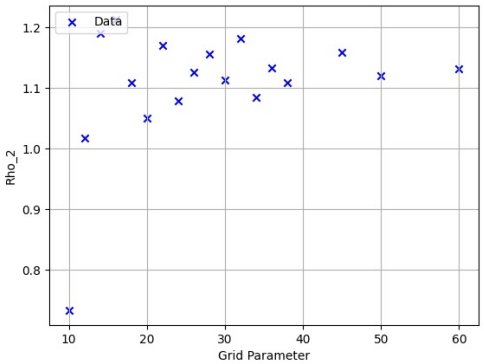
therewith the chance to measure the best route is most likely 0. Vice versa, samples with lower costs are more likely to result in larger values of ρ_1. This suggests on the one hand, that there is a correlation between ρ_1 and the cost. But one can also see, that samples in the maximum of ρ_1 are not necessarily in the cost minimum.

Which suggests on the other hand, that at least for the optimization of the hyperparameters of the problem, one might use a different cost function in the classical optimizer to optimize the angles on. An better choice for a cost function for that could be ρ_1 itself, even though one could not use it to solve real world instances because one needs the solution of the problem to calculate ρ_1. Also, one must assume that the best set of parameters in the optimization with ρ_1 are the same as for the optimization with the actual cost function.

To test $\rho_{1,2}$ as canditate to optimize the processes hyperparameters, one might plot $\rho_{1,2}$ in dependence of the grid parameter g and examine if one can observe a more noticeable trend.



(a) ρ_1 in dependence of the grid parameter g with distance matrix D_4 and $p = 1$ with $-\rho_1$ as the cost function.



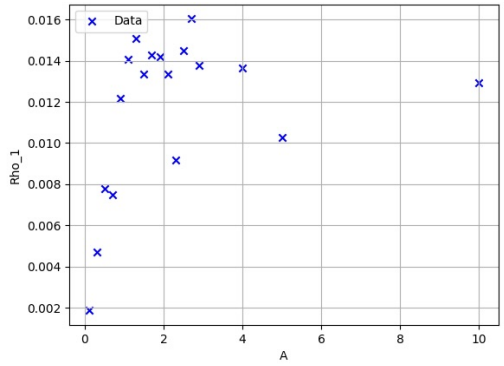
(b) ρ_2 in dependence of the grid parameter g with distance matrix D_4 and $p = 1$ with $-\rho_1$ as the cost function.

Figure 15: $\rho_{1,2}$ in dependence of the grid parameter g for the n=4 QUBO TSP QAOA with $-\rho_1$ as the cost function.

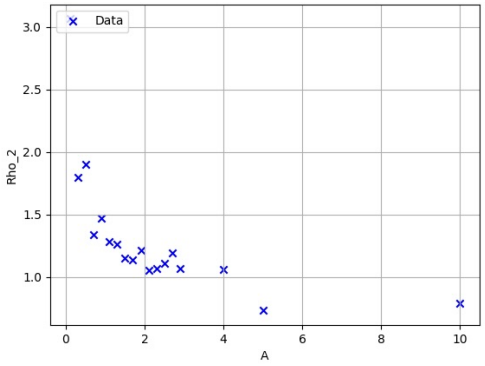
In figure 15 the same measurements as in figure 13 were made, but with $-\rho_1$ as the cost function. It is observable, that the grid parameter g has indeed an influence on the quality of the optimization results. With greater g, ρ_1 approaches a maximum value and ρ_2 converges. Since the results with $g \geq 40$ are nearly optimal and experiments with a greater grid parameter would take too long because the optimizer runtime grows with a factor of g^{2p}, I will use $g = 40$ in the following.

It is worth highlighting that the maximum value of ρ_1 is approximately 50% higher with 1.6%, which might be a demonstration that there could be better cost functions for the classical optimization then the QUBO TSP cost function.

With this in mind, the dependence of the parameter A in the quantum cost circuit on the quality of the results can now be investigated.



(a) ρ_1 in dependence of the cost circuit parameter A with distance matrix D_4 and $p = 1$ with $-\rho_1$ as the cost function.



(b) ρ_2 in dependence of the cost circuit parameter A with distance matrix D_4 and $p = 1$ with $-\rho_1$ as the cost function.

Figure 16: $\rho_{1,2}$ in dependence of the cost circuit parameter A for the n=4 QUBO TSP QAOA with $-\rho_1$ as the cost function.

It is evident that the variable A plays a significant role in the algorithm's performance. As A decreases, the ratio ρ_2 — representing percentage of the optimal route compared to percentage of the second best—increases. Conversely, as A increases, ρ_2 decreases and may even fall below one when A is either 5 or 10, indicating the algorithm's failure to find the optimal solution.

In contrast, the proportion of measurements yielding the optimal route either increases or stabilizes at a high level as A grows, while it decreases with smaller A values.

This behavior can be interpreted as follows: smaller A values result in a broader set of outcomes after optimization. However, the optimal solution still appears more frequently than other feasible ones, due to the large impact of path length differences on the overall cost. For larger A values, the solution set becomes more focused, causing ρ_1 to rise. At the same time, the contribution of path length differences to the cost becomes neglectable, resulting in more non-optimal but feasible routes being considered.

So, there is a trade-off: A must be large enough to produce a broad range of feasible solutions but small enough to ensure a high frequency of optimal route. Based on these insights, I plan to use $A = 1$ in future experiments.

Furthermore, these observations suggest another approach. One could initially use a high A value to filter out infeasible solutions, followed by a lower A value in later iterations to amplify the influence of path length differences on the solution set.

3.1.4 Limits

After extensively examining the optimizer's parameters, it is time to focus on the algorithm's most critical parameter: p. This parameter determines how often the mixer and the cost unitary operations are applied.

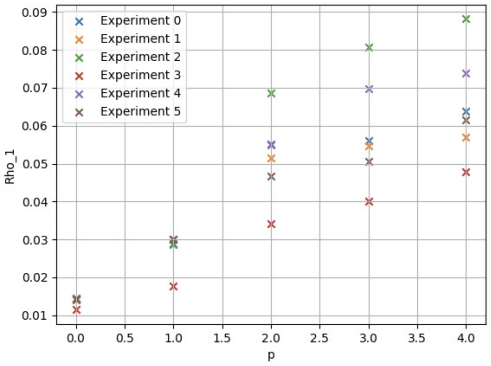
Since the runtime of the optimizer grows with a factor of g^{2p} if one uses a fine-grid search optimizer, researching the influence is exponentialy hard for greater values of

p. For instance, with $g = 40$, it becomes computationally infeasible to calculate for $p = 2$. However, a workaround can be applied: start by optimizing the first pair of angles (γ_1, β_1) for $p = 1$, and then sequentially optimize each successive pair (γ_i, β_i) for $p = i$, using the previously optimized angles. Mathematically, this can be represented as:

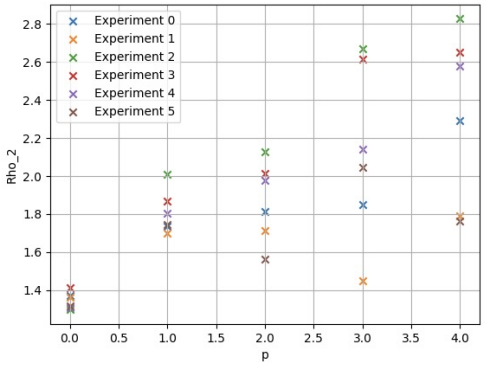
$$(\vec{\gamma}, \vec{\beta}) = ((\gamma_p, ..., \gamma_1), (\beta_p, ..., \beta_1)). \tag{54}$$

For that, one can see five series of experiments in figure 17. The TSP problem with distance matrix D_4 was used in each series and the QAOA with aboves optimization technique was applied.

It is evident that p significantly impacts the quality metrics $\rho_{1,2}$. For the initial p values, both $\rho_{1,2}$ indicators appear to grow linearly. However, this growth slows and eventually plateaus for $p > 3$. Unfortunately, even the linear growing runtimes for $p > 4$ make it unfeasible to investigate wether $\rho_{1,2}$ converge with larger values of p.



(a) ρ_1 in dependence of the parameter p, the number of times the mixer and the cost unitary get applied to the path encoding register. With distance matrix D_4 and $p = 1$ with $-\rho_1$ as the cost function.

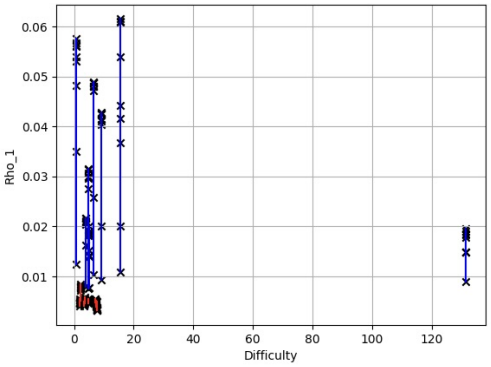


(b) ρ_2 in dependence of the parameter p, the number of times the mixer and the cost unitary get applied to the path encoding register. With distance matrix D_4 and $p = 1$ with $-\rho_1$ as the cost function.

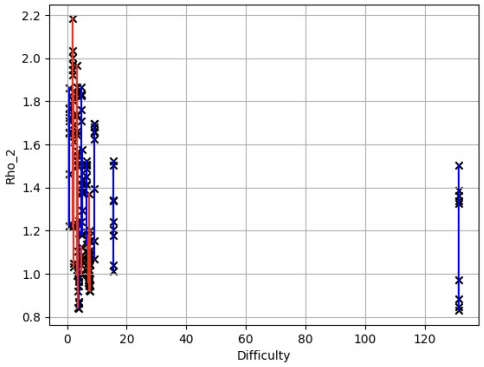
Figure 17: $\rho_{1,2}$ in dependence of the parameter p, the number of times the mixer and the cost unitary get applied to the path encoding register. With distance matrix D_4 and $p = 1$ with $-\rho_1$ as the cost function.

One should note that even though the same distance matrix D_4 was used for each series, all experiments had different outcomes. The reason for the difference can probably be traced back to the stochastic influence of the sample size.

Still to be explored is how the complexity of the problem impacts the findings. To assess this, one can create random symmetric TSP distance matrices, solve them using a brute-force algorithm, and quantify the problem's difficulty using equation 52. Subsequently, repeat the measurements displayed in figure 17, but plot ρ_i against the problem's complexity instead of p.



(a) Series of $\rho_1(p)$ in dependence of the problem difficulty with random distance matrices and $p \in \{1, ..., 9\}$ with the QUBO TSP cost (red) or $-\rho_1$ as the cost function (blue).



