Hochschule Flensburg

M A S T E R - T H E S I S

Thema: Quantum-Hybrid Evolutionary Algorithm for Combinatorial Optimization

von: Brendan Alexander Meins

Matrikel-Nr.:	650437
Studiengang:	Angewandte Informatik
Betreuer/in und Erstbewerter/in:	Prof. Dr. Jan Christiansen
Zweitbewerter/in:	Dr. Sebastian Hans Peter Rubbert
Ausgabedatum:	03.04.2023
Abgabedatum:	01.09.2023

Schriftliche Versicherung

Ich versichere, dass ich die vorliegende Thesis ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Quellen benutzt habe.

Brendan Alexander Meins

Danksagung

Zunächst möchte ich mich bei Jan Christiansen bedanken, der die Rolle als Erstprüfer und Betreuer der Thesis übernommen hat.
Ein weiterer Dank gilt Sebastian Rubbert, der die Aufgabe als Zweitprüfer meiner Thesis wahrgenommen und die Thesis ebenfalls betreut hat.
Danke, dass Sie beide jederzeit bei Fragen und Problemen zur Stelle waren!

Zuletzt möchte ich mich bei meinen Geschwistern und meinen Eltern für ihre Unterstützung und Geduld bedanken. Danke!

Abstract

Due to the limitations of current Quantum Computers, Quantum Computers are inapplicable to problems in combinatorial optimization at the scale of real-world problem instances. Instead of replacing algorithms that run on Classic Computers, current research also includes accelerating classic algorithms to leverage the potential speed-up of Quantum Computing. In this thesis, combinatorial optimization and Quantum Computing will be introduced, and some challenges regarding these topics will be elaborated. With these problems at hand, an Evolutionary Algorithm will be developed to optimize a Quadratic Unconstrained Binary Optimization problem and extended by Quantum Optimization. For this, a variety of genetic operators will be compared to create an Evolutionary Algorithm using the Traveling Salesman Problem as a benchmark. This Evolutionary Algorithm will be extended to a Memetic Algorithm with different local optimization techniques that are assumed to be efficient on a Classic Computer and on a Quantum Computer, respectively. After evaluating different variants of the latter, the results of the Memetic Algorithm with a classic local optimization and the quantum local optimization will be analyzed and compared. It will be shown that the quantum local optimization can indeed improve the Evolutionary Algorithm heuristic to solve Quadratic Unconstrained Binary Optimization problems. Finally, the resulting algorithm will be evaluated against Tabu Search and Simulated Annealing, showing that the quality of results produced by the developed algorithm is competitive relative to other heuristics.

List of Figures

List of Tables

List of Acronyms

TSP Traveling Salesman Problem

EA Evolutionary Algorithms

GA Genetic Algorithms

GP Genetic Programming

ES Evolutionary Strategies

EP Evolutionary Programming

MA Memetic Algorithm

QHEA Quantum-Hybrid Evolutionary Algorithm

QUBO Quadratic Unconstrained Binary Optimization

QAOA Quantum Approximate Optimization Algorithm

MILP Mixed-Integer Linear Programming

MIQP Mixed-Integer Quadratic Programming

MIP Mixed-Integer Programming

QP Quadratic Programming

DHM/ILC Dynamic Decreasing of high mutation ratio/dynamic increasing of low crossover ratio

ILM/DHC Dynamic Increasing of low mutation ratio/dynamic decreasing of high crossover ratio
.

LP Linear Programming

IP Integer Programming

VRP Vehicle Routing Problem

NISQ noisy intermediate-scale quantum

Contents

1 Introduction

Quantum Computing is a promising research area in physics and computer science as Quantum Computers have the potential to outperform classical computers for certain problems. Companies like IBM and D-Wave already developed their own Quantum Computers. While the capacity of current universal Quantum Hardware is rather limited and not sufficient for real-world problems, the capacities of Quantum Annealers are sufficient for first applications. To leverage the benefits of Quantum Hardware in the current state of development when facing actual problem instances, classic algorithms can be accelerated by shifting the computation of derived sub-problems to Quantum Computers. This thesis will make a suggestion for a combined algorithm that makes use of the advantages of today's Quantum Hardware by searching a large solution space of a problem on a Classic Computer and separately solving smaller problems on a Quantum Annealer. The focus will be on problems in the domain of combinatorial optimization.

The first chapter will outline the context of the proposed algorithm and introduce the necessary fundamentals. Important concepts, of course, will be examined extensively. However, I assume that the reader has a basic understanding of linear algebra, such as vectors and matrices, and also a basic knowledge of graph theory.

I first give a brief overview of the combinatorial optimization landscape and the associated challenges in the section 1.1. I will also introduce the Traveling Salesman Problem (TSP), which will be the problem used to analyze and benchmark the algorithm.
Secondly, Evolutionary Algorithms (EA) will be introduced in section 1.2. After an introduction to general aspects of this meta-heuristic, I will dive deeper into the topic by bringing in an advanced concept called Memetic Algorithm (MA).
Thirdly, I will address Quantum Computing, particularly the state of development and applications in optimization.
The rest of the thesis will focus on the suggested algorithm by first introducing its components and explaining them concerning the developing process. I will also analyze and discuss the results.
Finally, I will draw conclusions, address remaining challenges, and make suggestions for further research.

1.1 Combinatorial Optimization

Many problems in industry and economics are optimization problems that ask to minimize or maximize a given process regarding duration, costs, or profit [9]. Optimization problems usually depend on some input variables. The problem formulated as an objective function f, optimization seeks to find the variables of f, called decision variables, for which f is minimal or maximal. Moreover, f can be subject to one or more constraints, such as budget or other resource limitations. If a solution violates these constraints, the solution is infeasible, while a solution where no constraints are violated is called feasible. Optimization has several sub-fields that are concerned with different problem types. The sub-field called combinatorial optimization is concerned with finding the optimal solutions to problems where the set of feasible solutions is finite and either discrete or can be reduced to a discrete set [34]. An example of a combinatorial optimization problem are routing problems. Routing problems aim to find the optimal path within a graph depending on some conditions. A well-studied example is the TSP [39] [21], where the shortest path between all vertices of the graph is sought, such that every vertex of the graph is visited exactly once. In other terms, the shortest path within a set of countable, feasible paths. This is just one example among other problems in routing, such as the Vehicle Routing Problem (VRP) and more.

In optimization, there are various ways of modeling problems. In general, a distinction is made between discrete and continuous optimization. Combinatorial optimization is, as mentioned, a sub-field concerned with finite and discrete solution spaces. Problems are typically formulated as problems of Integer Programming (IP), where one or more decision variables can be integer. Problems that have both continuous and discrete decision variables are called problems of Mixed-Integer Programming (MIP), which are also common in this domain of optimization. [34] [6] [24].

Another distinction in optimization is made between convex and non-convex optimization problems. The intuitive description of a convex problem, e.g. f, is when all feasible solutions of f are above f, that is any two points of the solution space can be connected by a straight line without leaving the region of feasible solutions. This is favorable for the development of algorithms to solve convex problems [6]. Objective functions encountered in combinatorial optimization are often non-convex due to the characteristics of the discrete variables involved, which can make these functions discontinuous. Generally, problems of IP and MIP are therefore harder to solve than problems that involve continuous decision variables only [6].

When the objective function f is a linear function and the constraints are linear, f is a Linear Programming (LP) problem. If f is quadratic, f is a problem of Quadratic Programming (QP). Problems of LP and QP can have decision variables that are continuous, integer variables, or both. For both applies the statement from the paragraph above, that discrete decision variables complicate the problem and problems of Mixed-Integer Linear Programming (MILP) and Mixed-Integer Quadratic Programming (MIQP) are

harder to solve than of LP and QP, respectively. In combinatorial optimization, objective functions are often one of the two, linear or quadratic.

An objective function can be reformulated, such that the constraints are part of the function. For this, constraints are incorporated into the objective function as penalty terms. These terms penalize constraint violations and reward compliance, respectively. First, consider the following standard LP problem:

$$\begin{aligned} max\ f(x) =& c^T x \\ s.t. \quad & Ax \leq b \end{aligned} \tag{1}$$

where $c \in \mathbb{R}^n$, $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$. Here, x_i, $i = 1, ..., n$ are the decision variables. c is the so called cost vector and c_i are the costs or rewards for x_i. A is a matrix where the rows a_j, $j = 1, ..., m$ and the scalar b_j constraint x to the inequality $a_j x \leq b_j$. That is, the inequality of $Ax \leq b$ is meant component-wise, I refer to [6] for a more detailed description. To add the inequality constraints to the objective function, first note that the inequality constraint means that Ax must not be larger than b for all components. If some $a_j x$ are larger than b_j, that means that $b_j - a_j x < 0$. This violation can be penalized by a non-negative scalar value λ_j, hence $\lambda_j(b_j - a_j x)$. This term added to the objective function will decrease the value of the objective function if the constraint $a_j x \leq b_j$ is violated. Given a vector λ with scalar values λ_j for each constraint $a_j x \leq b_j$, the task is now:

$$max\ f(x) = c^T x + \lambda^T (b - Ax) \tag{2}$$

This example of adding constraints to the objective function is called Lagrangian Relaxation [32], λ_j are called Lagrange Multiplier. It is worth mentioning that the choices for λ_j are not trivial, however not the subject of this thesis.

A special case where the constraints of a problem are added to the objective function is the Quadratic Unconstrained Binary Optimization (QUBO) model. The standard representation [17] is:

$$min\ f(x) = \sum_{i=1}^{n} a_i x_i + \sum_{j>i}^{n} b_{ij,} x_i x_j \tag{3}$$

where x is a binary vector of decision variables of length n. The model can have linear terms which are represented by the first sum, the quadratic terms are represented by the second sum. A common representation of QUBO is the representation as a matrix:

$$min\ f(x) = x^T Q x = \sum_{i=1}^{n} \sum_{j=i}^{n} Q_{ij} x_i x_j \tag{4}$$

where x is a binary vector of decision variables of length n and Q is a matrix of dimension $n \times n$ containing all problem data [10]. Since the decision variables are binary, $x_i^2 = x_i$.

The linear terms are therefore located on the main diagonal of the matrix. As the authors explain in [10], many constrained problems in combinatorial optimization can be re-formulated as a QUBO by using quadratic penalties instead of explicit constraints. A special benefit of this model is its relation to the Ising model, which will be introduced in the section 1.4. In [23], TSP is formulated for n cities as a problem with n^2 variables. The distance calculation function is:

$$\sum_{i=0}^{n+1}\sum_{j=0}^{n+1}\sum_{t=0}^{n} d_{i,j}x_{i,t}x_{j,t+1} \tag{5}$$

In this model, d_{ij} denotes the distance from node i to node j and $x_{i,t}$ and $x_{j,t}$ is a binary decision variables that is

$$x_{i/j,t} = \begin{cases} 1 & \text{if node } i \text{ is visited at time } t \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

The authors do not reference the original model and leave out the constraints. Anyways, every node must be visited at exactly one point in time, and exactly one node must be visited at any point in time. That is

$$\begin{aligned} s.t. \quad & \sum_{i=0}^{n} x_{i,t} = 1,\ t = 1, ..., n \\ & \sum_{t=0}^{n} x_{i,t} = 1,\ i = 1, ..., n \end{aligned} \tag{7}$$

Reformulating this constraint as quadratic penalty terms they become:

$$\begin{aligned} & P(x_{0,t}+, ..., +x_{n,t} - 1)^2,\ t = 1, ..., n \\ & P(x_{i,0}+, ..., +x_{i,n} - 1)^2,\ i = 1, ..., n \end{aligned} \tag{8}$$

Here, P is the penalty analog to the Lagrangian multiplier, as the authors use in [10]. The quadratic unconstrained binary optimization problem is now:

$$\begin{aligned} & \sum_{i=0}^{n+1}\sum_{j=0}^{n+1}\sum_{t=0}^{n} d_{i,j}x_{i,t}x_{j,t+1} + P(x_{0,t}+, ..., +x_{n,t} - 1)^2 + P(x_{i,0}+, ..., +x_{i,n} - 1)^2 \\ & i = 1, ..., n\ t = 1, ..., n \end{aligned} \tag{9}$$

Briefly introduced, to transform this into the matrix representation, the indexes $x_{i,j}$ are summarised, for instance, $x_{2,1}$ becomes x_4. Consider a TSP with $n = 3$. The optimization

problem is:

$$\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{2,1} & x_{2,2} & x_{2,3} & x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix} \cdot Q \cdot \begin{bmatrix} x_{1,1} \\ x_{1,2} \\ x_{1,3} \\ x_{2,1} \\ x_{2,2} \\ x_{2,3} \\ x_{3,1} \\ x_{3,2} \\ x_{3,3} \end{bmatrix} \tag{10}$$

The authors of [10] provide a comprehensive tutorial on how to transform any objective function into a matrix.

The challenging property of many problems in combinatorial optimization, regardless of the representation, is that the set of feasible solutions grows significantly when increasing the problem size only by a little [29]. For example, for a symmetric TSP instance with n vertices, there are $\frac{(n-1)!}{2}$ possible feasible solutions [28], that is for $n = 5$ there are 12 possible solutions, 60 solutions with $n = 6$, 181440 solutions for $n = 10$ and 60822550000000000 solutions in a TSP with 20 nodes. Symmetric means that the graph is undirected. Traversing the edge from node i to node j is the same as traversing the edge from node j to node i. There exists no algorithm to solve this problem efficiently. In the theory of computational complexity, an algorithm is called efficient if it solves the problem deterministic in polynomial time. In that case, this problem lies within the complexity class of P. Problems for which only non-deterministic polynomial time algorithms exist to solve this problem are in the complexity class NP. There is a set of problems called NP-Hard. Every problem in NP can be reduced to every problem that is NP-Hard. If the problem is both NP and NP-Hard, then it is called NP-complete. That is an algorithm solving one NP-Hard problem efficiently solves all problems in NP too. Such an algorithm does not exist and many problems in combinatorial optimizations are problems of these complexity classes.

Instead of computationally expensive algorithms, algorithms are used that only approximate the optimal solution. These algorithms often make no or just little assumptions about the problem to solve and are called heuristics or meta-heuristics, which can provide satisfactory results. The distinction is that meta-heuristics are more generic.

An example of heuristics is Tabu Search, where the algorithm searches the solution landscape while maintaining a list of solutions that are "tabu". The criterion that determines whether or not a solution is added to the list depends on the tabu strategy, same applies to the stopping criterion.
An Evolutionary Algorithm is an example of a meta-heuristic in optimization and will be introduced in the next chapter.

1.2 Evolutionary Algorithms

EAs are meta-heuristics for solving optimization problems [31] and are generally stochastic search methods [4]. Inspired by natural evolution, EAs mimic the ability of a population of individuals to adapt to their environment and become stronger over consecutive generations [40] [4]. The population is a list of possible solutions to the optimization problem. In the context of EAs, these candidate solutions are called individuals, and the optimization problem is called the fitness function. The set of all possible solutions to the fitness function is called the fitness landscape, to which the population is to adapt. The solution quality of an individual, e.g. the result of the fitness function, is called the individual's fitness [40]. EAs make iterative changes to the population using observations made in nature, where individuals reproduce, mutate, and have to compete against each other, with their fitness as the primary decisive factor for success [40] [4].

EAs have several sub-fields, such as Genetic Algorithms (GA), Genetic Programming (GP), Evolutionary Strategies (ES) and Evolutionary Programming (EP) [4] [31]. However, the distinction is not too sharp as some algorithms of EAs can union aspects of several sub-fields. The main distinctions between the sub-fields are genetic representations, the problems they try to solve, and implementation details [31]. The most common type of EAs are GAs, where solutions are typically (but not necessarily) represented as a string of numbers. The whole string is frequently called the chromosome, a single number encoding information to the individual is called a gene, and a semantic group of genes is called a genome [40] [5] [25] [26]. Manipulation of individuals is made by so-called genetic operators that act on the individual's chromosomes and change or recombine their genes, respectively. As of now, GAs are used in a large variety of applications in operations research, such as scheduling, inventory control, and others [11]. Applications of EAs are also found in multimedia, for example, image processing and gaming [11]. This shows the versatility of the meta-heuristic in terms of adaptability to the problem that is to be solved, but also the robustness of GA. Current research also includes multi-objective optimization problems.

Some negative aspects of EAs have to be taken into account. A well-known problem is the premature convergence behavior that occurs according to [11] especially due to the loss of information during the evolution. That is, for example, every individual in the population has a binary gene set to 1, where 0 would encode information that is mandatory to find the optimal solution. In [11], which is focused on GAs, the authors figure that there should be a way to find the value of a gene by some local search. Generally, for EAs, this is an extension called Memetic Algorithm [4]. MAs bring in the idea of plasticity, which is an individual's ability to adapt to their surroundings and self-improve their fitness [4]. This idea tries to tackle the problem that EAs are well suited for global search, but have problems in finding solutions in the local environment of a previously found solution. EAs mimic this by improving an individual's fitness with local search heuristics, for example, Tabu Search, Hill Climbing, and other algorithms. For this, individuals are commonly selected arbitrarily [4].

1.3 Quantum Computing

In contrast to a Classic Computer that works with systems of bits that can be either in the state 0 or 1, Quantum Computers use laws of quantum mechanics, superposition and entanglement, to work with quantum states [36]. The atomic unit of a Quantum Computer is a qubit. While the state of a classic bit can be read, a qubit must be measured. Before the measurement, the qubit is in superposition, which means that it is both, 0 and 1. Only by the measurement the state of a qubit is determined, and the superposition collapses to either 0 or 1. Qubits can be manipulated in a way such that the probability of whether the qubit is measured as 0 or 1 is changed. In quantum systems that are composed of systems with two or more qubits, entanglement can be used such that the measurement of one qubit affects the measurement of another qubit in that system [17]. Quantum Computers that manipulate the states of qubits with quantum gates are called universal Quantum Computers. Analog to the Classic Computer, research on universal Quantum Computers aims to create a Quantum Computer that is programmable.

Working with qubits instead of classic bits, Quantum Computers could potentially solve some problems faster than Classic Computers [17][36]. A promising application of a sufficiently large universal Quantum Computer is the factorization of prime numbers, using Shors Algorithm [22], which would have a strong impact on public key cryptography.

The current state of development is called the noisy intermediate-scale quantum (NISQ) era [35]. Quantum Hardware available today is not large enough to outperform Classic Computers and is prone to errors. Quantum error correction is a problem that is hard to solve. Classic Computers can copy a bit multiple times, which is not possible for qubits because of the No-Cloning-Theorem [17]. It is not possible to copy the state of a qubit to a second qubit without changing the source qubit. Currently, for quantum error correction, exponential many qubits are needed for error correction [17]. This is caused by the fact that larger quantum systems are more likely to interfere with their environment and doing so lose their quantum properties. The loss of these quantum properties is called quantum decoherence [17]. The scalability of Quantum Computers strongly depends on the ability to maintain low levels of decoherence and the development of robust error correction techniques, which is why preventing quantum decoherence and also the problem of quantum error correction is considered the most significant challenges, and is not yet clear if it is possible to overcome them.

The largest circuit-based quantum processor is currently the IBM Osprey with 433 Qubits [33].

1.4 Quantum Optimization

One important application for Quantum Computers is optimization, as they potentially overcome the challenges introduced in the section 1.1.

A prominent example of a quantum algorithm is the Quantum Approximate Optimization Algorithm (QAOA) [2]. In QAOA, a quantum state $|\psi>$ is prepared depending on two parameters β and γ. The procedure is in [19] described as follows:

1. Prepare a quantum state with values for β and γ
2. Repeat until a convergence criterion is met:
   - Measure quantum state
   - Compute the expectation value
   - Find new values for the parameters β and γ using a classic algorithm
   - Replace β and γ with the new values

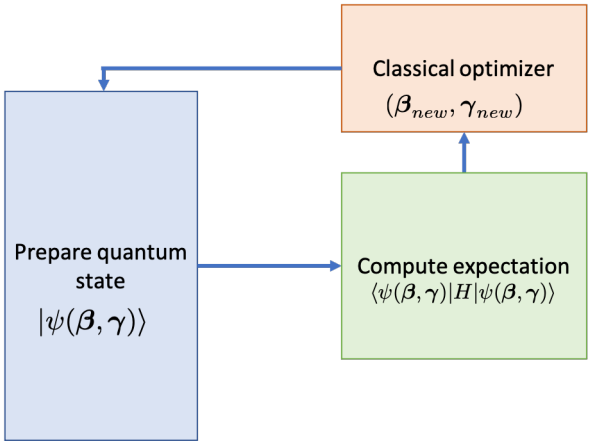


Figure 1: QAOA Schematic from [19]

The preparation of $|\psi(\beta, \gamma)>$ is done with a quantum circuit with gates that are problem-specific. I refer to [19] for a more comprehensive description. Note that QAOA only transforms an optimization problem into another optimization problem.

Some authors like [13] attempt to develop quantum algorithms that are inspired by their classic counterparts. For EAs, the authors of [13] try to develop a Quantum-Inspired Evolutionary Algorithm. However, they only use the quantum phenomena of superposition to represent multiple states but not entanglement. The result is an algorithm using the Quantum Computer as a random number generator, that does not leverage the full potential of Quantum Computing.

Besides Quantum Computers that implement gate-based circuits, there is another approach called adiabatic quantum computing [17]. The basis for this is the adiabatic theorem. The adiabatic theorem states that a quantum system in a low energy state will stay in a low energy state if it is changed sufficiently slowly [17]. The energy states of a quantum system are described by a Hamiltonian operator H. The first step is to translate an optimization problem into a Hamiltonian $H_{problem}$ such that the unknown minimum energy state encodes the solution to the problem. Next is to prepare a Hamiltonian H_{start} where the minimum energy state is known and a quantum system that is in the minimum energy state of H_{start}. Gradually changing the Hamiltonian H_{start} to the Hamiltonian $H_{problem}$ over time t:

$$H(t) = (1 - \frac{t}{T}) \cdot H_{start} + \frac{t}{T} \cdot H_{problem} \tag{11}$$

will end in a quantum system in the minimum energy state of $H_{problem}$, and the result can be obtained by measuring the quantum system. An important factor is that the process must be executed sufficiently slowly. Changing from H_{start} to $H_{problem}$ too fast can result in a quantum state that is not in the minimum energy state of $H_{problem}$. Any quantum circuit can be transformed into an adiabatic algorithm.

Quantum Annealing is a heuristic that can be explained by the manufacturing of metal [8]. In the cooling process metal eventually ends up in a state with a certain arrangement of atoms. Every arrangement corresponds to energy, and low-energy arrangements often have desirable properties [17]. While the metal cools down, the atoms arrange and the probability that the atoms temporally rearrange into an arrangement of higher energy is higher with higher temperatures. By this, the atoms can overcome arrangements that are local minima. This idea is used in a heuristic called Simulated Annealing [17].
In Quantum Annealing, a magnetic field is the analog to the temperature and controls the probability of quantum tunneling, which is used to tunnel through a potential energy barrier. Rather than "climbing" up energy levels, this tunneling is the quantum effect that is said to potentially cause a speed up[8][17].

To solve optimization problems with Quantum Annealing, the problem has to be in a suited representation, which is typically the Ising model.

$$h(s) = \sum_{i=1}^{n} h_i s_i + \sum_{j>i}^{n} J_{i,j} s_i s_j \tag{12}$$

with $s_i \in \{-1, +1\}$ [17]. The Quantum Annealing process is now an adiabatic process that slowly increases the effects of h_i and $J_{i,j}$, until the quantum system is described by $H_{problem}$. It is not possible to determine whether or not the quantum system is in the minimal energy state of $H_{problem}$. Quantum Annealing has no optimal guarantee. Also, as the authors state in [8], Quantum Annealing is known to be well suited where the solution landscape has many local optima with high but thin barriers and thus might not be an advantage to arbitrarily chosen problems.

The QUBO model can be transferred into an Ising model with $x_i \mapsto \frac{s_i+1}{2}$ and the Ising model can be transferred into a QUBO model with $s_i \mapsto 2x_i - 1$ [17].

The largest Quantum Annealing processor is currently the D-Wave Advantage with 5760 qubits [33].

1.5 Objective of this Thesis

After introducing the fundamentals of this thesis, some weaknesses and problems occur when facing problem instances in combinatorial optimization.

The first problem is that most combinatorial optimization problems are hard to solve. Secondly, heuristics to find a solution already exist, with EA as a meta-heuristic that is known to be robust and good at finding solutions in the global search space. However, EAs tend to premature convergence and are bad at finding solutions that are in the neighborhood of a current solution, that is, bad at local optimization.
Quantum Computers today have severe hardware limitations and are not suitable for optimization problems on the scale of real-world problem instances, but they offer promising advantages.

Some authors try to accelerate current algorithms with quantum computing instead of replacing algorithms completely, for example, [18]. The article is about polynomials over finite fields and not related to optimization but represents the general idea of leveraging quantum computing in the current state of development.

Albeit the current limitations, Quantum Annealing processors are large enough for small problem instances, with QUBO being the de-facto standard representation of combinatorial optimization problems. .

QUBO as a standard representation for optimization with the Quantum Annealing heuristic brings in the challenge that not all problems in optimization are naturally 0 / 1 decisions. Also, the algorithm can not use the constraints of the problem to restrict the search to feasible areas in the solution space, since all constraints are added to the objective function, which can generally lead to significantly larger solution spaces. Ultimately, the QUBO model representing a real-world problem is too large for a Quantum Annealer.

Before outlining the objective of this thesis more thoroughly, I make a few assumptions to begin with.

According to the authors of [8] and [17], Quantum Annealing is expected to be better and faster than Simulated Annealing, although the conditions to achieve these properties are not extensively studied.

Assumption 1

Quantum Annealing is at least as good as Simulated Annealing.

Good in terms of both, computation time and result quality. Against the background of this assumption, Simulated Annealing acts as a placeholder algorithm in this thesis. Positive observations therefore necessarily also hold for Quantum Annealing, while negative observations do not necessarily apply.

As an example, D-Wave offers a cloud solution to access their Quantum Annealer [7]. This communication generates an overhead compared to a situation where the algorithm has direct access to a Quantum Annealer. Also, the optimization tasks are enqueued with other tasks such that the queue position is also an important factor. Replacing the Simulated Annealing algorithm with the cloud solution, the result could be distorted because of this overhead.

Assumption 2

If Quantum Annealing is used, there is a negligible overhead.

For example, with direct access to a Quantum Annealer or close to 0 latency and no enqueue time.

A Classic Computer can optimize a single binary decision variable precisely by verifying if changing the value of this decision variable is an improvement. The more variables the Classic Computer has to consider in one optimization step, the harder the problem becomes.
Recalling that Quantum Annealing is a heuristic with no optimal guarantees, but is expected to be a strong heuristic, I assume that

Assumption 3

Quantum Annealing outperforms a Classic Computer with a growing number of decision variables.

I also refer to the authors of [8], that the quantum speed up is expected to increase with a growing number of decision variables. Later, I will refer to the optimization of one decision variable at a time as Classic Local Optimization because a Classic Computer can optimize one variable efficiently but has problems as the number of decision variables grows, and with Quantum Local Optimization I will refer to the optimization of multiple decision variables because of assumption 3.

The objective of this thesis is to elaborate an EA to optimize a QUBO, as QUBO is the de-facto standard of problem representation for quantum optimization, but quantum hardware is not sufficiently large for most applications. Now that EAs are bad in terms of locality and the concept of MAs aims to tackle this by including local optimizations, the

question is whether or not a Quantum Annealer can be used for this local optimization and if it is an improvement.

Note that it is not the goal of this thesis to create an algorithm that can compete with TSP specific algorithms. TSP is only used as a benchmark because it is a hard problem that is well-studied.

The algorithm developed belongs to the sub-field of GA. Now that both, GA and MA are sub-fields of EA, I will from now on use the term EA when referring to the GA without local optimization, "EA with Classic Local Optimization" when referring to the GA with a Classic Local Optimization and "EA with Quantum Local Optimization" or Quantum-Hybrid Evolutionary Algorithm (QHEA) when referring to the GA with Quantum Local Optimization.

Finally, the resulting QHEA will be analyzed and also compared with other QUBO solvers and I will draw conclusions regarding the questions asked.

2 Quantum-Hybrid Evolutionary Algorithm

In this section, I present the results of my research process regarding the genetic operators used in the final result. For each type of genetic operator, that is selection, crossover, and mutation, I explain the different variations that are considered, why they are considered, and how they perform in comparison. I measure the performance of the algorithm depending on the convergence behavior, and the quality of the result on the one side and also the generations needed to receive that result are the indicators of interest.

For better comparability of an operator's effect on the evolution, a baseline algorithm is used, with simple operators for each type of operator. During the experiments regarding a specific operator type, the base algorithm remains the same except for the operator type in question.

The base algorithm is composed of three different types of genetic operators: Selection, crossover, and mutation. The selection operator is responsible for selecting individuals for reproduction. The base algorithm implements a selection operator that starts by selecting the strongest and second strongest individual and continues until $n/2$ individuals of n individuals have been selected. This method for the selection operator is called elitist selection and will be introduced in section 2.1.1, among other methods. I chose this method because it does not depend on randomness.

The crossover operator recombines two individuals into new individuals, that is, the crossover operator is the reproduction step. The base algorithm implements a crossover where the first child inherits the first half of the genes of the first parent and the second half of the genes from the second parent. The second child inherits its genes vice versa. This crossover method is called 1-Point Crossover and is a variant of the k-point crossover, that will be introduced in section 2.1.2. I chose this crossover for the base algorithm because again, this crossover does not implement any randomness.

The mutation operator is responsible for the random mutations of genes, that is, the mutation operator is used to allow the algorithm to explore the solution space, such that genes that are not present in the population can occur randomly. I choose a probability of 5% for each gene in the chromosome because of findings in the literature, where 5% is frequently used [5] [40]. The effect of different mutation rates are compared in section 2.1.3.

The authors of [20] propose different strategies for replacing individuals, where for instance an individual that is similar to a parent will likely replace it. The replacement strategy in this thesis is the same for all experiments. After creation, the new individuals are appended to the population. Because the objective is to minimize a function, the population is sorted ascending by the individual's fitness and then truncated to the original size at the end of every generation. The only exception is the elitist selection which is explained in the corresponding section.

The starting algorithm does not implement any operators for local optimization at the beginning. The algorithm maintains a population of 100 individuals and the evolution takes 1000 generations. Note that these values for population size and number of generations are chosen arbitrarily and are just a guess for suited values based on experience. To have a large diversity in the population, all individuals are initialized randomly.