(b) Series of $\rho_2(p)$ in dependence of the problem difficulty with random distance matrices and $p \in \{1, ..., 9\}$ with the QUBO TSP cost (red) or $-\rho_1$ as the cost function (blue).

Figure 18: $\rho_{1,2}(p)$ in dependence of the problem difficulty with random distance matrices and $p \in \{1, ..., 9\}$ with the QUBO TSP cost or $-\rho_1$ as the cost function.

In figure 18, the outcomes of the experiment are displayed for eight series using the actual QUBO TSP cost function (in red) and another eight series using $-\rho_1$ as the cost function (in blue) within the classical optimization.

It is evident that the maximum values of $\rho_{1,2}(p)$ are influenced by the problem's complexity, as the maximum values of $\rho_{1,2}(p)$ tend to decrease with a higher problem difficulty. Further, optimizations using the actual QUBO TSP cost function show significantly poorer performance compared to those that use $-\rho_1$ as the cost function. This strengthens the assumption, that the QUBO TSP cost function might not be the best suitable option for the classical optimization process. One plausible explanation could be that a majority of potential states are not viable paths, thereby adding a penalty factor to the overall cost and possibly serving as a disruptive factor. With this in mind, I will exermine a more optimized representation of the problem and its QAOA in the following.

3.2 The Traveling Salesman Problem SIM-QAOA

In this section I will introduce another representation of the problem, that surpasses the QUBO QAOA in regards to the order of qubits and gates needed, the Traveling Salesman SIM-QAOA from [20]. The SIM-QAOA employs a binary encoding scheme for a city number visited at specific time t, requiring $\mathcal{O}(n \log n)$ qubits. This leads to fewer infeasible paths, reducing the number of bit string combinations that need to be penalized. This results in a smoother cost landscape in the angle space, potentially resulting in a more successful optimization process.

Using this encoding, the cost function can be expressed as:

$$C = \sum_{t=0}^{n-1} \sum_{\substack{i,j=0 \\ j \neq i}} W_{ij} \delta(b_t, i) \delta(b_{t+1}, j) + A[\{b_t\} \text{ is not a permutation}], \tag{55}$$

with b_t being the number of the city that is visited at time t, $[\phi]$ the Iverson notation

$$[\phi] = \begin{cases} 1 & \text{if } \phi \text{ is true} \\ 0 & \text{otherwise} \end{cases} \tag{56}$$

and δ the Kronecker delta.

The depth of this algorithm is as in the QUBO-QAOA also linear in p. The number of gates required is second degree polynomial, which outperforms the QUBO-QAOA, which requires a third degree polynomial number of gates with regards to the problem size.

3.2.1 SIM-QAOA Circuit for the TSP

To implement this cost function within a quantum circuit, the initial state preparation needs a closer examination, specifically the initialization of superposition across each of the n b_t registers. To store the city number for time t in a binary notation, each b_t register requires $\lceil \log n \rceil$ qubits and each register must be initialized to an equaly distributed superposition of the binary representations of $\{0, ..., n\}$

$$|b_t\rangle = \frac{1}{\sqrt{n}} \sum_{x=0}^{n-1} |x\rangle \,. \tag{57}$$

The previous approach to apply a Hadamard gate to every qubit to form a superposition of all $\{0,1\}^{\lceil \log n \rceil}$ states, is only applicable for instances where $n = 2^m$, $m \in \mathbb{N}$. Consequently, an alternative method for state preparation is needed.

The idea goes as follows: As the most significant bit is 0 in $2^{\lceil \log n \rceil - 1}$ of the n potential measurements of the superposition, specific gates must be employed to set the first bit as

$$\left|b_{t,\lceil \log n \rceil}\right\rangle = \sqrt{\frac{2^{\lceil \log n \rceil - 1}}{n}} |0\rangle + \sqrt{1 - \frac{2^{\lceil \log n \rceil - 1}}{n}} |1\rangle \,. \tag{58}$$

This state can be achieved with applying a Y-rotation with a suitable angle. Subsequently, controlled Hadamard gates can establish a uniformly distributed superposition across all minor bits, conditioned on the most significant bit being zero. The system thus becomes a superposition of all numbers up to $2^{\lceil \log n \rceil - 1}$. For numbers exceeding this range, probability amplitudes of the minor bits must be manipulated in a similar fashion but controlled by the prior bits. An example circuit for the initialization of $|b_t\rangle$ for $n = 14$ is depicted in figure [19].

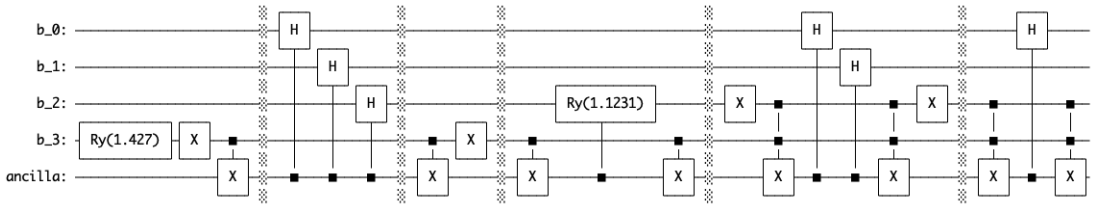


Figure 19: Circuit to generate an equal distributed superposition of binary representations of all numbers from 0 to 14 in the b-register in the little-endian convention.

The cost function unitary, that encodes aboves cost function can be devided into two parts. The first part takes care of the penalty part of the function, algorithmically verifying if the state represents a permutation. The second part calculates the actual path length encoded by the state.

The idea of the penalty part of the cost function is to use n ancilla qubits, go through all b-registers and flip the i-th ancilla qubit when a b-register contains the value i. Consequently, all ancilla bits will be set to one if the b-registers collectively form a permutation.

This algorithm can be represented in pseudocode as follows:

Algorithm 3: Penality part of the TSP SIM-QAOA cost unitary

Input:

Register $|b_t\rangle$, n ancilla bits $|c_i\rangle$, 1 ancilla bit $|flag\rangle$, penalty parameter A, γ - the trainable parameter of the ansatz

for $i = 1...n$ **do**
  $|c_i\rangle \leftarrow |0\rangle$
for $t = 1...n$ **do**
  for $i = 1...n$ **do**
    if $|b_t\rangle == i$ **then**
      $|c_i\rangle \leftarrow |c_i\rangle + 1$
if $\exists i : |c_i\rangle == 0$ **then**
  $|flag\rangle \leftarrow |1\rangle$
else
  $|flag\rangle \leftarrow |0\rangle$
$|flag\rangle \leftarrow R_Z(\gamma A)\,|flag\rangle$

A circuit to implement that can be seen in figure [20]. Here, $n = 2$ was used to have a clear structure.

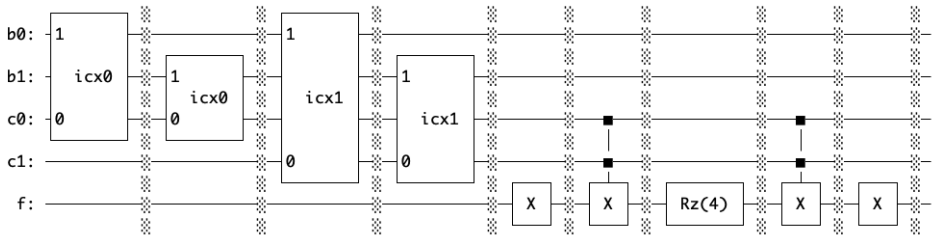


Figure 20: Circuit to implement the penalty part of the TSP SIM-QAOA cost function with $\gamma A = 4$, $n = 2$ and icx<int> being gates that flip the input 0 bit if the input 1 register is equal to the parameter <int>.

The **integer-controlled X-gate (icx<int>)** gate is designed to flip the input 0 qubit when the value in the register of input qubits $1, ..., \lceil \log n \rceil$ matches the specified integer parameter in binary notation. This is achieved by initially applying X-gates to the bits in the input register corresponding to positions where the binary representation of <int> contains a 0. Subsequently, a controlled X-gate is applied on the input 0 qubit, with the control conditioned on all bits in the input register. To revert the input 1 register to its original state, X-gates are reapplied to the same bits that were initially flipped. For <int> $= 5$ and q being input 0 bit one can see

it in figure 21.

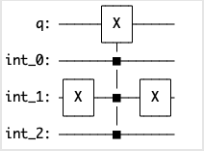


Figure 21: Circuit to implement the icx<int> gate for q as the input 0 bit and <int> = 5

Afterwards, the ancilla flag $|flag\rangle$ and the ancilla bits $|c_i\rangle$ must be reverted to their original states through uncomputation.

The procedure for implementing the path-length component of the cost function goes as follows. Initially, the path encoding in the $|b_i\rangle$ registers is transformed into a different representation stored in a new set of n ancilla registers, denoted as $|e_i\rangle$. These ancilla registers have the same size as the original $|b_i\rangle$ registers. In the new encoding, each $|e_i\rangle$ will hold the number of the city that will be visited after city i. With this representation, one can then iterate through all $|e_i\rangle$ registers and all potential city numbers j, applying a controlled Z-rotation that is proportional to the distance between cities i and j. This rotation is conditionally activated on a flag bit if the $|e_i\rangle$ register holds the integer value j.

For a conceptual understanding, consider the case where $n = 2$ and the path length is constant. A circuit implementing this algorithm is depicted in figure 22. In this circuit, 'ccopy' represents a controlled copy gate. This gate copies the value from the input 0 register to the input 1 register, conditioned on the input 2 qubit being set to 1.