The problem used to benchmark is, as mentioned in the introduction, TSP in the QUBO formulation. The QUBO formulation is not ideal for TSP since in the formulation used, the solution space grows to 2^{n^2} from $\frac{(n-1)!}{2}$ in the TSP formulation which is based on permutations. The reason for this is that the permutation-based representation restricts the solution space to valid Hamilton cycles, while the QUBO representation does not. The constraints are part of the cost function as penalty terms in the latter. That is, aside from the actual goal to minimize the objective, the algorithm also faces huge barriers to getting from one valid solution to another valid solution. Also, since the QUBO representation also does not work on a graph representation, some genetic operators that are TSP specific like Edge Assembly Crossover are not used, even though they are considered to be best for this problem. The Edge Assembly Crossover generates a graph from two-parent graphs by alternately selecting an edge from each parent [27].

After validating methods for different types of genetic operators independently from operators of other types, the best methods for every operator type are combined to further improve the algorithm and finally assembled into an EA, which aligns with the available literature on classic EA. This EA will then be expanded with classic and Quantum Local Optimization.

2.1 Evolutionary Algorithm

The objective of this section is to establish the Evolutionary Algorithm and extend it with a Classic Local Optimization. The different types of genetic operators will be introduced. For each type of genetic operator, that is selection, crossover, and mutation, the general motivation and evolutionary context will be explained, followed by the introduction of different methods considered for each operator and the variants that were implemented. Finally, a set of compositions of the selection method, crossover method, and mutation method will be evaluated to find the best composition from this set. This composition will be the EA that is to be extended by the Classic Local Optimization.

2.1.1 Selection Operator

In a natural evolution, strong individuals prevail in a population and reproduce while weak individuals tend to extinct. However, a weak individual still has a chance to survive and reproduce due to environmental factors and coincidence. This is important for the population because weak individuals might have some rare genes that are mandatory

for the evolutionary success of a population. The versatility of genes distributed in the population is therefore another aspect to consider besides the survival of the fittest individuals and strongest genes. The key idea behind the selection operator is to mimic the "survival of the fittest" manners. It is responsible for selecting individuals to reproduce and populate their genes and replacing weaker ones in every generation of evolution.

A selection operator that only lets the strongest individuals survive and reproduce is called an elitist operator. The disadvantage is, as previously described using natural evolution, that some individuals may have a few strong genes not distributed in the population, but have a weak overall fitness. As an elitist selection only takes the strongest individuals, this can have negative impacts on the solution quality. That is, I compare the elitist selection method with methods that allow weaker individuals to reproduce. Allowing the weaker individuals to reproduce allows the algorithm to explore the solution space while restricting the algorithm to only consider strong individuals is called exploitation. The correct balancing of exploration and exploitation is therefore significant.

Generally, a pair of two parents is used to generate a pair of two offspring. All selection methods select a total number of $n/2$ individuals, that is $n/4$ pairs of parents, such that $n/2$ children are generated, where n is the total number of individuals in the population. Note that this implies that, for the algorithm developed in this thesis, I presuppose that $0 \equiv n \mod 4$ due to convenience.

Some methods in literature [5] [40] generate more than two children for a pair of parents or select up to all individuals and create a population of children that is as large as the original population. However, the primary objective of this section is to compare different selection methods in terms of the diversity of the population and their impact and the solution quality, which is why I chose the number of new individuals for all selection methods to be the same, such that the number of individuals produced do not impact the result.

Elitist Selection

From a population P of n individuals, this method selects the individuals $\{P_1, ..., P_{n/2}\}$ for recombination and then pairs the individuals (P_1, P_2) ... $(P_{n/2-1}, P_{n/2})$ to generate children. The second half of the population $\{P_{n/2+1}, ..., P_n\}$ are replaced by the new individuals. The reason why this method replaces individuals directly can be illustrated with an example: If the population has n individuals and the elitist method would append $n/2$ individuals to the population, the population has $\frac{3}{2}n$ individuals before the sorting and truncation. At this very moment, the 1st $n/2$ individuals may or may not be better than the new individuals in the population, that is, the 3rd $n/2$ individuals. However, the 2nd $n/2$ of the population are those individuals that were not selected by the selection operator. Independent of how good or bad the new individuals are, this half of the population will never be better than the 1st $n/2$ of the population and therefore

will never be selected by the elitist selection method to contribute to the evolution. The first $n/2$ individuals will be either the same individuals as before or a mix of the first $n/2$ individuals and the newly created offspring. This means that the 2nd $n/2$ individuals are redundant. Selection methods where individuals in the 2nd half of the population also have a chance to reproduce therefore do not replace them directly since individuals in the second half can be selected at some point in time in the future and are ultimately of significance.

No other variants of this selection method are considered. This is the selection method that the base algorithm implements.

Linear Ranking

This method was introduced by Baker [37]. The idea is that the individual's fitness itself is not the selection criterion but the ranking, e.g. position in the population. The higher the position the higher the probability for an individual to reproduce. At the end of the generation, the population is sorted ascending by fitness, that is, the first individual is the strongest and the last the weakest. The probability Pr for rank position R_i is given by the selection pressure sp through

$$Pr(R_i) = \frac{1}{n}(sp - (2sp - 2)\frac{i-1}{n-1}) \tag{13}$$

where $1 \leq i \leq n$, $1 \leq sp \leq 2$, with $Pr(R_i) \geq 0$, $\sum_{i=1}^{n} Pr(R_i) = 1$.
For instance, with $n = 10$ individuals and $sp = 2$ the selection probabilities are

$$\begin{bmatrix} 0.2 & 0.18 & 0.16 & 0.13 & 0.11 & 0.09 & 0.08 & 0.04 & 0.02 & 0.0 \end{bmatrix}$$

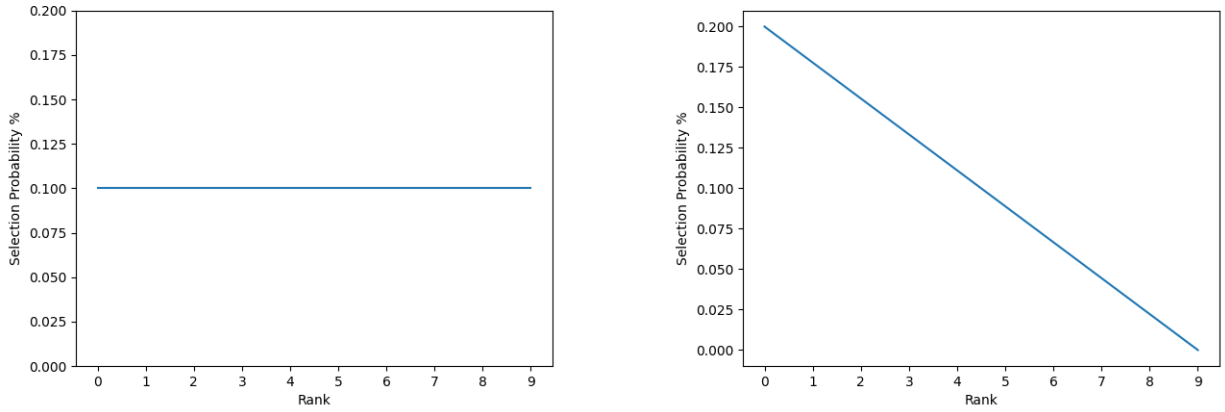
where the probability of 0.2 is the selection probability for the strongest individual. That is, the result is a linear (descending) function, which decreases stronger with increasing sp as displayed in Figure 2.

Recalling that $\sum_{i=1}^{n} Pr(R_i) = 1$, that means, for every execution, the operator selects one individual. The operator generates a uniform random number $r \in [0, 1)$ and, starting with the fittest individual, sums up the probabilities for each individual in the population until this sum is larger than r, and returns the index of that individual. This procedure is repeated $n/2$ times, such that $n/2$ individuals are selected for reproduction. This method does not exclude that an individual is selected multiple times.

Variants

The different variants of the selection method via a selection pressure are to examine the effects of different values for sp.

1. $sp = 1$



(a) $sp = 1$ (b) $sp = 2$

Figure 2: Selection probabilities for $sp = 1$ and $sp = 2$

With a selection probability of 1, every individual has an equal chance to be selected for reproduction. This is the same as uniformly random drawing the individuals from the population. For instance, for a population with $n = 10$ individuals, the probability for each individual is 10%, as shown by Figure 2a.

2. $sp = 1.1$

This will put very low pressure on the population. Even though the selection process now has a little bias towards the stronger individuals, this variant should be very similar to the variant above.

3. $sp = 1.6$

Setting the parameter $sp = 1.6$ will apply moderate pressure to the population. I expect that this parameter choice is already distinguishable from the random variant with a selection pressure of 1.

4. $sp = 2$

This is the highest value for the selection pressure. With this value for the parameter sp, the function in Figure 2b represents the selection probability by the rank of an individual in a population with 10 individuals.

5. *Increase linear*

I assume that increasing the selection pressure over time may allow the algorithm to

explore the solution space by allowing weaker individuals to distribute their genes in the population. Starting with a selection pressure $sp = 1$, the algorithm first randomly selects individuals from the population. With every generation, the pressure is increased until in the last generation, the selection pressure is at the highest value 2. The linear function is given by

$$sp = 1 + \frac{1}{totalGenerations} \cdot gen \tag{14}$$

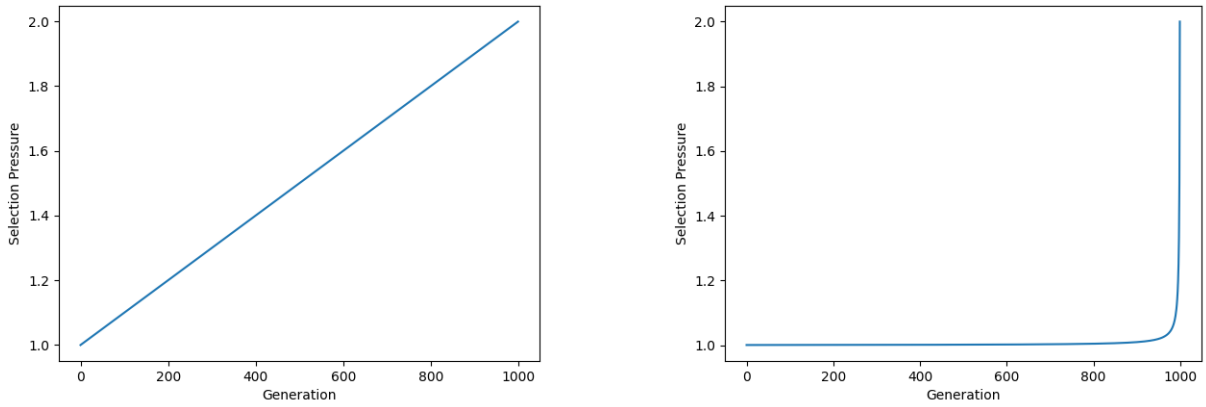
where $totalGenerations$ is the absolute number of generations the evolution takes, and gen is the current generation. This is shown in Figure 3a, where sp increases dependent on the generation. This moves the probability of selection from a uniform distribution towards the top of the population.

6. *Increase binomial*

Instead of linearly increasing sp, sp is close to a uniform distribution and increases sharply at the end of the evolution. The function is given by

$$sp = 1 + \frac{1}{totalGenerations - gen} \tag{15}$$

This is displayed in Figure 3b.



(a) Linear increase of selection pressure

(b) Binomial increase of selection pressure

Figure 3: Dynamic increase of selection pressure

Tournament Selection

The idea of this selection method is that individuals compete in a tournament against each other. For this, individuals are uniformly selected from the population. Two individuals are selected to compete against each other, with fitness being the factor determining the winner. The winner competes against the next individual who is also randomly drawn from the population, and so on. The tournament depth, by means of the number of competitions carried out, is set by a parameter k. The winner of the competition is the winner of the competition k and selected as a parent. This whole procedure is repeated $n/2$ times, once for every for every parent that is to be selected.

For recombination, that is crossover, two individuals have to be selected. For a pair of parents, this selection method is executed twice. The implementation restricts that one individual can not be selected as both parents. However, an individual can be selected multiple times in one generation.

Note that this selection method can be implemented in other, more complex ways than the procedure described above, some are described in [40] and [38].

Variants

As this selection method is controlled by the tournament parameter k, different variants of this method are different choices for the parameter k. I assume that suited values for k are strongly related to the total number of individuals, which is why I set the parameter in percentage to the population size.

1. $k = 20\%$

I chose the value for k because I expect it to be rather elitist since every individual has a chance of 20% to compete in the tournament. Even though this probability applies to weak individuals as well, they will lose the competition against, for instance, the strongest individual of the population, which also has a 20 percent chance of being drafted to the tournament.

2. $k = 5\%$

Note that, for a population size of 100, $k = 1\%$ would be a uniformly drawn sample from the population, which would be the same as the selection method that works with selection pressure with a pressure $sp = 1$. I choose a value of $k = 5\%$ because the uniform sample was already considered, however, I want the tournament selection method to include a variant that allows high diversity and thus more exploration.

Result

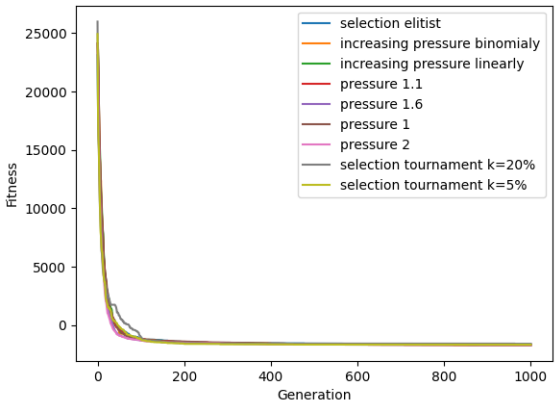
Figures 4 show the experiment results for a QUBO with 100 variables, that is, a TSP with 10 nodes. The fitness is on the y-axis and the generations are on the x-axis. Figure 4b shows the same graph as 4a, however, the y-axis is zoomed in such that the lines are distinguishable from one another.

Generally, as Figure 4a shows, all selection methods have similar convergence speeds and achieve a similar result quality. The slight differences are revealed by 4b, where the selection methods that allow less diversity generally converge earlier than the selection methods that allow more diversity in the population.

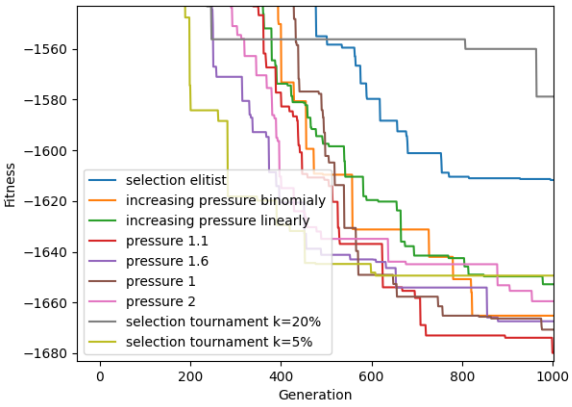
The elitist selection method as well as the tournament selection method overall scored the worse results in means of convergence speed and result quality.

The selection method that works with a parameter *sp* to control the pressure on the population performed better than the tournament selection and the elitist selection, with lower values for *sp* achieving the best result.

Increasing the selection pressure in any way did not achieve better results than choosing static values *sp*.



(a) Selection methods convergence behavior for TSP with 10 nodes



(b) Selection methods convergence behavior for TSP with 10 nodes, Zoomed

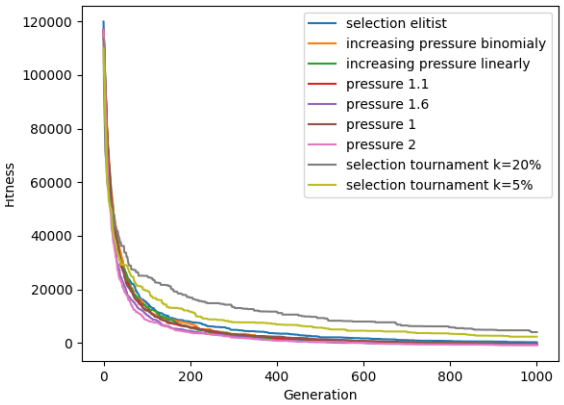
Figure 4: Selection methods for TSP with 10 nodes

Figure 5 shows the optimization results of a QUBO with 255 variables, representing a TSP with 15 nodes. The Figures again show the fitness development dependent on the generation.

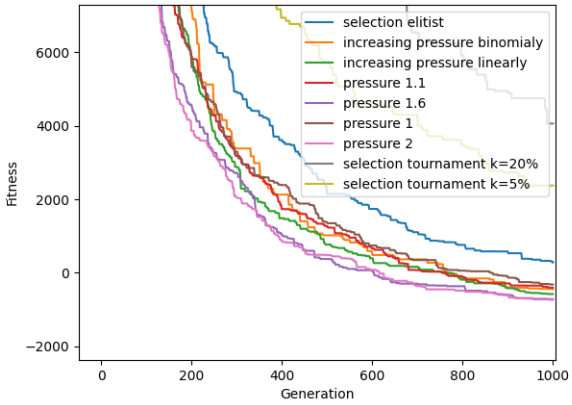
Increasing the problem size to 15 nodes and 255 variables underlines the previous results for the elitist selection method and the tournament selection.

The selection method by an individual's rank remains better than the elitist and tournament selection methods for all choices of parameter sp. However, the previous result where the selection by ranking is better with lower values for parameter sp is now invalid, as the values for $sp = 2$ and $sp = 1.6$ achieve better results than $p = 1$ and $sp = 1.1$.

Linearly adjusting the selection pressure achieved now better results than the static selection pressure $sp = 1$ and $sp = 1.1$, however is not better than $sp = 1.6$ and $sp = 2$.



(a) Selection methods convergence behavior for TSP with 15 nodes



(b) Selection methods convergence behavior for TSP with 15 nodes, Zoomed

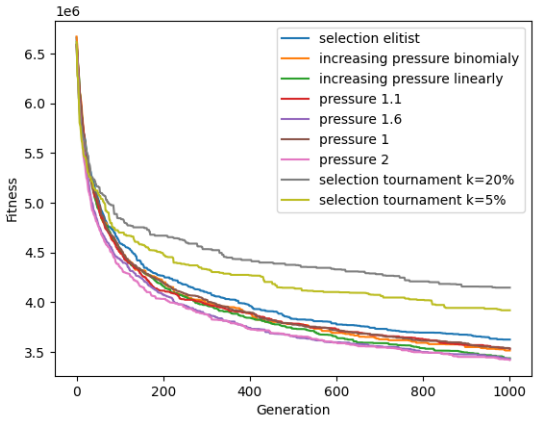
Figure 5: Selection methods for TSP with 15 nodes

Figure 6 shows the optimization results of a QUBO with 2500 variables.

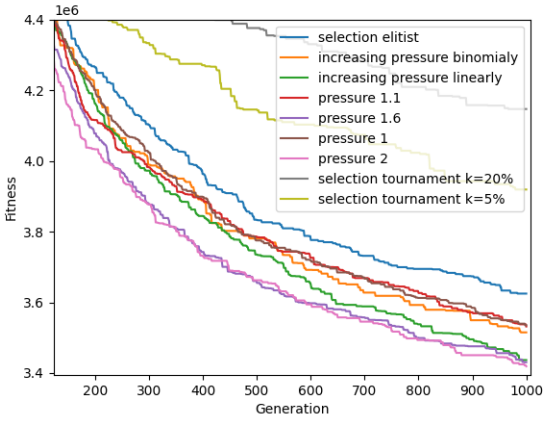
The elitist selection method as well as the tournament selection method are further shifted up towards worse solution qualities and early convergences, relative to the other selection methods.

The selection method with static $sp = 1$ and $sp = 1.1$ have a similar convergence as the variants that increase sp during the evolution.

Static values for $sp = 1.6$ and $sp = 2$ converge latest and achieve the best fitness values, with $sp = 2$ being slightly better than $sp = 1.6$. 6.



(a) Selection methods convergence behavior for TSP with 50 nodes



(b) Selection methods convergence behavior for TSP with 50 nodes, Zoomed

Figure 6: Selection methods for TSP with 50 nodes

The heuristic assumption derived from natural evolution, that the versatility of the genes represented in the population is important, is confirmed by the observations made in these experiments. The restriction is that meaningful versatility has some upper bound. From an algorithmic perspective, versatility allows the algorithm to explore the solution space and by that find promising areas of that space, but it is necessary to distinguish between "promising" and "not promising", where "promising" and "not promising" are stretchable concepts and need to be balanced. This is exactly what the selection pressure seems to do for the problem in question.

2.1.2 Crossover Operator

In the process of two individuals producing offspring, their genes are recombined into a new composition of genes. By chance, the new individual has a composition of genes that is stronger than its parents. Supported by the process of selection, this individual then has a good chance to populate its genes in subsequent generations. Also, in natural evolution, environmental factors might influence whether or not two individuals mate and produce offspring. This can be partly modeled by the selection process with the help of some probability of selecting weaker individuals. However, individuals may still not reproduce due to some coincidence. For example, because their offspring were prey to a predator.

The crossover operator mimics this evolutionary step of recombining genetic material from two individuals into new individuals. From the optimization perspective, the idea is to combine two solutions with the heuristic assumption that good but not optimal

solutions still have values for some decision variables that are ideal in means of the global optimum, and potentially recombine in a strong solution. The exact procedure of how genes are recombined is not trivial. Hence, different methods are evaluated in this section. Besides different methods for a crossover, the heuristic assumption that individuals might not reproduce even though they survived natural selection is modeled by the crossover operator with a probability, the crossover rate.

Crossover Rate

The crossover rate determines the probability that two parents that have been selected recombine into a new offspring. With the given probability, the crossover operator either recombines a given pair of individuals or omits this step, proceeding with the next pair of selected individuals.

Variants

1. 100%

A crossover rate of 100% is the same as not implementing a rate at all, as every pair of parents recombine at this rate. I consider this rate and use it as a reference to determine if lowering the crossover rate gives meaningful results, that is if the crossover rate has to be changed at all. The base algorithm omits no crossovers which is the same as a crossover rate of 100%.

2. 75%

This rate will omit $\frac{1}{4}$ of all crossovers. I expect that a difference will be noticeable because this means that the number of offspring generated is reduced by $\frac{1}{4}$

3. 50%

Only $\frac{1}{2}$ half of all crossovers are carried out.

4. 25%

Of all crossovers, $\frac{3}{4}$ are omitted.

5. ILC

The authors of [14] propose a technique called Dynamic Decreasing of high mutation ratio/dynamic increasing of low crossover ratio (DHM/ILC), where the mutation rate decreases linearly over time from 100% to 0% and the crossover rate increases from linearly from 0% to 100%.

6. DHC

Also, the authors of [14] propose a technique called Dynamic Increasing of low mutation ratio/dynamic decreasing of high crossover ratio (ILM/DHC), where the mutation rate increases linearly over time from 0% to 100% and the crossover rate decreases from 100% to 0%

The authors claim that both, DHM/ILC and ILM/DHC outperform the common static crossover and mutation ratios. However, I only change the crossover rate and leave the mutation rate unchanged, e.g. "ILC" and "DHC". The full DHM/ILC and ILM/DHC will be checked in the corresponding subsection of section 2.1.3.

Result

Figure 7 shows the fitness development throughout the evolution when optimizing a QUBO with 2500 variables. The observations were the same for smaller problem instances with 255 variables and 100 variables, respectively. Because of these similarities, the graphs are omitted in these results, Figure 7 is also a reference for other problem sizes.

Higher crossover rates have a better convergence behavior and achieve better results than lower crossover rates.

Changing the crossover rate over time converges better than a low crossover rate of 25%, however worse than high crossover rates. As Figure 7 shows, there was no significant difference between increasing or decreasing the crossover rate.

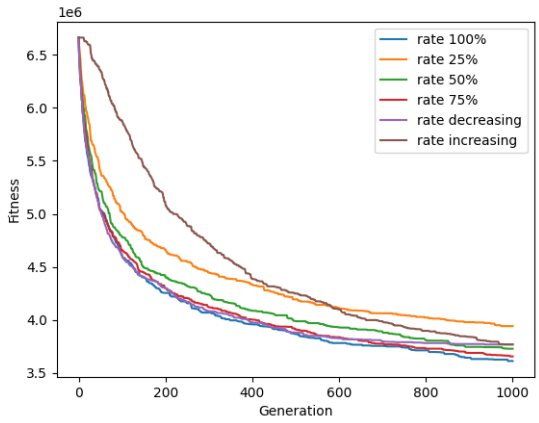


Figure 7: Crossover rate convergence for TSP with 50 nodes

The method of selection by ranking and a pressure evaluated in the previous section determines *how* the solution space is explored by also considering some less promising areas. On the opposite, the crossover rate determines *whether at all* the solution space is explored, independent of the selected individuals rank or fitness. This results in an algorithm arbitrarily omitting the generation of new solutions, which is neither exploring nor exploiting. I conclude that it is not purposeful to mimic this aspect of natural evolution.

Crossover Methods

As mentioned at the beginning of section 2, the QUBO formalism comes with the challenge that problem-specific crossover operators can not be applied. Also, the QUBO formalism only works with binary variables, which limits the possibilities of how to perform the crossover even more. However, there are some well-studied crossover operators that are problem-specific and act on binary variables. In [25] Stjepan Picek et al. provided a comparison of common approaches for crossovers on binary variables, with the result that the uniform crossover and the one-point crossover, among others, performed best on their set of optimization problems. The authors state that the result can not be generalized and applied to every problem. Hence, aside from the k-point crossover of which the one-point crossover is a variant, and the uniform crossover, other crossover methods and variants are taken into account. All crossover methods work by generating a sequence with values $\in \{0, 1\}$, the so-called crossover sequence, where 0 and 1 indicate if the offspring inherits its genes from parent 1 or parent 2, respectively. Most crossover methods produce two offspring by negotiating the crossover sequence for the second individual. Exceptionally, the uniform crossover method generates a single individual, and this procedure is performed twice to yield two individuals from a pair of parents.

K-Point Crossover

The K-Point Crossover generates the crossover sequence by first generating k random crossover points. Example for $k = 2$ and $n = 16$:

$$[\ 4 \quad 10\ ]$$

With these crossover points, the crossover sequence is created by setting the first 4 bits to 1, the bits 5-10 to 0, and 11-16 to one [25] [26] [12]:

$$[\ 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1\ ]$$

Variants

1. $k = 1$ with a fixed point

The baseline algorithm implements a special case of the k-point crossover, where $k = 1$ and the crossover point is located in the middle of the chromosome. This simply cuts the chromosome in two halves, resulting in two genomes.

2. $k = 1$ with a random point

The one-point crossover $k = 1$ is the random counterpart to the implementation in the base algorithm and is used to examine if the added flexibility to the crossover point has an impact on the result.

3. $k = 2$ with random points

To verify if the number of randomly generated crossover points matters, the K-Point crossover variant with $k = 2$ is also compared.

Uniform Crossover

The uniform crossover generates a sequence of n random values uniformly drawn from the interval $[0, 1)$.

$$[\ r_1 \quad r_2 \quad r_3 \quad r_4 \quad r_5 \quad r_6 \quad r_7 \quad r_8 \quad r_9 \quad r_{10} \quad r_{11} \quad r_{12} \quad r_{13} \quad r_{14} \quad r_{15} \quad r_{16}\ ]$$

In the standard form, a probability p of 0.5 determines from which parent the offspring inherit their genes. That is, if r $<$ p, the offspring inherits the gene from parent 1, and if r $>$ p from parent 2 [12]. This probability can be used to push the resulting offspring towards one parent [30]. I use this idea to decide if an offspring inherits genes from the stronger parent. The higher the probability p, the more likely the offspring will inherit the stronger parent's gene, which is why I call p a *bias*. This crossover just generates one offspring, which is why it is executed two times on the same pair of parents.

Variants

1. $b = 0.5$

For the sake of completeness, the uniform crossover with a bias of 0.5 is the same as the so-called random respective crossover [12]. This bias is used as a reference if shifting the probability to the stronger parent is purposeful at all.

2. $b = 0.6$

A bias of 0.6 is considered because I assume a rather small effect, as a lot of genes are still inherited from the weaker parent. Anyways, only a small effect could give insights towards exploration and exploitation, as the bias 0.5 is completely random and thus only exploring dependent on the parent solutions, while a bias of 0.6 should search more around the stronger solutions.

3. b *increasing*

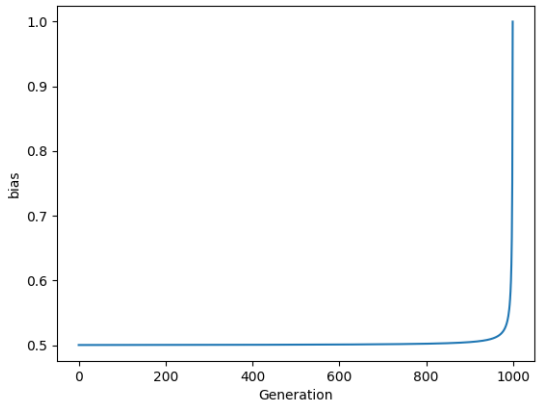


Figure 8: Increasing Bias

To adapt the idea of varying this parameter over time, an increasing bias from 0.5 to 1 is also considered. The function increasing the bias b is given by

$$b = 0.5 + \frac{1}{2} \cdot \frac{1}{totalGenerations - gen} \tag{16}$$

This is illustrated in Figure 8 8

Again pointing out the balance of exploration and exploitation, I assume that this results in an algorithm that has the chance to explore the solution space based on the parents and searches strongly around the strong parents at the end of the evolution.

Squares

To expand the collection of tested crossovers with methods a little more tailored to the problem and its representation, I try to form genomes that align with the problem structure. To recap the QUBO formulation for TSP, the penalty terms try enforce that

$$\begin{aligned} s.t. \quad & \sum_{i=0}^{n} x_{i,t} = 1, \ t = 1, ..., n \\ & \sum_{t=0}^{n} x_{i,t} = 1, \ i = 1, ..., n \end{aligned} \tag{17}$$

That is, the solution should have only one gene set to 1 for every i, and only one gene set to 1 for every t. In other words, reshaping this chromosome vector with 16 genes

$$[\ x_{1,1} \quad x_{1,2} \quad x_{1,3} \quad x_{1,4} \quad x_{2,1} \quad x_{2,2} \quad x_{2,3} \quad x_{2,4} \quad x_{3,1} \quad x_{3,2} \quad x_{3,3} \quad x_{3,4} \quad x_{4,1} \quad x_{4,2} \quad x_{4,3} \quad x_{4,4}\ ]$$

into a matrix results in

$$\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{bmatrix}$$

In this shape, the penalties enforce that for each row and each column, only one gene is set to 1. It can be interpreted as a time-location representation where each row represents a location and each column is a point in time. Since the TSP considered is a symmetric TSP, each row can also be interpreted as a point in time and each column as a location. The result would be the same cycle in the graph.