Algorithm 4: Path length part of the TSP SIM-QAOA cost unitary

Input:

Register $|b_t\rangle$, n ancilla registers $|e_i\rangle$ of size $\lceil \log n \rceil$, n ancilla bits $|flag_i\rangle$, penalty parameter A and γ - the trainable parameter of the ansatz

for $i = 1...n$ **do**
  $|e_i\rangle \leftarrow \left|0^{\lceil \log n \rceil}\right\rangle$
  $|flag_i\rangle \leftarrow |0\rangle$
for $t = 1...n$ **do**
  for $i = 1...n$ **do**
    $|flag_i\rangle \leftarrow |b_t\rangle == |i\rangle$
    if $|flag_i\rangle$ **then**
      $|e_i\rangle \leftarrow |b_{t+1}\rangle$
uncompute $|flag_i\rangle$
for $i = 1...n$ **do**
  for $j = 1...n$ **do**
    $|flag_i\rangle \leftarrow |e_i\rangle == |j\rangle$
    $|flag_i\rangle \leftarrow R_Z(\gamma W_{i,j}) |flag_i\rangle$

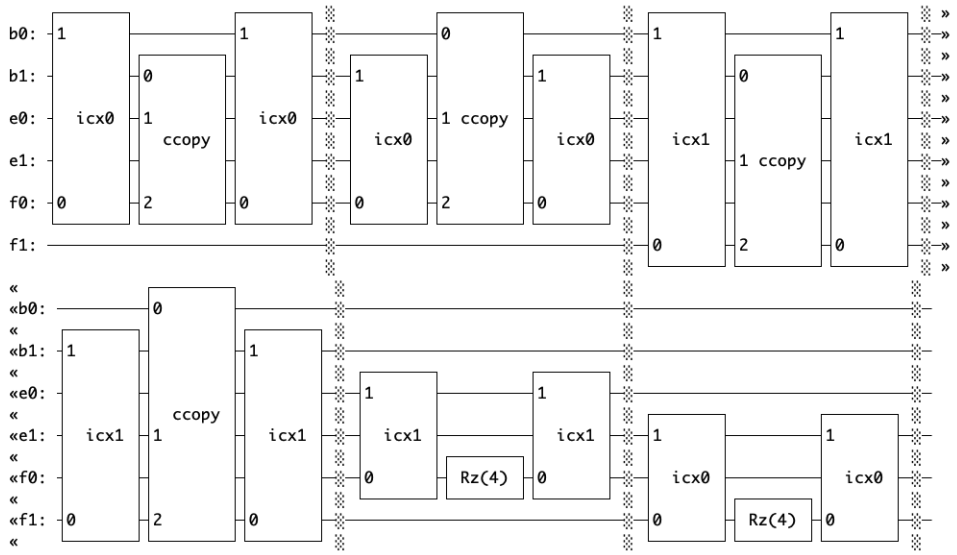


Figure 22: Circuit to implement the path length part of the TSP SIM-QAOA cost function with $W_{i,j}\gamma = 4$, $n = 2$ and ccopy as a controlled copy gate that does copies the value from the input 0 register to the input 1 register if the input 2 bit is 1.

The **controlled copy gate** can be implemented by n multicontrolled X-gates. Specifically, a multicontrolled X-gate is applied to each bit of the target register, with the control conditions set by the corresponding bit in the control register and the control-bit. A circuit diagram illustrating the implementation of a controlled copy gate for a 3-bit register is provided in figure [23].

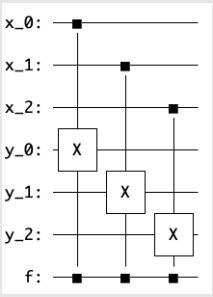


Figure 23: Circuit to implement the ccopy gate for 3 bit registers with $|x\rangle$ as the input register, $|y\rangle$ as the target register and $|f\rangle$ as the control-bit.

The uncomputation process for the $|e_i\rangle$ registers has not been outlined here, but it is a necessary step. Combined, these two algorithmic circuits form the cost unitary circuit for the TSP SIM-QAOA.

Due to the algorithm using a superposition of only a subset of possible measurable states in the path encoding registers $|b_i\rangle$, a standard mixer unitary, as used in traditional QAOA, is unsuitable. That would cause a probability greater then zero to measure a state in the $|b_i\rangle$ registers that does not encode a proper city number, which would be disadvantageous for the accuracy of the algorithm.

Instead, an alternative type of mixer, known as the Grover Mixer as described in [21], is used. This mixer has the property of effectively mixing outcomes but restricts it to a feasible set of states.

The Grover Mixer uses the composed unitary gate that initializes the $|b_i\rangle$ registers to an equal superposition of integers ranging from 0 to n for each register, defined by:

$$|b\rangle = (\frac{1}{\sqrt{n}} \sum_{x=0}^{n-1} |x\rangle) \otimes ... \otimes (\frac{1}{\sqrt{n}} \sum_{x=0}^{n-1} |x\rangle) =: U_S \left|0^{n\lceil \log n \rceil}\right\rangle =: |F\rangle \,. \tag{59}$$

With that, the Grover Mixer operation is given by:

$$GM(\beta) := e^{-i\beta |F\rangle\langle F|}. \tag{60}$$

This can be transformed as follows:

$$
\begin{aligned}
e^{-i\beta|F\rangle\langle F|} &= \sum_{k=0}^{\infty} \frac{(-i\beta)^k (|F\rangle \langle F|)^k}{k!} \\
&= \mathbb{I} + \sum_{k=1}^{\infty} \frac{(-i\beta)^k}{k!} |F\rangle \left(\langle F|F\rangle\right)^{k-1} \langle F| \\
&= \mathbb{I} - (1 - e^{-i\beta}) |F\rangle \langle F| \\
&= U_S(\mathbb{I} - (1 - e^{-i\beta}) |0\rangle \langle 0|) U_S^\dagger.
\end{aligned} \tag{61}
$$

Implementation-wise, this involves applying the conjugate transpose of the state preparation unitary $U_S^\dagger$, followed by a layer of X-gates, a multicontrolled phase gate, another layer of X-gates, and finally reapplying the state preparation unitary U_S. The composition of this gate for $n = 5$ and $\beta = -4\pi$ is displayed in figure [24].

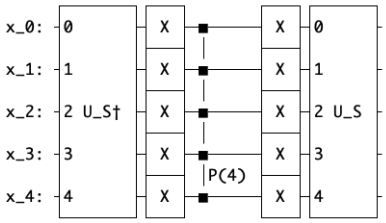


Figure 24: Circuit to implement a Grover mixer on 5 qubits with $\beta = -4\pi$.

With these components in place, the full TSP SIM QAOA circuit can now be implemented.

3.2.2 Results

In the results section, as done for the QUBO TSP, it's essential to first examine the cost functions distribution of measurement results at the cost minimum.

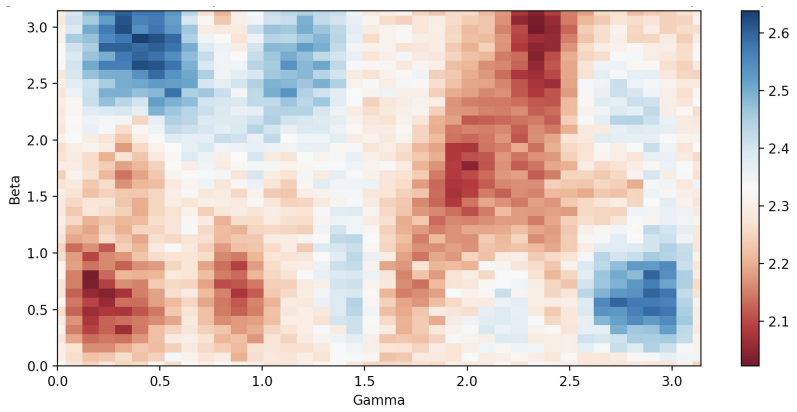


Figure 25: Cost function of the p=1 SIM-QAOA with distance matrix D_4 averaged over 2000 measurements per point without fine-tuned parameters.

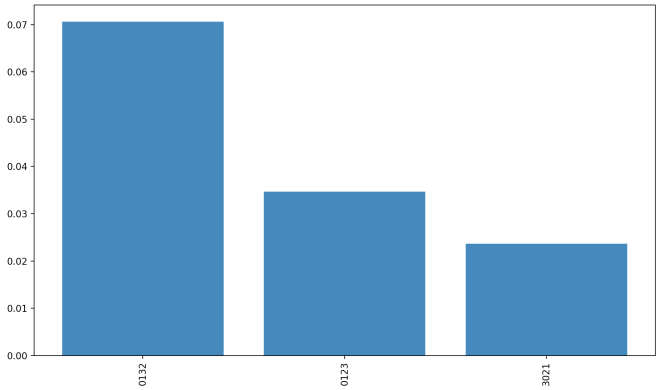


Figure 26: Filtered histogram of the results of the p=1 SIM-QAOA with distance matrix D_4 with $100,000$ measurements without fine-tuned parameters.

In figure 26, it is evident that a $p = 1$ optimization has significantly improved results compared to a $p = 1$ optimization for the QUBO QAOA representation, with an approximate ρ_2 value of 2 and a 7% probability of measuring the correct result. As illustrated in figure 25, the cost function appears to lack symmetries and possesses four local minima, similar to the $p = 1$ cost for the QUBO TSP.

3.2.3 Fine-tuning

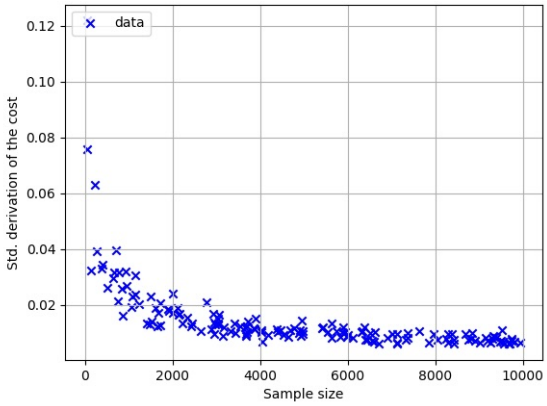


Figure 27: Empirical standard deviation $\bar{\sigma}(cost)$ of the cost function in a measurement sample in dependence of the sample size for the SIM-QAOA.

As depicted in figure [27], the standard deviation converges stochastically towards zero. However, when compared to the QUBO TSP, the rate at which this deviation decreases relative to the sample size is faster. It is evident that a sample size of around 3000 should be sufficient for the statistical variation in the cost value to become negligible.

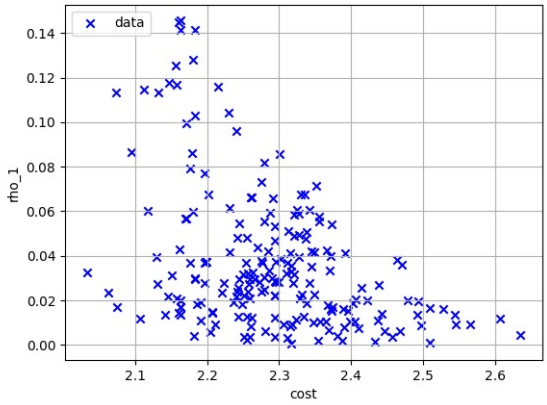


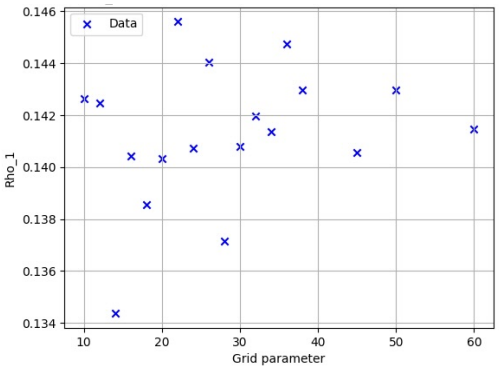
Figure 28: Distribution of $(\rho_1, cost)$ with a sample size of 2000 for 200 randomly choosen angles for the SIM-QAOA.