Coming from this perspective, the square crossover orders genes into genomes by cutting the time-location representation into squares. A square of size $m \times m$, therefore, represents m nodes of the TSP graph to m points in time. The crossover orders genes of the chromosome in the time-location representation into squares of equal sizes. A random number of genomes is used to finally do the crossover, such that the first offspring inherits the randomly selected genomes from parent 1 and everything else from parent 2, and the second offspring inherits the selected genomes from parent 2 and everything else from parent 1.

Variants

1. *squares* 2×2

This variant orders genes $2x2 = 4$ genes into genomes. This is a reference for small genomes. Note that this implies that the number of nodes of the TSP is divisible by 2. For experiments with a TSP with 15 nodes, this is replaced by squares of $3x3$

2. *squares* 5×5

This variant orders genes $5x5 = 25$ genes into genomes, which I assume is a reference for larger genomes, albeit *large* of cause is generally relative to the number of decision variables. For a QUBO with 2500 variables, this is rather small. The number of nodes of the TSP must be divisible by 5.

Rows and Columns

This crossover also orders genes into genomes according to the time-location representation of the chromosome. As mentioned, a row in this representation is a point in time, and a column in this representation is a location. This crossover randomly decides if the genomes are ordered into rows or columns. For a chromosome with 16 genes, the row crossover sequence in the time-location representation would be

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

and written as a vector

$$[\ 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0\ ]$$

Note that this is a special case of the K-Point crossover, where the list of crossover points contains points at positions i in the chromosome where $0 \equiv i \mod 2$. Likewise, the crossover sequence of the column crossover would be

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

and written as a vector

$$[\ 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0\ ]$$

This again is a special case of the K-Point crossover, where the crossover points are at a fixed position i in the chromosome where $0 \equiv t \mod 2$.

This crossover method randomly decides if the crossover is either done by rows or by columns.

No other variants are considered.

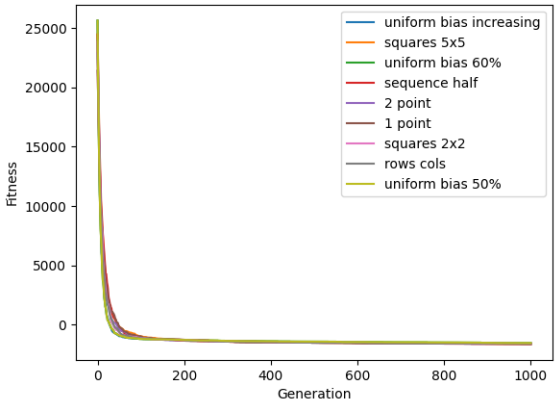
Result

Figure 9 shows the optimization results of QUBO with 100 variables. Again, fitness development is shown throughout the evolution.

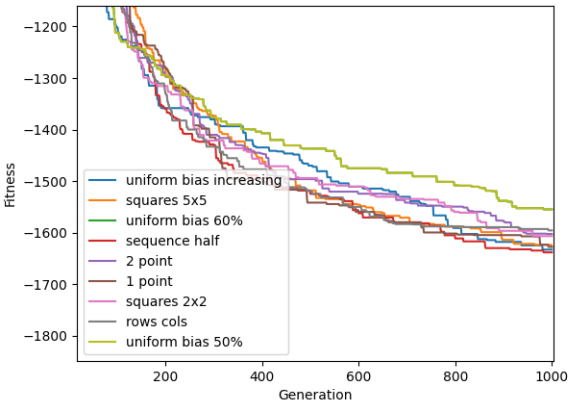
The K-Point crossover method with $k = 1$ in the random and static variant is a little better than others in convergence and result quality. The static variant is called *sequence half* in the Figure because it, as explained, cuts the chromosome into two equal-sized halves.

The Uniform crossover method has performed a little worse than others, only the variant with an increasing bias is competitive with other crossover methods. That is, the assumption that the exploration and exploitation can generally be balanced meaningfully by the bias.

The problem-tailored crossover methods both are slightly worse than the K-Point Crossover, but better than the uniform crossover, except for the variant with an increasing bias.



(a) Crossover methods convergence behavior for TSP with 10 nodes



(b) Crossover methods convergence behavior for TSP with 10 nodes, Zoomed

Figure 9: Crossover methods for TSP with 10 nodes

With this, they overall rank in the midfield.

Figure 10a shows the optimization results of a QUBO with 255 variables.

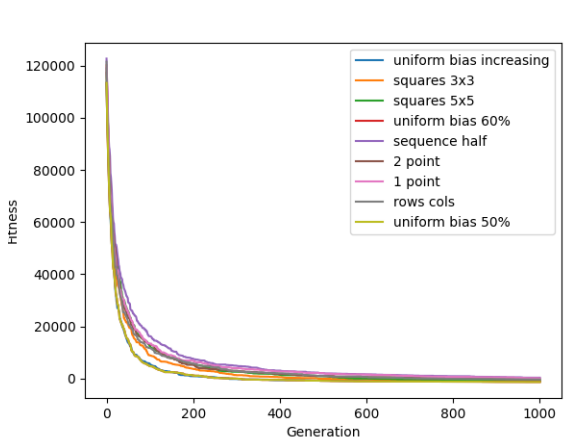
The K-point crossover method is not suitable as it is outperformed by every other method. However, as $k = 2$ achieved better results than the variants where $k = 1$, I conclude that the number of crossover points, and with that the size of the genomes, matters.

The Uniform crossover method and its variants are suited significantly better for more variables and the variants with a bias of 0.5 and an increasing bias are the best in comparison to all other crossover methods considered. Shifting the bias towards the better parent therefore has little effect on this problem size.

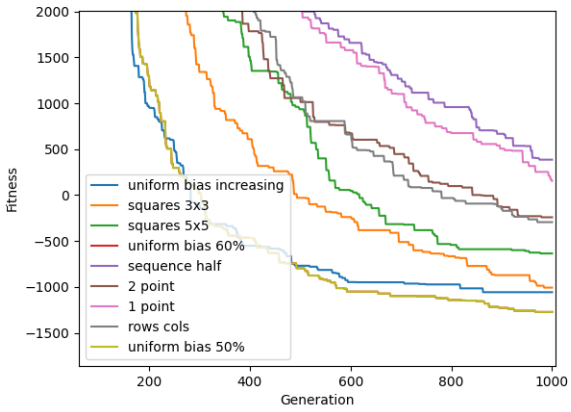
The problem-tailored crossovers on average still rank in the midfield. The variant of the crossover method that works with squares has a worse convergence than the uniform crossover method with an increasing bias but obtains the same result quality.

The tendencies observed for growing problem size from 100 to 225 variables are also observed in the experiments with a QUBO with 2500 variables, as shown in Figure 11.

The K-Point crossover is not suited for that many variables.



(a) Crossover methods convergence behavior for TSP with 15 nodes



(b) Crossover methods convergence behavior for TSP with 15 nodes, Zoomed

Figure 10: Crossover methods for TSP with 15 nodes

The crossover methods that are related to the problem rank in the midfield.

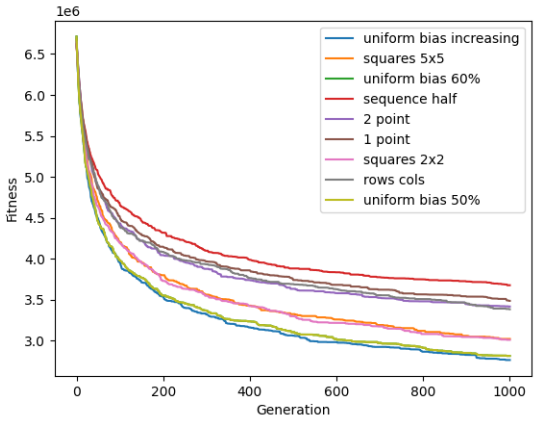
The Uniform crossover method and all its variants achieve the best results. The uniform crossover with an increasing bias was the strongest variant, followed by the random bias of 0.5. I conclude that the increasing bias was stronger than the bias of 0.6 because of the timing of exploration in the beginning and exploitation at the end of the evolution.

As mentioned, the crossover methods that order genes into genomes by rows and columns or squares act on the time-location representation and relate to the K-Point crossover. Derived from the observations made, I conclude that ordering genes into genomes that are meaningful in the underlying problem is an improvement to the ordering of the K-Point crossover. Anyways, genes that are ordered into genomes still have dependencies on genes that are not part of the genome. For example, consider two individuals with their chromosomes in the time-location representation:

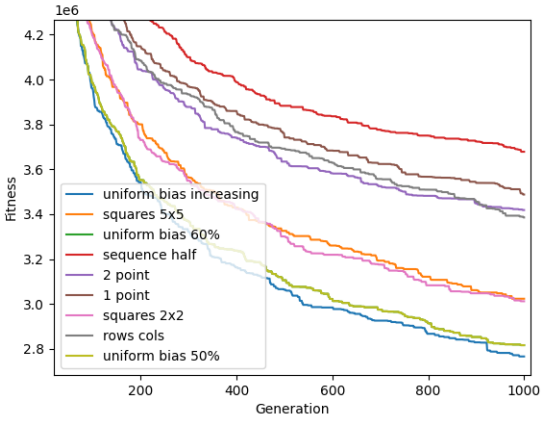
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Both individuals represent valid solutions. The rows and columns crossover applied to these two individuals can result for example this offspring:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



(a) Crossover methods convergence behavior for TSP with 50 nodes



(b) Crossover methods convergence behavior for TSP with 50 nodes, Zoomed

Figure 11: Crossover methods for TSP with 50 nodes

The result is an invalid solution. If an individual has a gene set to 1 in a row and column and inherits this gene, no other gene in this row or column can be set or otherwise the solution is invalid. Two different individuals encoding two different solutions both have in *every* row and column *exactly* one gene set to 1, that is, if an offspring inherits a row from one parent, there *must* be a row in the other parent chromosome that conflicts with this gene. Two individuals that encode two valid solutions will therefore *never* recombine into a third, *valid* solution. The only way to produce a valid solution is the case where the offspring is equivalent to the parent. On the other hand, consider a uniform crossover sequence of

$$[\ 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0\ ]$$

and the two valid solutions

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

will result in

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

This new individual now encodes a valid solution, too, and is different than both parents. The rows and columns crossover method would again produce an invalid solution. I

figure that the probability of creating a new valid solution with the uniform crossover method is higher than with the rows and columns crossover method and the squares crossover method and I figure that this is why the uniform crossover performed best for the considered problem.

2.1.3 Mutation Operator

In nature, a species may develop and change its physical appearance and capabilities over time. This evolutionary change is driven by mutation. Mutations can naturally occur in the process of reproduction, such that an individual has some random genes that are not inherited by one of its parents. If this mutation gives that individual an advantage, that individual is more likely to survive and reproduce than other individuals without this mutation. By this, this mutation is populated in the species in subsequent generations.

The mutation operator mimics this important evolutionary step. The common method to implement this for EA is to randomly change one or more genes. This probability is called the mutation rate. Given this mutation rate mr, a random number $r \in [0, 1)$ is drawn uniformly for every gene. If $r < mr$, the gene is negotiated. That is, the higher the mutation rate, the more genes are affected. This again determines exploration and exploitation. The more genes are affected by random changes, the more the algorithm is allowed to make random steps through the solution space.

Variants

1. Mutation rate 1%

This is a reference for a mutation operator that makes only small steps through the solution space and thus explores the solution space only by a little.

2. Mutation rate 5%

This mutation rate will change 5% of all genes and make moderate steps through the solution space. This is the rate that the base algorithm implements.

3. Mutation rate 10%

I assume that this is a relatively large value for the mutation rate as on average 10% of all genes are changed.

4. ILM

Just like with the crossover rate, I consider changing the mutation rate linearly over time from 0% to 100%, as the authors do in [14] 5. DHM

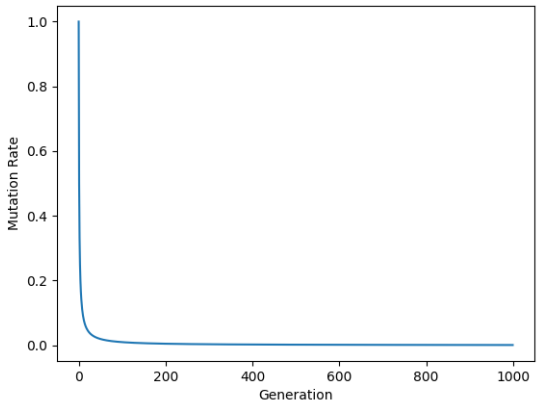


Figure 12: Hyperbolic mutation rate

I also implement the analog of decreasing the rate, from 100% to 0% during the evolution. 6. Decrease hyperbolic

Lastly, I consider changing the mutation rate hyperbolic, given by the function

$$mr = \frac{1}{gen} \tag{18}$$

where *gen* denotes the current generation. The result is the function in Figure 12 I assume that a very high mutation rate at the beginning that decreases strongly early in the evolution will result in an algorithm that has a strong exploratory behavior in the beginning of the evolution and again searches more around the present individuals in later parts of the evolution.

Result

Figure 13 shows the optimization results of a QUBO with 100 variables.

The resulting solution qualities are similar for all mutation rates. Anyways, there are differences in the convergence of the population towards the solutions.

Decreasing the mutation rate had the overall worst result. Increasing the mutation rate linearly performed better than the decreasing counterpart. Surprisingly, decreasing the mutation rate hyperbolic achieves the best results in convergence and solution quality within the class of dynamically changing mutation rates.

The static mutation rates on average perform better than dynamic rates. Generally the lower the mutation rate, the better the overall result.

Figure 14 shows the experiment results with a QUBO with 255 variables.

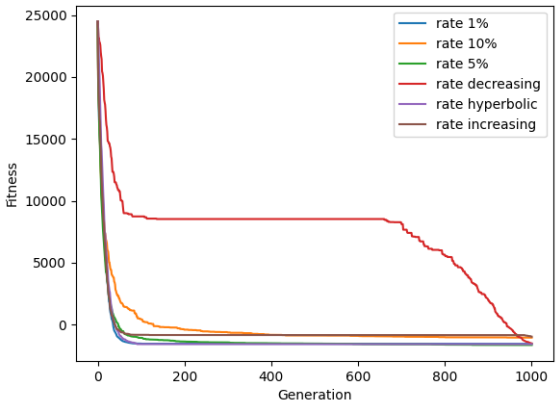


Figure 13: Mutation rates for TSP with 10 nodes

Linearly changing the mutation rate and the higher static mutation rate of 10% is not purposeful, as they show premature convergence and bad solution qualities.

Lower, static mutation rates achieve better results than the mentioned approaches above.

Changing the mutation rate hyperbolic is as good as the mutation rate 1%.

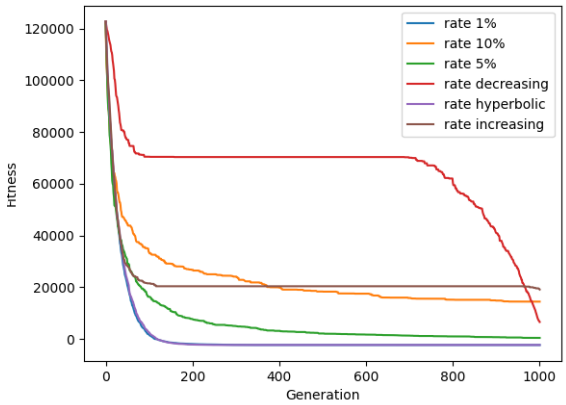


Figure 14: Mutation rates for TSP with 15 nodes

Figure 15 shows the results for a QUBO with 2500 variables. Again, the lower the mutation rate, the better the result. Linearly changing the mutation rate performs worse than a static rate. The lower the static rate, the better the result. However, decreasing the mutation rate hyperbolic outperforms all static variants.