As illustrated in figure [28], the distribution of the likelihood to measure the correct path and the measurement's cost value, differs significantly from that of the QUBO TSP. In this case, there is no concentration of points for $\rho_1 = 0$. This may be attributed to

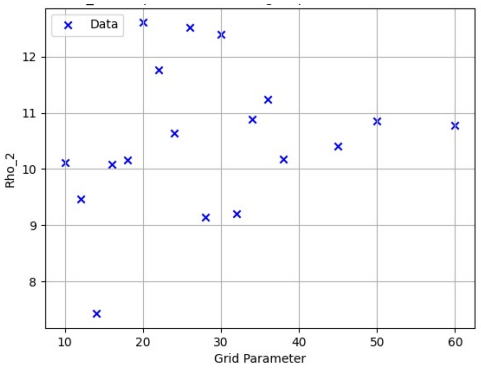
the reduced cardinality of the set of possible states, which increases the probability of measuring a correct result.

Further, one can see that the data points are more equaly distributed in the lower left triangle of the plot which indicates that the cost and ρ_1 are more correlated. Even though this distribution looks like a successful optimization could now be easier to archive, it would be even better if the ρ_1 and the cost would be even more correlated and the data points therewith closer distributed on a line from ρ_{max} and the maximum cost.

Also, since plots depicting the relationship between cost and other parameters, such as the grid parameter are still not evaluable because of no recognizable correlation, I will again use $-\rho_1$ as the cost function to find the optimal parameters for the optimization.



(a) ρ_1 in dependence of the grid parameter g with distance matrix D_4 and $p = 1$ with $-\rho_1$ as the cost function and the SIM-QAOA circuit.

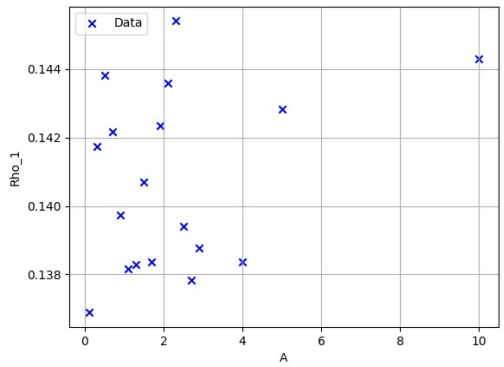


(b) ρ_2 in dependence of the grid parameter g with distance matrix D_4 and $p = 1$ with $-\rho_1$ as the cost function and the SIM-QAOA circuit.

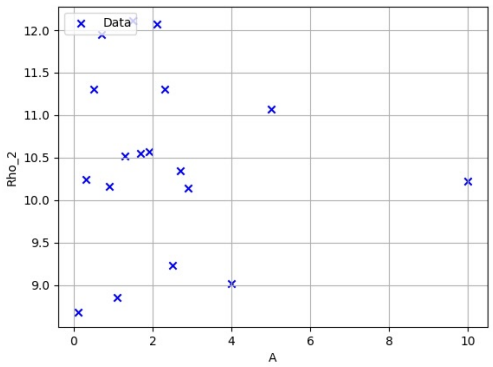
Figure 29: $\rho_{1,2}$ in dependence of the grid parameter g with distance matrix D_4 and $p = 1$ with $-\rho_1$ as the cost function and the SIM-QAOA circuit.

In the figure 29, it is evident that $\rho_{1,2}$ varies with the grid parameter in a manner similar to its behavior in the QUBO TSP. Here, the trend is less clear. One can again extract from the plot that an adequate value for the grid parameter is 40.

Regarding the cost-function parameter A, one can observe in figure 30 that the measurements for $\rho_{1,2}$ in dependence of parameter A show a similar trend for the SIM-QAOA as they do for the QUBO QAOA. Although, it's worth noting that the trend is considerably less distinct this time. To not choose A too high or too low again, I will proceed to use $A = 5$ in the following analysis.



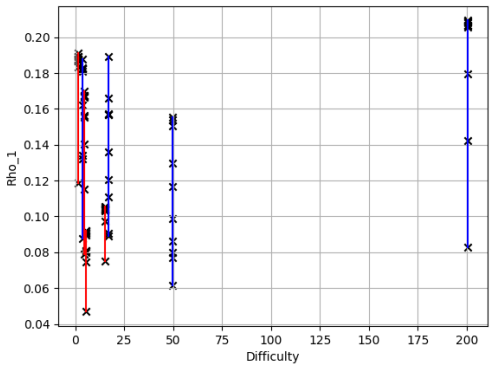
(a) ρ_1 in dependence of the cost circuit parameter A with distance matrix D_4 and $p = 1$ with $-\rho_1$ as the cost function and the SIM-QAOA Circuit.



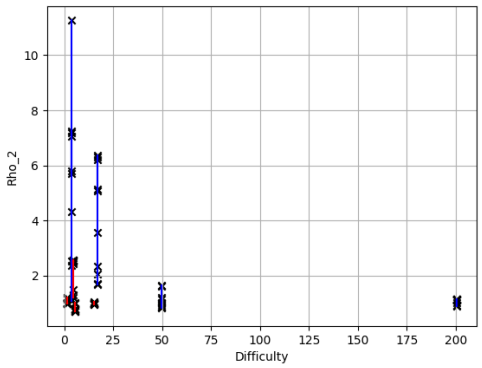
(b) ρ_2 in dependence of the cost circuit parameter A with distance matrix D_4 and $p = 1$ with $-\rho_1$ as the cost function and the SIM-QAOA Circuit.

Figure 30: $\rho_{1,2}$ in dependence of the cost circuit parameter A for the n=4 QUBO TSP QAOA with $-\rho_1$ as the cost function and the SIM-QAOA Circuit.

3.2.4 Limits



(a) Series of $\rho_2(p)$ in dependence of the problem difficulty with random distance matrices and $p \in \{1, ..., 9\}$ with the SIM-QAOA cost or $-\rho_1$ as the cost function.



(b) Series of $\rho_2(p)$ in dependence of the problem difficulty with random distance matrices and $p \in \{1, ..., 9\}$ with the SIM-QAOA cost or $-\rho_1$ as the cost function.

Figure 31: Series of $\rho_2(p)$ in dependence of the problem difficulty with random distance matrices and $p \in \{1, ..., 9\}$ with the SIM-QAOA cost or $-\rho_1$ as the cost funciton.

In figure 31, one can see that the values for $\rho_1(p)$ have significantly improved compared to the values for $\rho_1(p)$ in the QUBO TSP. The maximum values have increased from previously up to 6% to now up to 21%. It is also noticeable that the performance of the

experiments using the actual cost function is now roughly comparable to those where ρ_1 was used as the cost function. Moreover, the difficulty of the distance matrix seems to have little or no influence on the quality of the results of $\rho_1(p)$.

In addition to the significantly better results for ρ_1, the results for ρ_2— that is, the ratio of the probability of measuring the best tour to the probability of measuring the second-best tour — have also improved. Here, peak values of 11 were achieved. However, it is visible that the measurement series with ρ_1 as the cost function performed considerably better. It's also evident that some measurement series barely exceeded a value of 1, thus not properly solving the problem instance. Additionally, the quality of the results for ρ_2 appears to decrease with the difficulty of the TSP instance in contrast to the results forρ_1.

Unfortunately, only a few measurements could be made for this plot, as simulating the algorithm for four cities already nearly required unfeasible amounts of computational ressources.

3.3 The Permutation Index QAOA

When examining the results of the SIM-QAOA implementation, one may wonder why it has significantly better outcomes compared to the QUBO TSP QAOA. A plausible explanation could be the difference in the cardinality of the sets of potential states. In the case of the QUBO TSP, the cardinality is 2^{n^2}, significantly greater than that of the SIM-QAOA, which is $2^{n \log n}$. This larger set size might introduce more noise in the averaged cost function over numerous measurements or demand greater precision from the optimizer to achieve comparable results. Moreover, the necessity to penalize more outcomes could increase this effect.

To investigate that further, I developed an encoding and a cost function circuit for the TSP, that eliminates the need for penalty terms, as all potential outcomes will represent valid paths. The number of bits needed to encode a path will therefore be the natural limit $\log(n!)$. The cardinality of the set of possible outcomes can then be estimated as follows:

$$\begin{aligned} 2^{\log n!} &= 2^{n \log n - n \log e + \Theta(\log n)} \\ &\approx 2^{n \log n - n \log e} \\ \Rightarrow 2^{\log n!} 2^{n \log e} &\approx 2^{n \log n}. \end{aligned} \tag{62}$$

This size is approximately an exponential factor smaller than the set of potential outcomes for the SIM-QAOA. The idea of the encoding is to initialize a register $|p\rangle$ that stores the index of the permutation among all possible permutations of the cities. Both the cost and mixer unitaries would then act on this register.

3.3.1 PI-QAOA Circuit for the TSP

The Permutation Index QAOA (PI-QAOA) circuit shares several similarities with the SIM-QAOA circuit. In both algorithms, the state initialization process is nearly identi-

cal. However, for PI-QAOA, the state is initialized in a single register that holds integers in binary format, ranging from 0 to $(n-1)!-1$. The initialized state is represented as follows:

$$|p\rangle = |F\rangle = U_S \left|0^{\lceil \log (n-1)!-1 \rceil}\right\rangle = \frac{1}{\sqrt{(n-1)!}} \sum_{x=0}^{(n-1)!-1} |x\rangle \,. \tag{63}$$

As in the SIM-QAOA, the Grover Mixer is used for the mixer unitary, with U_S and $|F\rangle$ defined as above. The difference between the two approaches is the cost function.

For the PI-QAOA, the objective is to first convert the permutation index stored in the $|p\rangle$ register into the SIM-QAOA's path encoding in the $|b_t\rangle$ registers. The SIM-QAOA cost circuit is then applied to this transformed state, but the penalty terms are omitted. The idea how to transform the permutation index into the path encoding goes as follows.