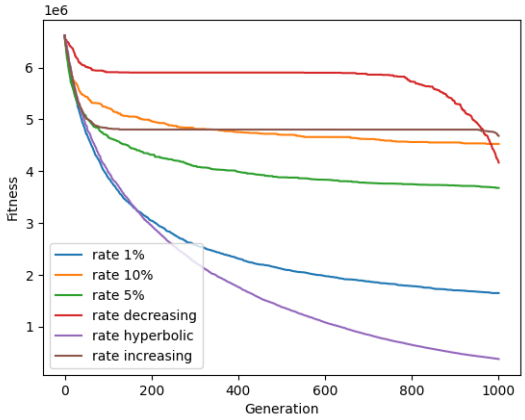


Figure 15: Mutation rates for TSP with 50 nodes

One aspect of the mutation rate is the average mutation rate throughout the evolution. Changing the mutation linearly results in an average mutation rate of 50%, which is much higher than the considered static mutation rates. The hyperbolic mutation rate is high at the beginning of the evolution, but close to 0% most of the time, resulting in a low average mutation rate.

Another aspect is the timing of when the mutation rate is high. Linearly decreasing the mutation rate results in the algorithm being unable to narrow the solution down for a long time. Increasing the mutation rate results in an algorithm finding better solutions at first but then getting stuck due to randomness. Moreover, by linearly decreasing the mutation rate during the evolution, the algorithm ends up with a better solution quality than its counterpart.

Both aspects together show that a high mutation rate at the beginning of the evolution is better than a high mutation rate at the end of the evolution, and lower average mutation rates are better than high average mutation rates. Decreasing the mutation rate hyperbolic conjuncts these aspects, which explains why this approach achieved the best result for this problem. That is, giving the algorithm the opportunity to explore the solution space at the beginning of the evolution but restricting this exploration quickly, such that new solutions are close to solutions present in the population and the steps made in the solution space by mutation are relatively small, is the best choice.

ILM/DHC and DHM/ILC

Crossover rates lower than 100% were not purposeful. Since the authors of [14] claim that changing the mutation rate and changing the crossover rate over time opposite to

one another can give good results, the compositions of increasing and decreasing the crossover rate and mutation rate are evaluated in this section.

ILM/DHC linearly increases the mutation rate from 0% to 100% and decreases the crossover rate from 100% to 0%.

DHM/ILC linearly decreases the mutation rate from 100% to 0% and increases the crossover rate from 0% to 100%.

The obtained result in 16 is that DHM/ILC finds good solutions at the end of the evolution. ILM/DHC finds good solutions at the beginning of the evolution. DHM/ILC achieve the best solution quality. The overall result of this experiment is not too different

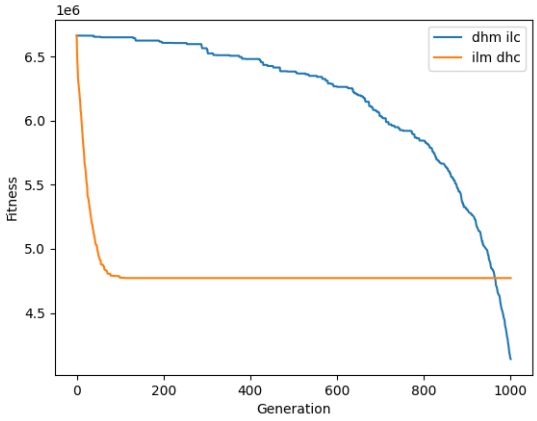


Figure 16: Adapting rates for TSP with 50 nodes

from the observations made with different mutation rates. High mutation rates in the beginning converge slower than high mutation rates at the end of the evolution.

Anyways, comparing these results with a static crossover rate of 100% in Figure 7 and a mutation rate of 1% in Figure 15, ILM/DHC and DHM/ILC both are relatively bad in convergence and solution quality, so my observation differs from those that the authors made in [14]. It has to be said that this can vary depending on the problem representation and instance and is possibly only true for the problem considered in this thesis. Also, the authors strongly put the population size into perspective, while I focus on result quality and convergence. I figure the effect on population sizes is caused by the fact that, with a high mutation rate, large steps through the solution space are made, but because of a low crossover rate, these results are rarely (but not never) distributed. With a low mutation rate and high crossover rate, the exploitative behavior is reduced and the solutions are more likely to be distributed by the high crossover rate. In other words, the smaller the step size given through the mutation rate, the more often these steps are made. With this attempt to balance the algorithms' exploration and exploitation of the

solutions space, other implementations of EAs may improve.
Definitely still worth consideration since negative results on this methodology, allowed some conclusions, especially DHM and ILM for the mutation rate in the previous section.

2.1.4 Different Compositions

The results obtained from the last section are now used to assemble an algorithm. Note that EAs are a heuristic that allows creativity, the amount of operators to choose from is endless. I do not rule out that there is an operator that has not been considered but would improve the algorithm. Anyways, the most prominent operators were examined. It is essential to mention that some operators that archived worse results than others when used alone may work better in combination with some other operators. Hence, to find the best combination of operator methods, the cartesian product of all operator methods must be evaluated rather than just compressing the best methods into a final result. However, the goal of this section is to find a good enough combination of different operators that can then be extended by some local optimization strategy.

To reduce the set of different compositions, I assume that the crossover rate of 100% is the best choice for crossover and has a rather small impact on other operators. I assume this because the crossover rate only determines if parents reproduce and thus only the quantity and not the quality of the produced offspring.

The selection operator is decoupled from the crossover operator and mutation operator in the means of selecting individuals on one side and generating individuals on the other. While the selection operator influences which individuals reproduce and by this influences the offspring quality indirectly, the crossover operator and mutation operator influence the offspring quality directly. I choose the selection tournament method with $k = 20\%$. I assume that other selection methods would not significantly improve when combined with other crossover and mutation methods or variants than the ones used in the baseline algorithm.

The crossover operator and the mutation operator can correlate with one another because the mutation operator may fix or destroy the result of a crossover. To reduce the set of different methods and variants considered in this section further, the worst methods and variants for crossover and mutation are ruled out, such as the K-Point crossover method, its corresponding variants and linearly changing the mutation rate.

To summarize, the best composition of cartesian products in

is sought. Hence $|Crossovers \times Mutation\ \ Rates| = |Crossovers| \cdot |Mutation\ \ Rates| =$ 12 compositions are compared.

For the squares crossover method, the 5x5 variant is used.

Crossovers		Mutation Rates
uniform bias increasing		mutation rate 1%
uniform bias 0.5	×	mutation rate 5%
rows and columns		mutation rate 10%
squares		mutation rate hyperbolic

Table 1: Sets of crossover methods and mutation rates

From now on, only the TSP problem instance with 50 nodes and 2500 variables will be used to benchmark.

Uniform Crossover with Bias increasing and different mutation rates

Figure 17 shows the convergences of the uniform crossover with an increasing bias when used with the different mutation rates. Generally, the uniform crossover tends to early convergence with higher mutation rates and achieves worse solution qualities, that is, fitness. The best result however achieved the mutation rate that is decreases hyperbolic

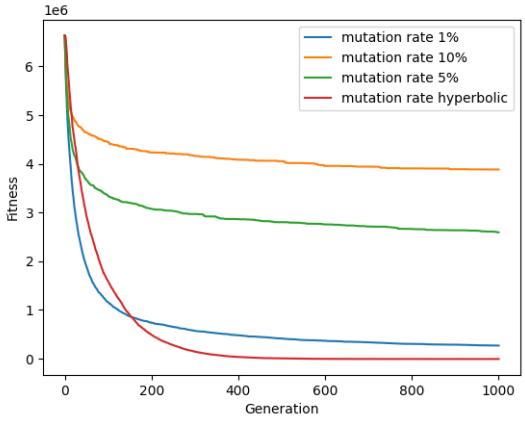


Figure 17: Compositions with uniform crossover bias increasing

during the evolution. Since this observation is the same as the observations made in the comparison of different mutation variants in the previous section, I conclude that the uniform crossover with 0.5% is independent of the mutation rate in means of effect to the latter.

Uniform Crossover with Bias 0.5 and different mutation rates

Figure 18 shows the experiment results of the composition of a uniform crossover biased with 0.5 and different mutation rates. The result is the same as it was for the uniform crossover with an increasing bias. Again I conclude that the uniform crossover generally does not affect how the parameter of mutation rate is to be chosen.

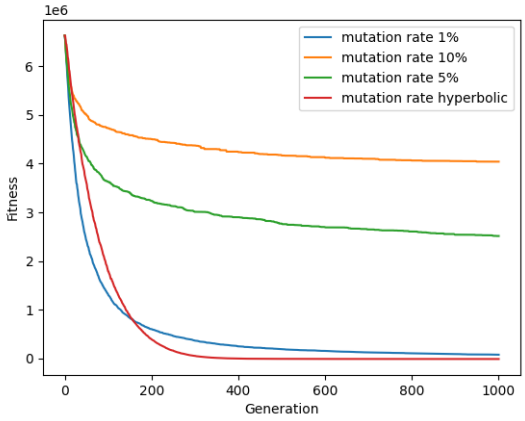


Figure 18: Compositions with uniform crossover bias 0.5

Rows and Columns Crossover with different mutation rates

Figure 19 shows the experiment results of the rows and columns crossover composed with different mutation rates. The observations are the again the same.

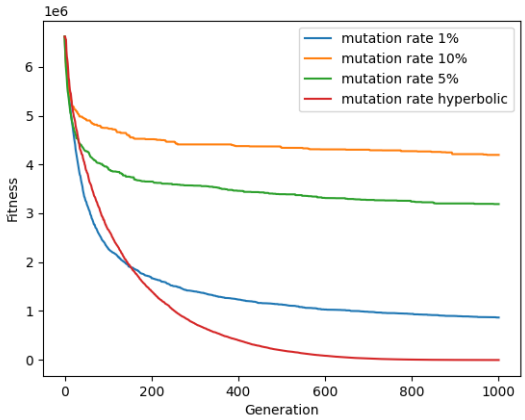


Figure 19: Compositions with rows and columns crossover

Squares Crossover with different mutation rates

Figure 20 shows the optimization results of compositions of the squares crossover method with different mutation rates. The observations are again the same. I conclude that the crossover operator has no effect on the mutation rate.

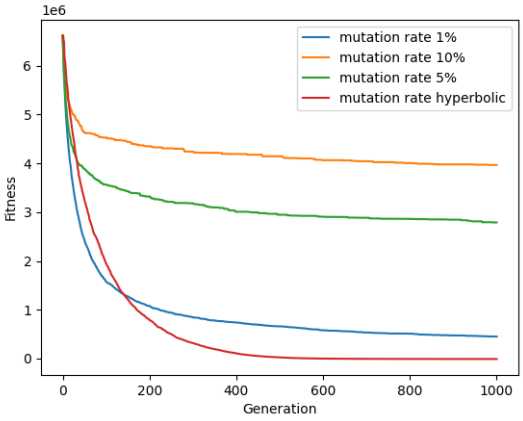
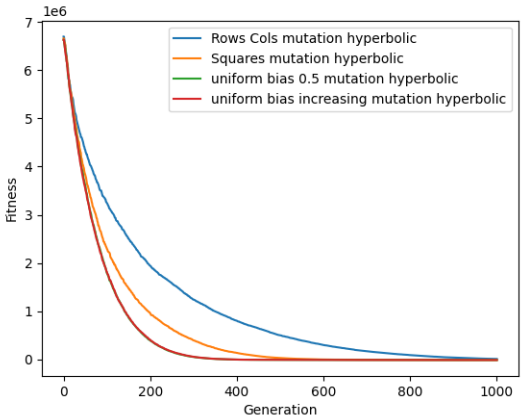


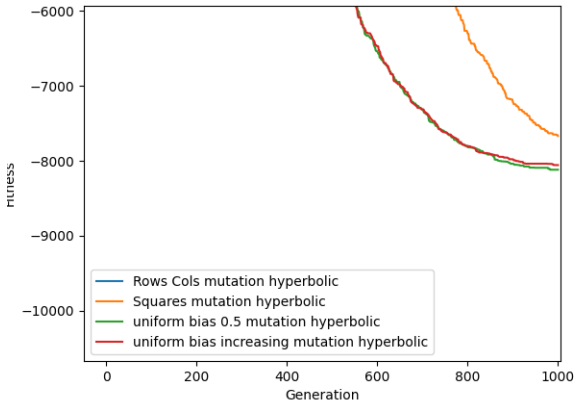
Figure 20: Compositions with squares crossover

Result

For all compositions tested, every crossover method performs best with the mutation rate that decreases hyperbolic over time. To finally assemble the classic algorithm, all crossover methods with the decreasing mutation rate are compared to each other in Figures 21a and 21b.



(a) Best Composition for every crossover



(b) Best Composition for every crossover

Figure 21: Result for compositions

The rows and columns crossover converges slower than other crossover methods and achieves the worst result quality.

The squares crossover converges noticeably slower than the uniform crossover method

with static and increasing bias, however better than the rows and columns crossover. The resulting quality is similar to the resulting quality of the uniform crossovers.

The uniform crossover method with both, a static bias and an increasing bias performed best in convergence and result quality. The observations for both variants are almost identical. The slight difference can be seen in Figure 21b, where the green line of the uniform crossover with a static bias of 0.5 is negligible below the red line for the uniform crossover with an increasing bias.

Recalling that the experiment shown in Figure 11 was to optimize a QUBO with 2500 variables. The uniform crossover that increases the bias was the strongest in that experiment. Interestingly, the uniform crossover with a bias of 0.5 now achieves slightly better results than the uniform crossover with an increasing bias, albeit the observation of the experiments regarding different compositions of crossover methods and mutation rates was that the crossover has no effect on the mutation rate.

I conclude that this observation but in reverse is not true. Even though the crossover operator generally has no effect on which mutation rate is the strongest, the mutation rate can improve the crossover method.

With these results obtained, the algorithm that will be extended by a local optimization is the composition of

- Selection Linear Ranking with $sp = 2$
- Uniform crossover with a static bias of 0.5
- A mutation rate that decreases hyperbolic

2.1.5 Classic Local Optimization

As mentioned in the introductory chapter, the extension of an EA by a local search represents the plasticity of an individual to adapt to the environment. One approach in optimization is to fix some decision variables and optimize the problem dependent on the fixed variables.
Consider a binary vector x with $n = 3$ variables and a QUBO matrix Q of dimension $n \times n$, $q_{i,j} \in Q$. With values of $x_1 = 0$, $x_2 = 1$ and $x_3 = 0$ and x_2, x_3 are fixed and with x_1 being the only decision variable, the optimization problem is now:

$$\begin{aligned} &q_{1,1}x_1x_1 + q_{1,2}x_1 \cdot 1 + q_{1,3}x_1 \cdot 0 \\ &+q_{2,1}x_1 \cdot 1 + q_{2,2} \cdot 1 \cdot 1 + q_{2,3}1 \cdot 0 = q_{1,1}x_1x_1 + q_{1,2}x_1 \cdot 1 + q_{2,1}x_1 \cdot 1 + q_{2,2} \\ &+q_{3,1}x_1 \cdot 0 + q_{3,2} \cdot 0 \cdot 1 + q_{3,3} \cdot 0 \end{aligned} \tag{19}$$

Note that setting $x_1 = 0$ will result in

$$q_{1,1} \cdot 0 \cdot 0 + q_{1,2} \cdot 0 \cdot 1 + q_{2,1} \cdot 0 \cdot 1 + q_{2,2} = q_{2,2} \tag{20}$$

In minimization problems, $x_1 = 1$ is the better choice for x_1 than $x_1 = 0$, that is, an improvement, only if

$$q_{1,1} \cdot 1 \cdot 1 + q_{1,2} \cdot 1 \cdot 1 + q_{2,1} \cdot 1 \cdot 1 + q_{2,2} < q_{2,2} \tag{21}$$

This is the case exactly when

$$q_{1,1} + q_{1,2} + q_{2,1} < 0 \tag{22}$$

Derived from that, the upper bound for the minimum of the problem only depending on the decision variable x_1 is 0. To generalize this observation for a decision variable x_i, it is sufficient to verify if

$$x_i(q_{i,i} + \sum_k (q_{i,k} + q_{k,i})x_k) < 0 \mid k \in B \tag{23}$$

where B is the index set of fixed variables. This means informally that only the row and column i of Q are optimized.

The local optimization used for the Classic Local Optimization now iteratively optimizes every x_i individually according to Eq. 23, such that each x_i is optimized once.

According to [4], individuals are selected arbitrarily. Also, the local optimization is meant to extend the EA, not replace it. The probability of an individual being selected by the local optimization is set to 0.005, that is 0.5%.

Variants

1. *ordered*

The local optimization iteration starts at x_1 and ends with x_n

2. *unordered*

The local optimization iteration is in a randomly generated order

Result

Figure 22 shows the experiment results. Both strategies improve the algorithm significantly compared to the algorithm without a local optimization in Figure 21. Randomly choosing the order of optimization however yields a slightly better result. This could be caused by the fact that this allows the local optimization to create a higher diversity of individuals, which again gives the insight that exploration of the solution space is important and can be achieved with more subtle factors than for example the mutation rate.

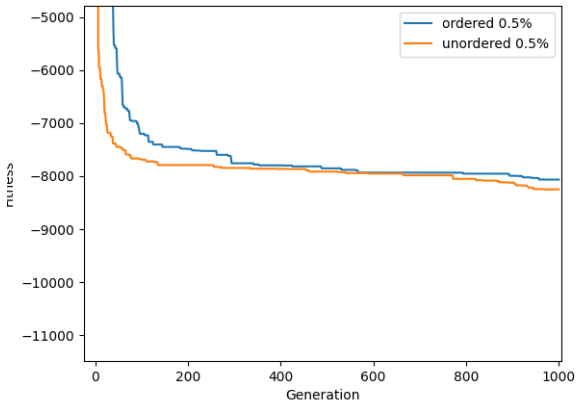


Figure 22: Convergences for Classic Local Optimization

The development of the EA with Classic Local Optimization is completed in this section. Also, since the results of the two variants compared are similar, I will now analyze some more statistical properties of the algorithm.

Figure 23 shows the histogram of multiple executions on the same problem, with the resulting frequency on the y-axis and the fitness on the x-axis. Table 2 presents the corresponding statistical data. Generally, the objective is to find the variant where the frequencies of solutions are rather to the left side of the histogram because lower fitness

values are the better solutions to the minimization problem. Also, the distances between the results, that is, the gap between the vertical bars, that is, scattering, is important in means of the reliability of the algorithm and reveals if the algorithm is working directed or rather arbitrarily. That is, an algorithm with a lower mean value and the lowest standard derivation is better than an algorithm with a high mean and a high standard derivation.

The algorithm with the ordered iteration approach achieves an average result of -8066.2 with a standard derivation of ~ 136. The unordered iteration yields better results, with a mean of -8251.2 and a standard derivation of ~ 73.

Recalling that the algorithm optimizes an objective function with 2^{2500} possible solutions, and the best individual in the first generation has a fitness value $>$ 600k, the difference in both, mean and standard derivation, is rather small. Also, this analysis shows that the algorithm extended by both variants for Classic Local Optimization does not produce arbitrary results.

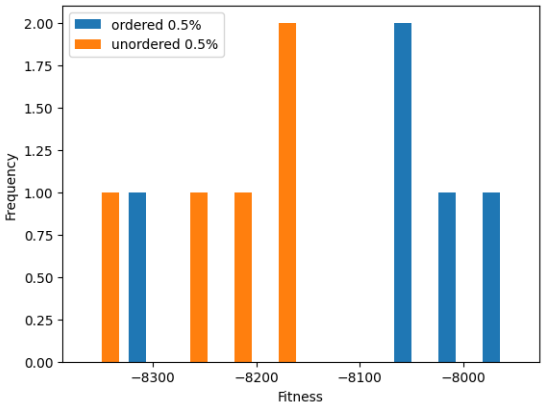


Figure 23: Histogram for Classic Local Optimization

Strategy	Mean	Median	Variance	Standard Derivation
Ordered 0.5%	-8066.2	-8032.0	18451.7	135.8370347143959
Unordered 0.5%	-8251.2	-8231.0	5385.2	73.3839219447966

Table 2: Statistic analysis for Classic Local Optimization

To summarize, the algorithm extended by the local optimization that optimizes variables in a random order is the better one. I assume that this is, as mentioned, because

of a very subtle balance of exploration and exploitation, where the randomness in the optimization order explores the solution space, and the local optimization itself is exploitation.

2.2 Quantum-Hybrid Evolutionary Algorithm

The objective of this section is to replace the Classic Local Optimization with a local optimization technique that is suited for Quantum Annealers. Thus, the derived sub-problems to optimize must be in the format of a QUBO matrix. With this, instead of a single decision variable, multiple decision variables are optimized at once, the optimization therefore considers the dependencies between those decision variables.

Recalling Eq. 23, the Classic Local Optimization approach is to verify if for the decision variable, $x_i = 1$ the result of Eq. 23 is smaller than 0, that is, a decision.
For more than one decision variable, this is

$$\min f(x) = \sum_i \sum_j q_{i,j} x_i x_j + \sum_i \sum_k (q_{i,k} + q_{k,i}) x_i x_k \mid i, j \in A, k \in B \tag{24}$$

where A is the index set of decision variables and B is the index set of fixed variables. The task is now no longer a decision but an optimization problem, where the objective values of the decision variables are on the main diagonal axis of the new QUBO for the sub-problem, and the objective values dependent on other decision variables are off the diagonal axis and objective values dependant on the fixed variables are added to the main diagonal axis of the new QUBO matrix.

Since multiple decision variables can be optimized at once, the question is no longer just the order of optimization, but also the number of variables and how to group the variables, that is, in terms of an EA, ordering genes into genomes that are to be optimized.

Current Quantum Annealers, although they have more than 5000 qubits, often have a topology that is not fully connected [17] [1]. For instance, the D-Wave Pegasus processor has a topology where each qubit is connected with 15 other qubits [3]. That is, to define a fully connected mash, multiple qubits must be grouped into logical qubits called chains [1]. The number of decision variables that can be optimized depends on the density of the QUBO matrix.

The number of variables for the Quantum Local Optimization is set to 100 decision variables.

Next, the methods of building genomes to optimize, that is the grouping of decision variables, are introduced. The probability of optimizing an individual will be the same as for the Classic Local Optimization.

Random

The random method will order genes into random groups by generating a random permutation of indexes and then optimizes each group. For instance, if the number of decision variables was set to 4 instead of 100 and the chromosome has a length of 16, the number of groups in total would be $\frac{16}{4} = 4$. The permutation is thus split into 4 groups as shown in 27.

$$[\underbrace{1 \quad 7 \quad 5 \quad 2}_{1} \; \underbrace{8 \quad 16 \quad 6 \quad 3}_{2} \; \underbrace{10 \quad 11 \quad 13 \quad 4}_{3} \; \underbrace{15 \quad 12 \quad 14 \quad 9}_{4}] \tag{25}$$

Every block of variables 1-4, e.g. genome, is now optimized once by the Quantum Local Optimization, with every other block fixed.

For this method, no other variant is implemented.

Segments

This method also slices the chromosome into groups but generally preserves the order of genes within these groups. For instance, the chromosome with 16 variables and a segment size of 4. One group of genes, that is, the genome, is optimized at a time, with all other genomes fixed.

$$[\underbrace{1 \quad 2 \quad 3 \quad 4}_{1} \; \underbrace{5 \quad 6 \quad 7 \quad 8}_{2} \; \underbrace{9 \quad 10 \quad 11 \quad 12}_{3} \; \underbrace{13 \quad 14 \quad 15 \quad 16}_{4}] \tag{26}$$

is split into 4 groups with 4 genes each.

Variants

1. *ordered*

The chromosome is optimized from segment 1 to segment n.

2. *unordered*

The segments are optimized in a random order. The distinction from the random method is that the order of genes inside the segments is preserved.

Squares

Analog to the square crossover introduced, this method builds genomes as squares in the matrix representation of the chromosome, that is

$$\left[\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{array}\right]$$

and written as a vector:

$$[\underbrace{1\ \ 2}_{1}\ \underbrace{3\ \ 4}_{2}\ \underbrace{5\ \ 6}_{1}\ \underbrace{7\ \ 8}_{2}\ \underbrace{9\ \ 10}_{3}\ \underbrace{11\ \ 12}_{4}\ \underbrace{13\ \ 14}_{3}\ \underbrace{15\ \ 16}_{4}\ ] \tag{27}$$

Every segment again is optimized once with all other segments fixed.

Variants

1. *ordered*

The chromosome is optimized from square 1 to square n.

1. *unordered*

The squares are optimized in a random order. Again, the distinction between this variant and the random method is that the genomes themselves do not change.

All methods to sort variables into groups of decision variables and fixed variables, that is random, segments and squares, have in common with one another that they optimize every variable, that is, gene, exactly once.

Result

Figure 24 shows the experiment results with the 2500 variable QUBO and illustrates the development of the fitness value throughout the evolution.

The general observation is that the random method converges earliest and achieves the worst result quality.

Grouping genes into squares results in later convergence and also in a lower fitness

Slicing the chromosome into segments achieved the best results.

A noticeable property of both methods, the slicing method into segments and the squares methods, is that the unordered approach achieved better results for both. However, the random method performed worse. The reason for this could be that some randomness is needed for exploration, anyways leads to early convergence if there is no structure at all, which again confirms my observation that it is possible to balance exploration and exploitation in subtle manners.

Figure 25 shows the histogram of the experiments with the Quantum Local Optimization methods. Table 3 contains the corresponding statistical data.

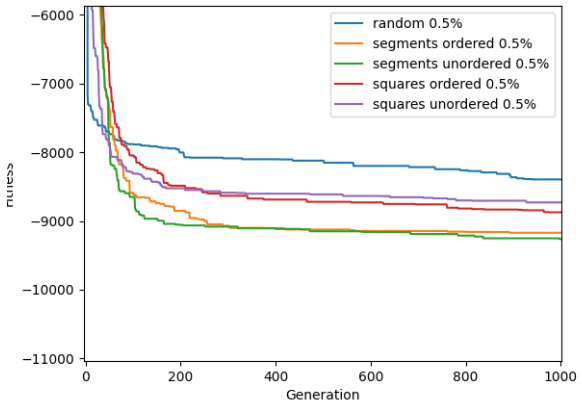


Figure 24: Convergences for Quantum Local Optimization

The methods that were observed to have better convergences and better result qualities also have better statistical properties.

The random method has the highest scattering with $\sim$ 140 standard derivation and the worst average result with -8396.

The ordered variant of the segment method achieves better results than the random method with a standard derivation of $\sim$ 53 and a mean of -9171, while the unordered variant has a standard derivation of $\sim$ 41 and a mean of $\sim$ -9263.

The squares method in the ordered variant with a standard derivation of $\sim$ 126 and a mean of -8874 performed worse than the unordered variant with a standard derivation of $\sim$ 83 and a mean of $\sim$ -8730.

Strategy	Mean	Median	Variance	Standard Derivation
Random 0.5%	-8396.0	-8433.0	19516.0	139.69967788080257
Segments Ordered 0.5%	-9171.0	-9172.0	2781.0	52.73518749374084
Segments Unordered 0.5%	-9262.8	-9260.0	1687.7	41.08162606324146
Squares Ordered 0.5%	-8874.0	-8835.0	15838.5	125.85110249815057
Squares Unordered 0.5%	-8730.4	-8749.0	6845.8	82.73934976781

Table 3: Statistic analysis for Quantum Local Optimization

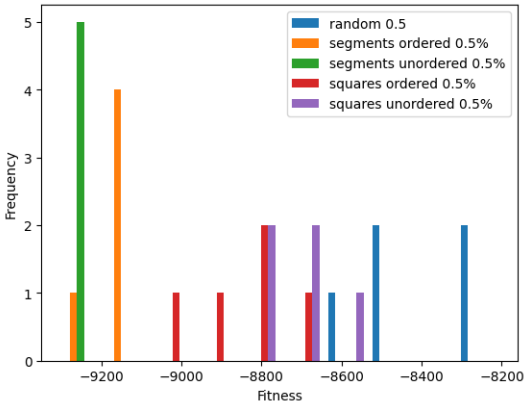


Figure 25: Histogram for Quantum Local Optmization

Ultimately, the Quantum Local Optimization method that slices the chromosome into segments and optimizes these segments in random order has the best statistical properties, the best convergence, and the best result quality.

The final QHEA is thus the classic EA extended by this local optimization.

2.3 Optimizing hyper parameters

The last parameters to balance are the population size and the number of generations executed during the evolution. Currently, the QHEA is parameterized with a population size $p = 100$ and a generation count $gen = 1000$. During the evolution, the algorithm therefore iterates over $p \cdot gen = 100000$ individuals. I assume that increasing the population size and generation count such that $p \cdot gen = |solution\ space|$ is the same as solving the optimization problem by validating every possible solution individually. I also assume that $p \cdot gen = 1$ is the same as a random sample from the solution space. That is, the objective is to determine if other values for p and gen such that $p \cdot gen = 100000$ optimize the fitness value.

Variants

1. $p = 10$ $gen = 10000$

2. $p = 100$ $gen = 1000$

3. $p = 1000$ $gen = 100$

4. $p = 10000$ $gen = 10$

The values for all variants were chosen to monitor the effect of shifting the weight from population size to generation count and compare it with the variant that was implemented throughout all experiments in this thesis.

Result

Figure 26 shows the fitness evolution over the generations for different parameter pairs p and q.

The parameter pairs $p = 10000$ and $gen = 10$ achieved the worst results (see the small red line at the top left corner in Figure 26).

All other parameter choices were close to one another. However, the parameter choices $p = 100$ and $gen = 1000$ obtained the best result quality.

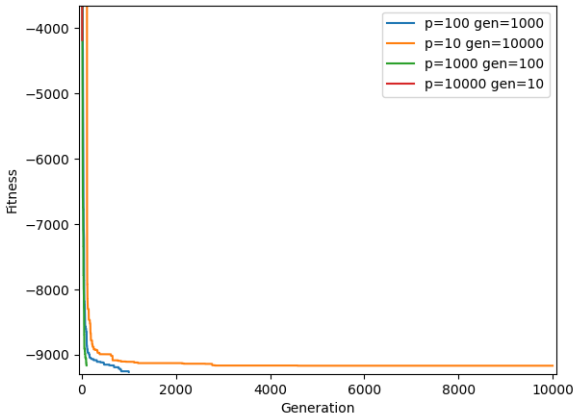


Figure 26: Convergences for parameters p and gen

The histogram in Figure 27 also shows that the scattering for $p = 100$ and $gen = 1000$ is lower than for other parameter choices. Generally, the closer p and gen are to one another, the better the distribution of results. Again, the parameter choices $p = 10000$ and $gen = 10$ achieved the worst results as they are more distributed in the solution space and also worse in fitness. Table 4 underlines this observation with a standard derivation of ~ 41 and an average result of $\sim$ -9263 that the parameter choices $p = 100$ and $gen = 1000$ are the best in this experiment.