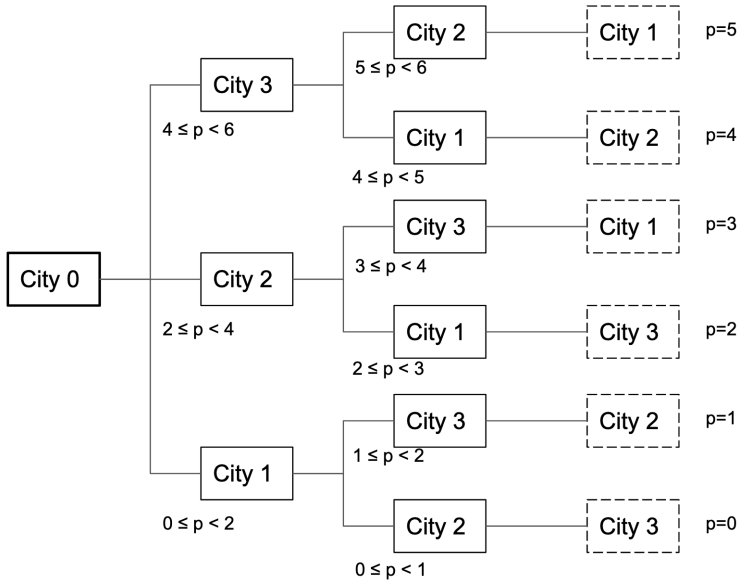


Figure 32: The decision tree diagramm for a $n = 4$ TSP with the potential intervals for the permutation index. Without loss of generality, the traveling salesman starts at city 0 - the root node - and decides to which city to go next in every layer. Every path from the root node to a leave node forms a potential path for the traveling salesman. And every layer between the first and last layer narrows down the feasible interval for the permutation index.

The Traveling Salesman Problem (TSP) can be visually represented as a tree diagram, with the starting point at city 0 serving as the root node. From this root, subsequent layers of nodes represent the choices for the next unvisited city to visit, ending in leaf nodes that signify the last cities that can be visited. In this representation, any path from the root node to a leaf node corresponds to a possible complete path for the traveling salesman.

With this concept, every subsequent city decision progressively narrows the potential permutation indices for the remainder of the tour. But more importantly, this relationship is reverseable. By iteratively determining if the permutation index lies within increasingly restrictive intervals, we can extract all necessary details to reconstruct the respective city sequence or permutation from the permutation index.

The pseudocode outlined in algorithm 5 provides a structure for this algorithm.

Algorithm 5: Classical permutation index to permutation transformation

Input:

Permutation index p

Output:

Path b_t

```
1 min ← 0
2 for j = 0, ..., n − 2 do
3     for i = 0, ..., n − 1 − j do
4         if min + i(n − 2 − j)! ≤ p < min + (i + 1)(n − 2 − j)! then
              // the permutation is in interval number i at decision j
5             b_j = at(i, j)
6             min += i(n − 2 − j)!
7             break
8 return b_t
```

With "at" being a function that manipulates one or more b_t registers with the interval number i and decision number j as inputs in a way that the whole procedure maps permutation indices in $|p\rangle$ to permutations in $|b_j\rangle$. One way to implement this is to initially set b_t to t and use

$$\text{at}(i, j) := \text{SWAP}_{b_j, b_{j+i}}. \tag{64}$$

Meaning, to swap register b_j with register b_{j+i} if the permutation index is in interval i of decision j.

On a quantum computer, the situation is more complex. While the underlying framework remains similar, the necessity for the corresponding operator to be unitary—and thus for the algorithm to be invertible—requires storing prior interval boundaries in ancillary registers ($|\text{offset}_i\rangle$).

All in all one needs for this version:

1. $n - 3$ ancilla registers $|\text{offset}_i\rangle$ of size $\lceil \log (n-1)! \rceil$

2. two ancilla registers $|min\rangle$, $|max\rangle$ of size $\lceil \log{(n-1)!} \rceil$ and $\lceil \log{(n-1)!} \rceil + 1$
3. two single bit ancilla registers $|\geq\rangle$, $|<\rangle$
4. the permutation register $|p\rangle$ of size $\lceil \log{(n-1)!} \rceil$
5. n output city path registers $|b_t\rangle$ of size $\lceil \log{n} \rceil$

Which sums up to

$$n\lceil \log{(n-1)!} \rceil + n\lceil \log{n} \rceil + 3 \tag{65}$$

qubits. The overall complexity of this is $\mathcal{O}(n \log{n!})$, which is less efficient than both the QUBO and SIM QAOA methods. It is worth noting that most of this qubit overhead stems from the offset registers. A more compact way to encode previous choices could potentially reduce the algorithmic complexity to $\mathcal{O}(n \log{n})$, similar to the SIM-QAOA approach. For instance, one could store the interval number of the permutation indexes in binary format for each decision. This improvement alone would result in a complexity similar to that of the SIM-QAOA with $\mathcal{O}(n \log{n})$.

Subsequently, the output registers will serve as input for the SIM-QAOA Hamiltonian, excluding the penalty term. The runtime on the other hand is equal to the runtime of the SIM-QAOA with a second degree polynomial overhead.

The quantum algorithm can be summarized in pseudocode, as in algorithm [6].

As mentioned, in contrast to the classical approach, the quantum version requires keeping track of the offsets from previous decisions. These offsets represent the minimum values of the intervals where the permutation index was located in the last decision layers. To set the interval boundaries for each decision layer, one can use the offset from the preceding layer. One then loops through the possible permutation index intervals to see if the index falls within the given boundaries. If one finds a match in a specific interval, one updates the offset for that layer and performs a swap operation. Finally, one resets all ancillary registers.

Algorithm 6: Quantum permutation index to permutation transformation

Input:

Permutation index register $|p\rangle$,
city registers $|b_t\rangle$,
$n-3$ ancilla registers $|\text{offset}_i\rangle$,
two ancilla register $|min\rangle$,$|max\rangle$,
two ancilla registers $|\geq\rangle$, $|<\rangle$.

Output:

Number of city visited at time t in $|b_t\rangle$

for $j = 0, ..., n-1$ **do**
  $|b_j\rangle \leftarrow |j\rangle$
for $j = 0, ...n-4$ **do**
  $|\text{offset}_i\rangle \leftarrow |0\rangle$
$|min\rangle \leftarrow 0$
$|max\rangle \leftarrow 0$
// for each layer j in the decision tree
for $j = 0, ..., n-2$ **do**
  // prepare the interval boundaries
  if $j \neq 0$ **then**
    $|min\rangle$ += $|offset_{j-1}\rangle$
    $|max\rangle$ += $|offset_{j-1}\rangle$
  $|max\rangle$ += $|(n-2-j)!\rangle$
  // for each potential city to go next
  for $i = 0, ..., n-1-j$ **do**
    $|\geq\rangle \leftarrow \text{GEQ}(min, p)$
    $|<\rangle \leftarrow \text{LESS}(max, p)$
    // check wether the permutation index is in the interval of the city
    if $|\geq\rangle$ *and* $|<\rangle$ **then**
      SWAP($|b_{j+1}\rangle, |b_{j+1+i}\rangle$)
      $|\text{offset}_j\rangle = |min\rangle$
    uncompute $|\geq\rangle$,$|<\rangle$
    // increase the interval boundaries
    $|min\rangle$ += $|(n-2-j)!\rangle$
    $|max\rangle$ += $|(n-2-j)!\rangle$
  // uncompute ancilla registers for this layer
  $|max\rangle$ −= $|(n-2-j)!\rangle$
  for $i = 0, ..., n-1-j$ **do**
    $|min\rangle$ −= $|(n-2-j)!\rangle$
    $|max\rangle$ −= $|(n-2-j)!\rangle$
  if $j \neq 0$ **then**
    $|min\rangle$ −= $|offset_{j-1}\rangle$
    $|max\rangle$ −= $|offset_{j-1}\rangle$
uncompute $|\text{offset}_i\rangle$

For this, several higher-level gates need to be introduced. To perform the operation of adding the values stored in two registers, as in $|min\rangle$ += $|\text{offset}_i\rangle$, one can use a CDKMRippleCarryAdder as described in [22]. The inverse of this operation can similarly be employed to execute subtraction between the values in two registers.

More complex operations like adding a constant integer to one in a quantum register, as well as comparison operations such as LESS and GEQ, will be elaborated in the following.

The comparison gates **LESS and GEQ** can both implemented using the same logic. The GEQ gate is logically a NOT-LESS gate and can thus be implemented with a LESS gate and a bit flip on the output. To implement the LESS gate, which compares integers x and y and determines if x is less than y, one could iteratively go through all the bits of x, starting with the most significant. In each iteration, a controlled-X gate is applied, controlled by the corresponding y-bit. Then, a controlled X-gate on the result bit controlled by mentioned y and x bits and all prior x bits is applied. Subsequently, an ordinary X-gate is applied on the x bit of the current order. See figure [33].

After processing all bits of x, the result will be stored in the output bit. However, the original state of the x register must be restored. This can be accomplished by repeating the loop without the controlled-X gates affecting the result bit.

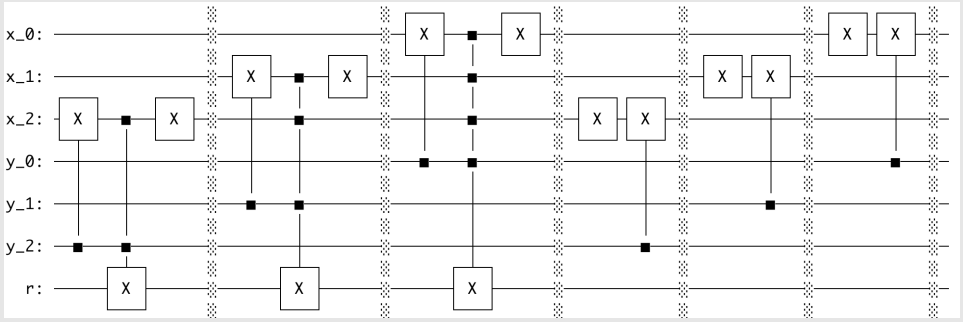


Figure 33: Circuit to implement the LESS gate, that returns in bit r if the value of the integer in register x is smaller than the value of the integer stored in register y. Both integers must be in the little-endian notation.

To implement the gate that **adds a constant to an integer in a quantum register**,

the constant should first be converted to its binary representation. Then, loop through its bitstring, starting from the least significant bit and moving to the most significant one. If a bit in the constant's binary string is zero, no operation is required. However, if a bit is one, an X-gate should be applied to the corresponding bit in the quantum register. After this, another loop initiates from the current bit

to the most significant bit, applying a not-multi-controlled X-gate at each bit along the way. These not-multi-controlled X-gates are conditioned by all previous bits up to the first bit that is zero.

In figure 34 you can observe a circuit where a 5-bit quantum register x is incremented by a constant value of 14. Here, "nmxc"-gates represent not-multi-controlled X-gates conditioned on the control bit being zero.

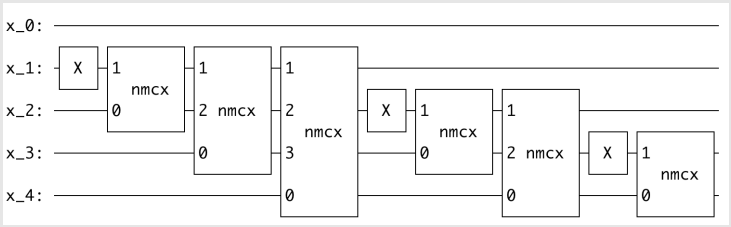


Figure 34: Circuit to implement the gate, that adds a constant to a quantum register. Here, the 5 bit sized register in little endian notation gets incremeted by 14. The "nmcx" gates are multicontrolled X-gates on the first input bit, that are applied if all control bits are 0.

In figure 35 a **not-multi-controlled X-gate** is implemented with q_0 being the result bits and all others being the control bits. The result bit will be flipped if all control bits are zero.

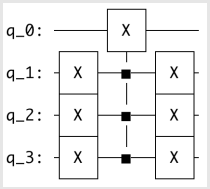


Figure 35: Circuit to implement the "nmcx"-gate, that applies a X-gate, if all control bits are 0. Here, q_0 is the target bit, $q_{1,2,3}$ are the control bits.

With the explanation of these gates now in place, we can have a look at the conceptual circuit depicted in figure 36. This circuit transforms the $n = 3$ permutation index stored in the single-bit register $|p\rangle$ into the $n \log n$ representation of a path in the SIM-QAOA, stored in the $|b_t\rangle$ registers, with the algorithm described in [6].

The example uses a three city instance for illustration purposes, as attempting to display this for four or more cities would become too large for this format. It is worth noting that for the case of three cities, the circuit does not require the use of $|\text{offset}_i\rangle$ ancilla registers and therewith has no need for their uncomputation. To find a more complex example for $n = 4$ in the appendix [41].

Also, the full circuit of the PI-QAOA circuit for $p = 2$ and $n = 3$ can be seen in figure 37.

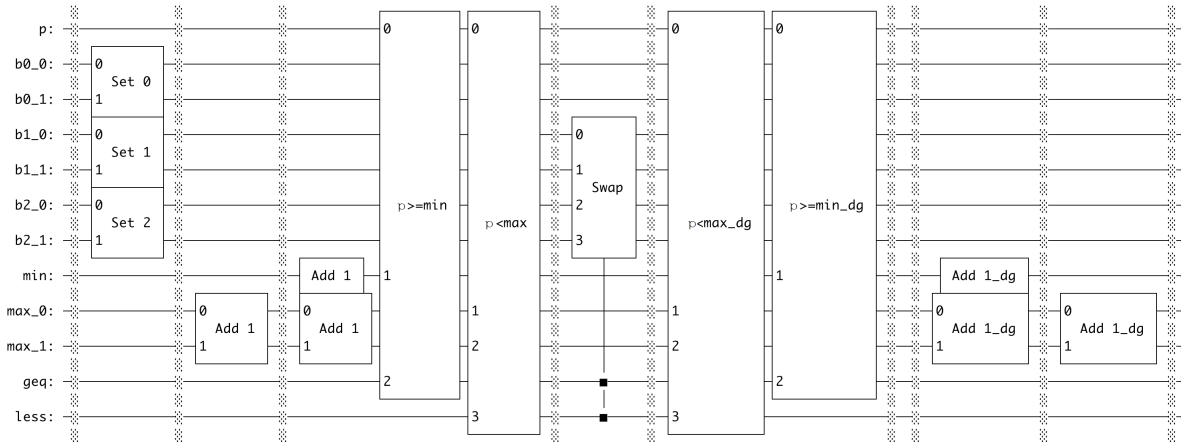


Figure 36: Circuit of the permutation index transformation with $|p\rangle$ as the permutation index and $|b_t\rangle$ being the output registers for $n = 3$. "_dg" stands for the inverse gate

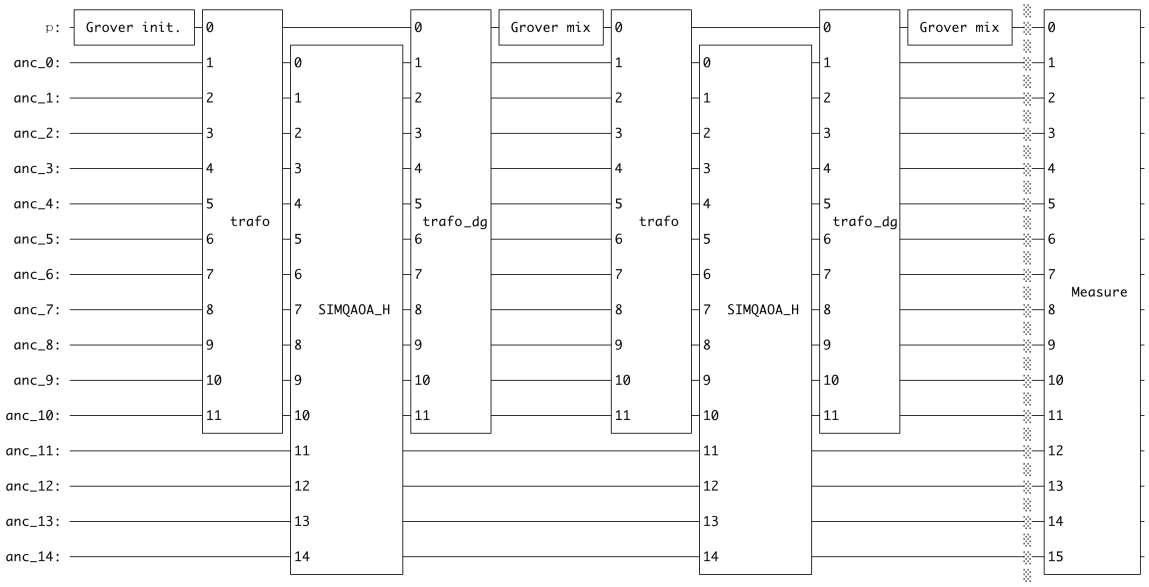


Figure 37: Conceptual circuit of the PI-QAOA TSP circuit with $n = 3$ and $p = 2$. <trafo> denotes the transformation unitary that generates the path in $n \log n$ representation from the permutation index stored in bit p. "_dg" stands for the inverse gate

3.3.2 Results

When examining the cost function shown in figure 38, it is evident that this function is smoother compared to those of SIM-QAOA or QUBO-QAOA. This function has only two distinct global minima and two distinct global maxima. This makes it appear more suitable for optimization using gradient methods or other subexponential techniques, in contrast to the cost functions from other representations. However, one possible reason for this clarity might be the limited number of possible outcomes for the circuit. As the size of the TSP problem increases, this cost function may also become more uneven.

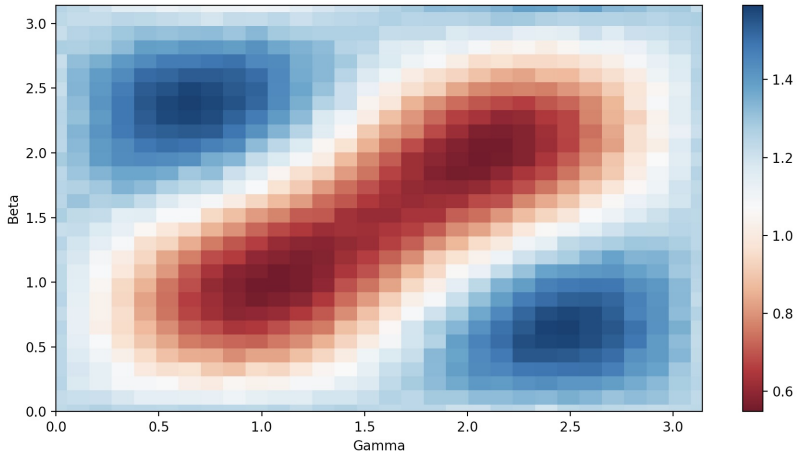


Figure 38: Cost function of the p=1 PI-QAOA with distance matrix D_4 averaged over 2000 measurements per point without fine-tuned parameters.

In figure 39, 100,000 measurements were made at the angle values corresponding to the cost function's minimum. The outcomes are telling. The optimal tour was observed with almost 100% probability, while the less favorable tours occurred only a handful of times, with a probability close to 0.05%. This represents am improvement of a factor of 10 in the likelihood of finding the best tour compared to other algorithms. However, it is important to note that this algorithm is searching for the optimal solution among just six possible tours. In contrast, SIM-QAOA has to find the best tour among 256 options, and QUBO-QAOA among 65,626. Therefore, the complexity of the optimization task is significantly lower for this algorithm.

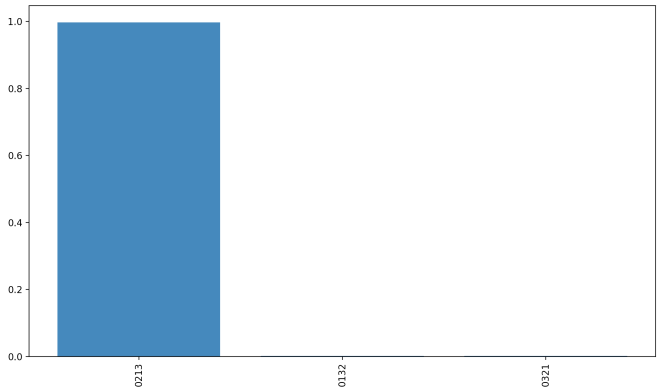
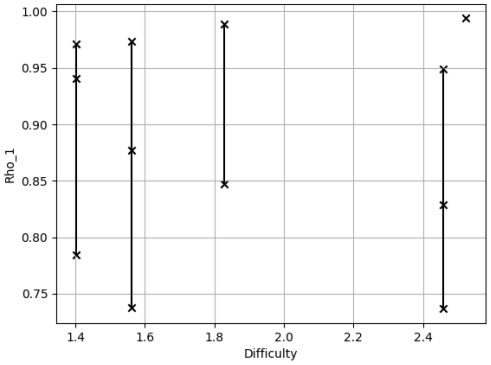
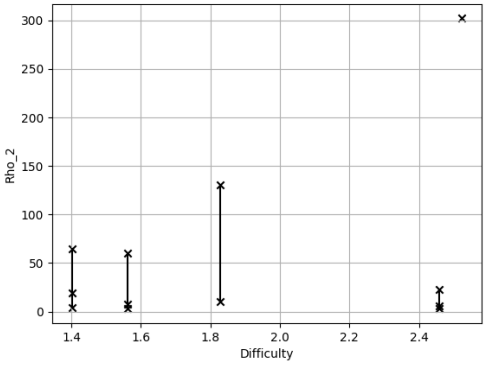


Figure 39: Filtered histogram of the results of the p=1 PI-QAOA with distance matrix D_4 with $100,000$ measurements without fine-tuned parameters.

Given that the algorithm consistently achieves nearly 100% accuracy for various random optimization parameters tested, there is no point in optimizing the other parameters. However, to make a comparison with other TSP QAOA versions, it would be beneficial to evaluate the algorithm's accuracy across different problem complexities.



(a) Series of $\rho_2(p)$ in dependence of the problem difficulty with random distance matrices and $p \in \{1, ..., 3\}$ with the PI-QAOA.



(b) Series of $\rho_2(p)$ in dependence of the problem difficulty with random distance matrices and $p \in \{1, ..., 3\}$ with the PI-QAOA.

Figure 40: Series of $\rho_2(p)$ in dependence of the problem difficulty with random distance matrices and $p \in \{1, ..., 3\}$ with the PI-QAOA.

In Figure 40, it becomes evident that the algorithm reliably achieves a probability of over 95% for $p \geq 3$, regardless of the difficulty of the problem. It's important to note for these plots that an accuracy of $> 95\%$ was a termination condition for the optimization to keep the runtime within a manageable range.

Regarding the initial question of whether allowing and penalizing infeasible tours adversely affects result quality, it appears that optimizing without considering these infeasible tours result in better outcomes. However, for a truly comparable analysis, the PI-QAOA should be tested on larger problem instances to ensure that the sets of possible outcomes have similar sizes as well.

4 Conclusion and Outlook

In summary, traditional methods for solving the Traveling Salesman Problem (TSP) fall into two categories. They are either approximate solutions or computationally inefficient, making them practical only for solving instances with a few thousand cities or less. For a majority of practical applications where TSP solvers are currently used, these classical techniques may be enough. They can often find approximate solutions in a reasonable time frame. However, there are specific areas where improved methods for approximating or even solving larger instances would be highly advantageous.

The findings of this thesis offer a review regarding the effectiveness of the Quantum Approximation Optimization Algorithm (QAOA) as an alternative method for solving the TSP. On the positive side, the data shows that the QAOA can indeed solve small, four-city instances of the TSP exactly with a proper encoding of the problem, even with limited flexibility in adjusting the QAOA parameter p. However, this result alone does not provide evidence for the QAOA being potentially superior to classical approaches for this problem.

One has to examine several concerns bevor considering the QAOA as a potential better algorithm for the TSP.

Firstly, scalability is a critical concern. While the quantum component of the algorithm displays a polynomial runtime, the classical optimizer that finds the optimal angles exhibits an exponential growth in runtime with respect to the parameter p, as a grid search was used in this thesis. This necessitates further investigation into two directions: One could either explore efficient classical optimizers if they could perform global optimization of the angles, or study how the minimum value of p needed to provide meaningful results scales. Additionally, it is important to evaluate the probability of measuring the optimal solution. If this probability decreases exponentially as the problem size increases, then the QAOA would not be a practical solution for larger TSP instances, as the number of experiments needed to find the minimum would become impractical due to exponential growth.

Secondly, the hardware requirements in the context of current quantum computing technology must be assessed. Since quantum computing remains an active research area with undefined boundaries concerning noise robustness, the number of qubits available, and superposition stability, it remains uncertain whether near-future quantum devices will be capable of running the QAOA for real-world, large-scale TSP instances.

To conclude, the QAOA has shown feasibility for solving the TSP with four cities based on the experiments in this thesis. To evaluate its potential for larger instances of the TSP, either a significant advancement in classical hardware is needed to simulate these larger instances, or we would require quantum computing devices that are robust against noise and have a sufficient number of qubits. Then, further research could be conducted to ascertain whether the QAOA could outperform existing classical algorithms — exact or approximate — in efficiency, either now or in the future.

List of Figures

5 References

[1] Wikipedia (2023). Quantum optimization algorithms, (https://en.wikipedia.org/wiki/Quantum_optimization_algorithms)

[2] Wolfram (2023). Visualization of the Traveling Salesman Problem, (https://mathworld.wolfram.com/TravelingSalesmanProblem.html)

[3] Haider Abdulkarim, Ibrahim Fadhil Alshammari (2015). Comparison of Algorithms for Solving Traveling Salesman Problem, (https://www.researchgate.net/publication/280597707_Comparison_of_Algorithms_for_Solving_Traveling_Salesman_Problem)

[4] Edward Farhi, Jeffrey Goldstone, Sam Gutmann (2014). A Quantum Approximate Optimization Algorithm, (https://arxiv.org/pdf/1411.4028.pdf)

[5] Wikipedia (2023). Travelling salesman problem, (https://en.wikipedia.org/wiki/Travelling_salesman_problem)

[6] Wikipedia (2023). Nearest Neighbour Algorithm, (https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm)

[7] Wikipedia (2023). The Algorithm of Christofides and Serdyukov, (https://en.wikipedia.org/wiki/Christofides_algorithm)

[8] William Cook, André Rohe (1999). Computing Minimum-Weight Perfect Matching, (https://www.math.uwaterloo.ca/~bico/papers/match_ijoc.pdf)

[9] Wikipedia (2023). Spannbaum, (https://de.wikipedia.org/wiki/Spannbaum)

[10] Keld Helsgaun (2009). General k-opt submoves for the Lin-Kernighan TSP heuristic, (https://www.researchgate.net/publication/227121797_General_k-opt_submoves_for_the_Lin-Kernighan_TSP_heuristic)

[11] Frank Neumann, Dirk Sudholt, Carsten Witt (2009). Computational Complexity of Ant Colony Optimization and Its Hybridization (https://link.springer.com/chapter/10.1007/978-3-642-04225-6_6)

[12] Wikipedia (2023). Ant colony optimization algorithms, (https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms)

[13] Wikipedia (2023). Branch and cut, (https://en.wikipedia.org/wiki/Branch_and_cut)

[14] Wikipedia (2023). Branch and bound,
(https://en.wikipedia.org/wiki/Branch_and_bound)

[15] William Cook (2015). Concorde,
(http://www.math.uwaterloo.ca/tsp/concorde/)

[16] Wikipedia (2023). Concorde TSP solver,
(https://en.wikipedia.org/wiki/Concorde_TSP_Solver)

[17] University of Waterloo (2023). Optimal 85,900-Point Tour,
(http://www.math.uwaterloo.ca/tsp/pla85900/index.html)

[18] Qiskit (2021). Examples for MaxCut and TSP,
(https://qiskit.org/documentation/stable/0.26/tutorials/optimization/6_examples_max_cut_and_tsp.html#Traveling-Salesman-Problem)

[19] Qiskit (2023).
(https://qiskit.org/)

[20] Bence Bakó, Adam Glos, Özlem Salehi and Zoltán Zimboràs (2023). Optimal QAOA design for the Traveling Salesman Problem.

[21] Andreas Bärtschi, Stephan Eidenbenz (2020). Grover Mixers for QAOA: Shifting Complexity from Mixer Design to State Preparation,
(https://arxiv.org/pdf/2006.00354.pdf)

[22] Qiskit (2023). CDKMRippleCarryAdder,
(https://qiskit.org/documentation/stubs/qiskit.circuit.library.CDKMRippleCarryAdder.html)

[23] Wikipedia (2023). Problem des Handlungsreisenden,
(https://de.wikipedia.org/wiki/Problem_des_Handlungsreisenden)

[24] Wikipedia (2023). Algorithmus von Christofides,
(https://de.wikipedia.org/wiki/Algorithmus_von_Christofides)

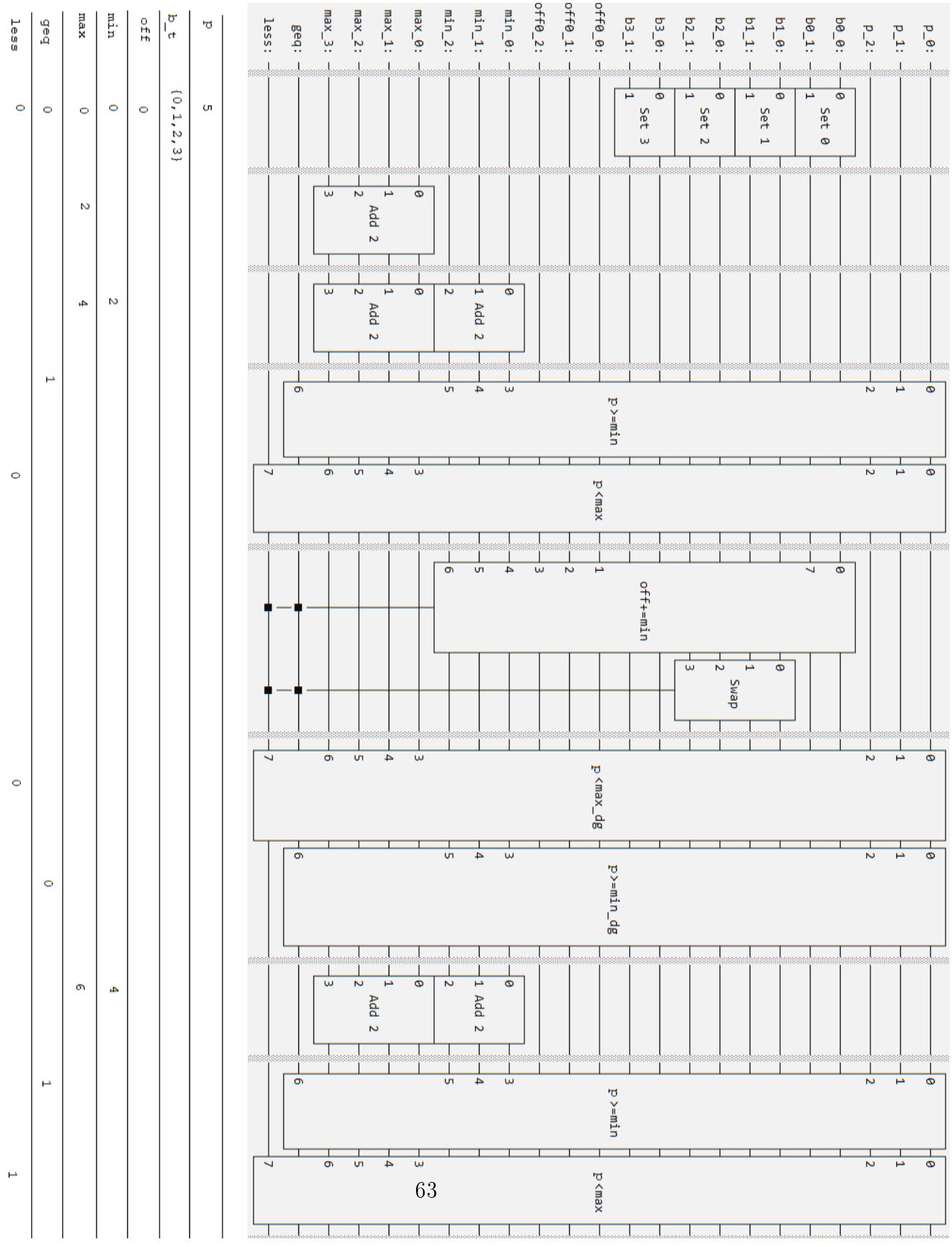
[25] Virginia Commonwealth University (2023). Illustration of the Nearest Neighbour Algorithm,
(https://www.people.vcu.edu/~gasmerom/MAT131/repnearest.html)

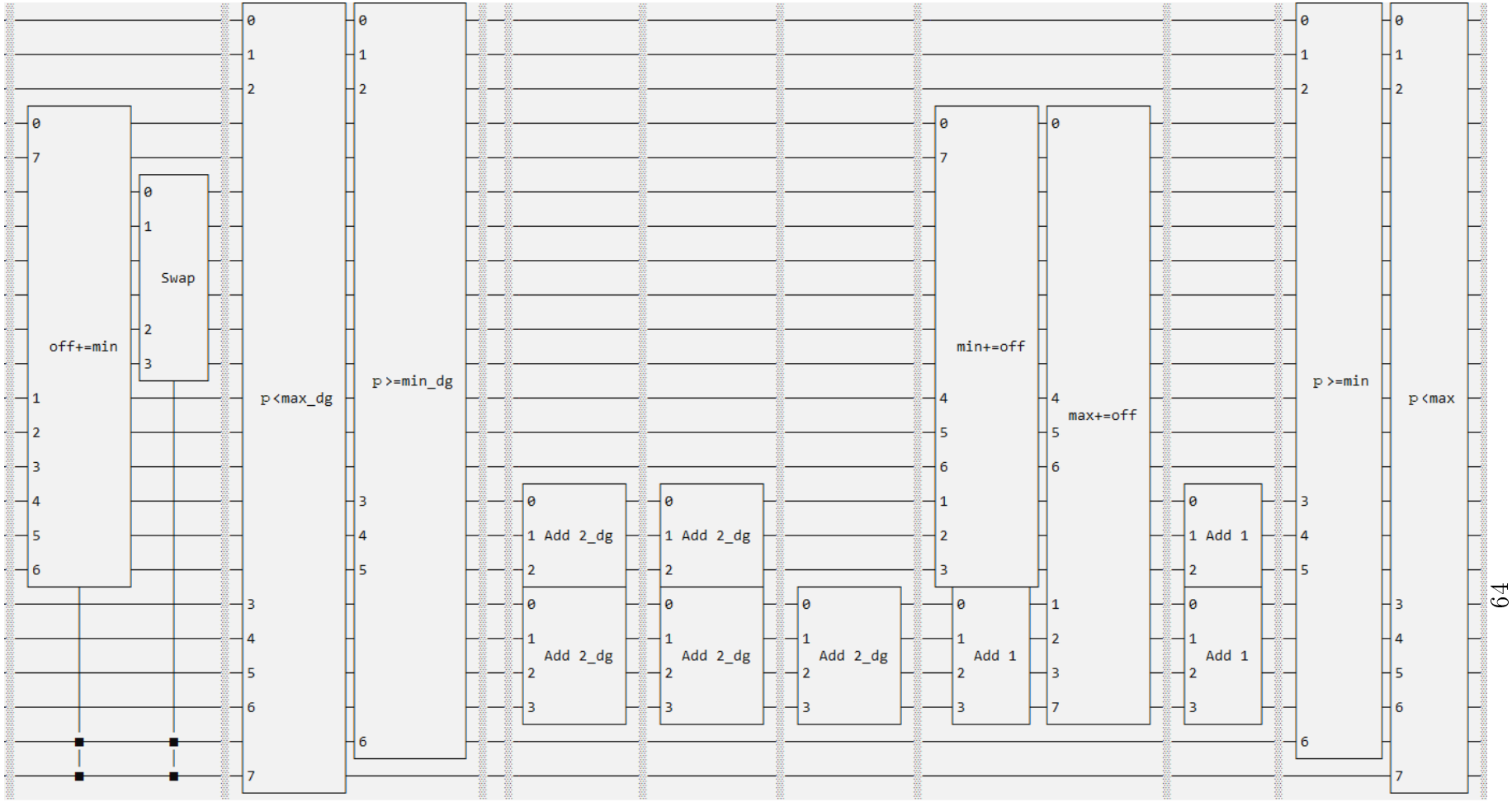
[26] Wikipedia (2023). Linear Programming Relaxation,
(https://en.wikipedia.org/wiki/Linear_programming_relaxation)

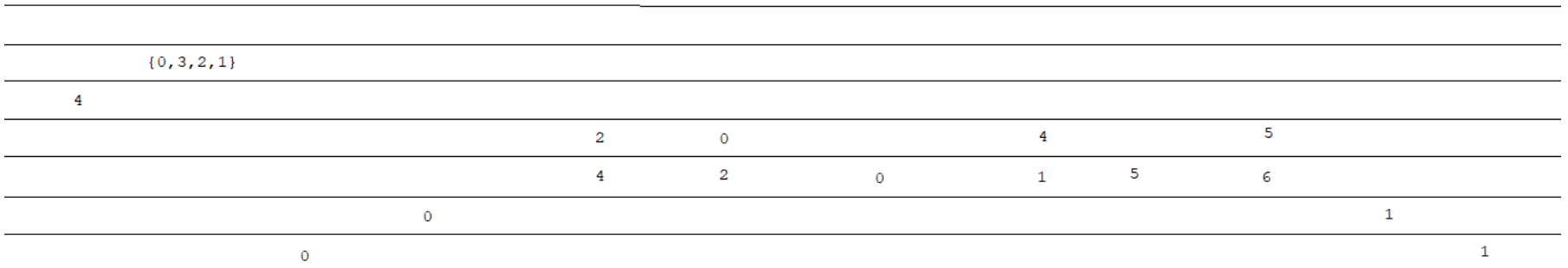
[27] Tadeusz Sawik (2016). A note on the Miller-Tucker-Zemlin model for the asymmetric traveling salesman problem,
(https://www.researchgate.net/publication/308707033_A_note_on_the_Miller-Tucker-Zemlin_model_for_the_asymmetric_traveling_salesman_problem)

6 Appendix

6.1 Circuit of the permutation index transformation for $n = 4$




off+=min
Swap
p<max_dg
p>=min_dg
Add 2_dg
Add 2_dg
Add 2_dg
min+=off
max+=off
Add 1
Add 1
p>=min
p<max


{0,3,2,1}
4
2
0
4
5
4
2
0
1
5
6
0
1
0
1

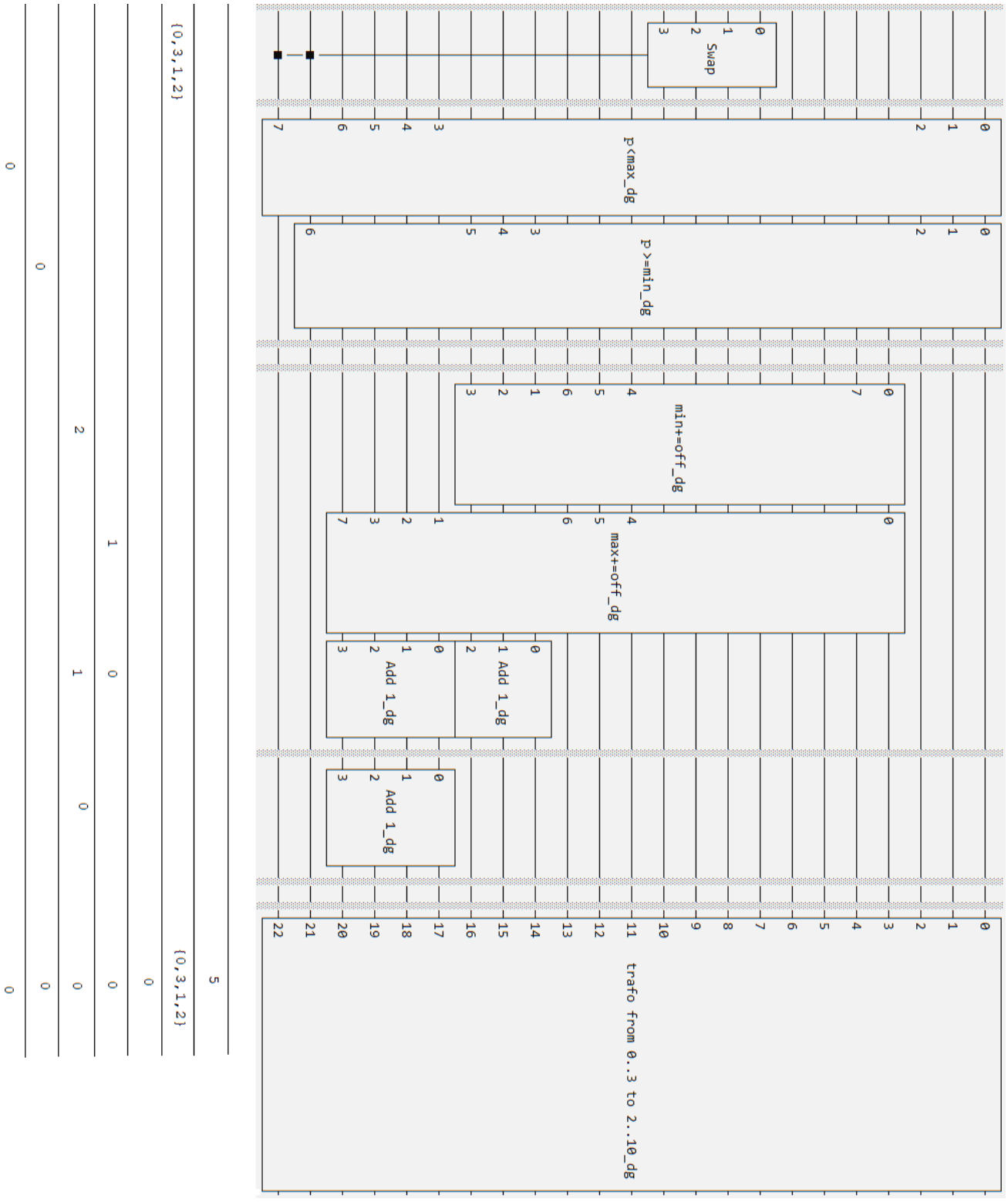


Figure 41: Circuit of the permutation index transformation with $|p\rangle$ as the permutation index and $|b_t\rangle$ being the output registers for $n = 4$. "_dg" stands for the inverse gate. Beneath the Circuit, one can find the integer values stored by the registers and their updates.