The parameter choices $p = 1000$ $gen = 100$ and $p = 10$ and $p = 10000$ had a mean of ~ -9164 and ~ -9169 respectively. In these terms, they were similar. They defer from each other more in standard derivation, with a standard derivation of ~ 86 and ~ 70.

A very interesting comparison is between $p = 10$ and $gen = 10000$ and $p = 10000$

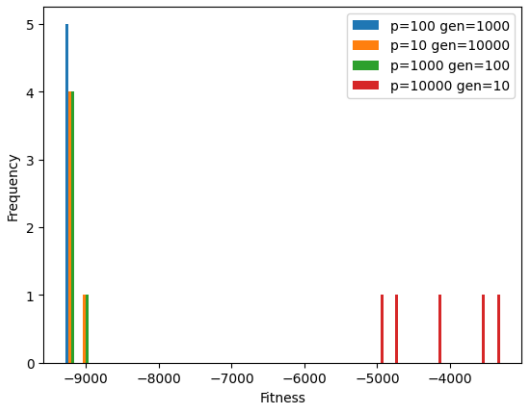


Figure 27: Histogram for parameters p and gen

and $gen = 10$. There is a significant difference between these pairs of parameters in both, mean and standard derivation.

The same observation is made when comparing $p = 100$ and $gen = 1000$ and $p = 1000$ and $gen = 100$.

In both of the last two comparisons above, the parameter values were swapped, and in both comparisons, the statistical properties are better when p is the lower value, and gen is the higher value.

Values	Mean	Median	Variance	Standard Derivation
p=10 gen=10000	-9166.8	-9175.0	4918.7	70.13344423311891
pop=100 gen=1000	-9262.8	-9260.0	1687.7	41.08162606324146
pop=1000 gen=100	-9163.2	-9185.0	7322.7	85.57277604472114
pop=10000 gen=10	-4183.4	-4193.0	613102.8	783.0088122109482

Table 4: Statistic analysis for parameters p and gen

I explain the differences between for example $p = 10$ $gen = 10000$ and $p = 10000$ $gen = 10$ that the first parameter pair allows the algorithm to apply the genetic operators more often. This means that the genetic operators used in the EA have a strong effect that increases the more often the operators are applied.

3 Analysis

The objective is to analyze the final QHEA and compare it to other solvers.

One aspect is to compare the Classic Local Optimization which optimizes one variable with the Quantum Local Optimization, to allow for a conclusion if the optimization of multiple variables by a Quantum Computer does improve the solution quality.

Also, an observation drawn from the previous section was that the evolutionary heuristic, in means of the application of genetic operators, still has an effect on the result. To create a better picture of the relevance of the evolutionary heuristic, I use the Quantum Local Optimization as a standalone heuristic and compare it with the QHEA.

There are some other solvers for QUBOs available. To evaluate the relevance of the algorithm developed in this thesis, QHEA has to be compared with them such that an assessment in this regard is possible.

The objective of this thesis was was to optimize a QUBO, rather than developing a new algorithm for TSP. Anyways, since this was the problem to benchmark, I will use QHEA and other solvers to solve a TSP instance from an official TSP benchmark library which allows to put the solution quality to the TSP into context.

3.1 EA with Classic Local Optimization and EA with Quantum Local Optimization

Figure 28 shows the fitness evolution dependent on the generations. The Figure shows that the Quantum Local Optimization was better in means of result quality and convergence.

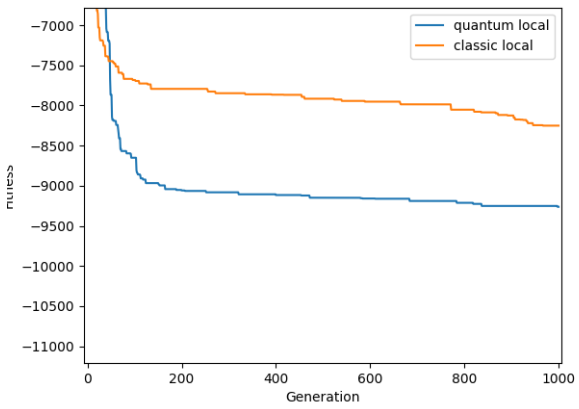


Figure 28: Convergences of Classic Local Optimization and Quantum Local Optimization

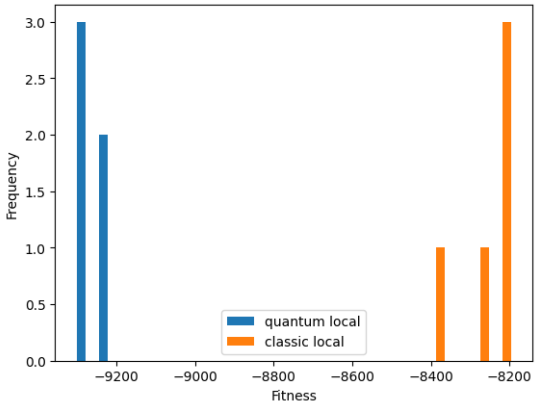


Figure 29: Histogram of Classic Local Optimization and Quantum Local Optimization

Figure 29 shows the histograms of both local optimization techniques. The Quantum Local Optimization has less scattering than the Classic Local Optimization and also generally achieves lower fitness values.

Table 5 contains the data of this comparison.
The Classic Local Optimization has a mean of ~ -8251 and a standard derivation of ~ 73, while the Quantum Local Optimization has a mean of ~ -9263 and a standard derivation of ~ 41. The difference between the two optimization techniques is therefore statistically significant.

Optimizer	Mean	Median	Variance	Standard Derivation
Classic Local	-8251.2	-8231.0	5385.2	73.3839219447966
Quantum Local	-9262.8	-9260.0	1687.7	41.08162606324146

Table 5: Statistic analysis of Classic Local Optimization and Quantum Local Optimization

3.2 EA with Quantum Local Optimization and Quantum Local Optimization only

In QHEA, with $p \cdot gen = 100000$ and a probability for local optimization of 0.5%, the local optimization is executed approximately 500 times. To determine the impact of the evolution on the solution quality, the Quantum Local Optimization is used to optimize a randomly initialized starting solution 500 times, without the surroundings of an EA heuristic.

Figure 30 shows the histogram that compares the distribution of multiple executions of both algorithms. It generally shows that QHEA has a lower scattering and also achieves

better fitness values than the Quantum Local Optimization as a standalone heuristic. The use of the EA is therefore justified.

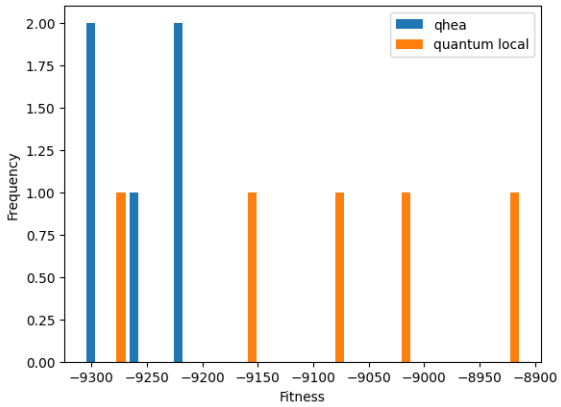


Figure 30: Histogram of QHEA and Quantum Local Optimization

Comparing the statistical data in Table 6, the QHEA has a lower standard derivation of ~ 41 compared to ~ 140 of the Quantum Local Optimization. Also the mean result with ~ -9263 is better than the mean of the local optimization which is -9090.

Solver	Mean	Median	Variance	Standard Derivation
QHEA	-9262.8	-9260.0	1687.7	41.08162606324146
Quantum Local	-9090.6	-9085.0	19811.8	140.75439602371216

Table 6: Statistic analysis of QHEA and the Quantum Local Optimization

To summarize this experiment, the Quantum Local Optimization of QHEA itself is capable of finding solutions that are not too far from the results of the QHEA. Anyways, it has been shown throughout this thesis that the Quantum Local Optimization improves the EA, and this experiment shows that the EA heuristic is also an improvement to the Quantum Local Optimization.

3.3 Comparison of Quantum-Hybrid Evolutionary Algorithm with other solvers

Other solvers for QUBO problems implement, among others, the heuristics Tabu Search and Simulated Annealing. In question is how the elaborated QHEA performs compared with these heuristics. I choose Tabu Search as one subject of comparison because it is a heuristic that is very common in optimization. I compare QHEA with Simulated Annealing because this heuristic is used as a placeholder for a Quantum Annealer. In question is if QHEA is an improvement to Simulated Annealing or if Simulated Annealing is better without the embedding in an EA.

Figure 31 shows the histogram of Tabu Search, QHEA, and Simulated Annealing. The histogram shows that Tabu Search is the strongest of the three heuristics as it has the least scattering and obtains the best objective value to the QUBO problem.

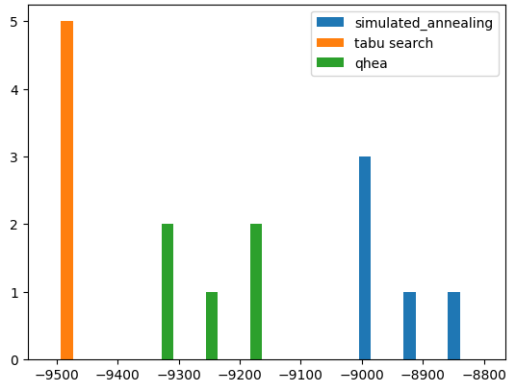


Figure 31: Histogram of different solvers

Solver / Heuristic	Mean	Median	Variance	Standard Derivation
Simulated Annealing	-8919.2	-8945.0	5515.7	74.26775881901916
Tabu Search	-9499.0	-9503.0	380.5	19.50640920313116
Qhea	-9262.8	-9260.0	1687.7	41.08162606324146

Table 7: Statistic analysis of different solvers

Table 7 contains the data of the executions. Tabu Search has the best mean value of -9499, followed by QHEA with a mean of ~ -9363. Simulated Annealing obtained a mean of ~ -8919.

Tabu Search also has the lowest standard derivation, followed by QHEA. The distance between the means of QHEA and Tabu Search is smaller than the distance between the means of QHEA and Simulated Annealing. The same applies to the standard derivation.

3.4 TSP distance

The problem in this experiment is the *kroD100* TSP benchmark instance that was published by the University of Heidelberg [15]. This instance has 100 cities. That is, in the QUBO formulation, a problem with 10000 binary decision variables. The actual optimal distance is 21294 km [16].

Table 8 shows the QUBO objective values that were obtained by the heuristics QHEA, Tabu Search and Simulated Annealing, as well as the distance to the TSP.

All three heuristics obtained a solution to the QUBO that was feasible for the underlying TSP.

Simulated Annealing achieved the worst results with an objective value of ~ -693425 and a distance of ~ 112233 km.

Again, Tabu Search was the best heuristic with an objective value of ~ -762280 and a distance to the TSP of ~ 43378 km.

QHEA was better than Simulated Annealing, but worse than Tabu Search. The obtained objective value was ~ -709332, and the distance to the TSP was ~ 98014 km.

Solver / Heuristic	QUBO Objective	TSP Distance
Simulated Annealing	-693424.9449920654	112233.0997204321
Tabu Search	-762280.4225979146	43377.6190053445
QHEA	-709332.2271763855	98014.41414261505

Table 8: Objective value and TSP distance of different solvers

None of the used heuristics was able to find the actual optimal solution to the TSP, which is not surprising as they have to search a solution space of 2^{10000}.

4 Conclusion

In this thesis, an EA has been developed to optimize a QUBO problem. For this, multiple experiments were carried out for every genetic operator. After evaluating the results, the best combination of genetic operators was selected. Secondly, the EA was extended to a MA with a local optimization technique that is efficient for a Classic Computer. Thirdly, the local optimization technique was replaced by a technique that generates subproblems which are assumed to be efficiently solvable by a Quantum Annealer.

Then, QHEA was compared with the EA with the Classic Local Optimization technique, and also the Quantum Local Optimization technique as a standalone heuristic.

Also, the algorithm was compared using result quality and statistical properties with other solvers available. The benchmark problem used was TSP formulated as a QUBO problem.

Finally, concerning the actual distance of the result to a TSP, the algorithms were compared using the $kroD100$ TSP benchmark instance.

It has been shown that the QHEA was able to find solutions that were better than the EA with Classic Local Optimization, and also that QHEA has better statistical properties than the latter. It has also been shown that the QHEA strongly benefits from the Local Optimization, but the Local Optimization technique is also improved by the EA heuristic.

In comparison to other solvers, the QHEA was compatible in means of result quality and statistical properties, however not better than Tabu Search and probably other heuristics that were not considered in the comparison. Anyways, QHEA was improved by the Simulated Annealing algorithm, but the QHEA was also able to find solution that were better than the results of Simulated Annealing and also had better statistical properties, thus improved Simulated Annealing. Note that this observation is different from the observation from the previous paragraph. Although the Quantum Local Optimization uses Simulated Annealing, it does not optimize the whole QUBO as one problem, but iterates over the solution vector to optimize problems that are small enough to fit on a Quantum Annealer.

Not subject of a direct comparison was the Quantum Local Optimization with Simulated Annealing. Recalling that the Quantum Local Optimization as a standalone heuristic had mean of ~ -9091 and Simulated Annealing a mean of ~ -8919, the Quantum Local Optimization that utilizes Simulated Annealing was better than Simulated Annealing itself in terms of result quality, while the Simulated Annealing hat a better standard derivation of ~ 74 compared to ~ 140. That is, the Quantum Local Optmization improves the Simulated Annealing in terms of result quality but brings in some more scattering, and the EA that embeds the Quantum Local Optimization further improves the solution quality and also "fixes" the higher scattering.

QHEA was capable of finding a feasible solution in a large solution space of 2^{10000}. The distance to the TSP was shorter than the distance returned by the Simulated Annealing algorithm, however almost twice the distance of the result returned by Tabu Search. All algorithms were far off the actual optimal distance of the problem instance.

Not subject of the comparison was the absolute run time of the algorithm. The reason for this is that the algorithm was implemented in Python, which limits the performance compared to an implementation in programming languages like Rust or C++. Also, rather than implementing a high performance algorithm, the question was whether at all the EA heuristic can be expanded by a quantum computer. To solve the QUBO with 10000 variables, the absolute time needed was $\sim$ 35 minutes, compared to $\sim$ 1.5 min that were needed by the Tabu Search and Simulated Annealing.

For the Quantum Local Optimization, a placeholder was used. In the introduction, I made two assumptions regarding the speed and result quality of Quantum Annealing. The first was that Quantum Annealing is at least as good as Simulated Annealing. The second was that, if Quantum Annealing is used, there is negotiable overhead generated by communication, queue time, etc.

Under these perfect conditions, I conclude that EAs can be reasonable extended by Quantum Annealers, as EAs benefit from the Quantum Local Optimization, and QHEA achieves better results than some other heuristics, albeit no other benchmark beside TSP in the QUBO formulation was considered. That is, to generalize the result of this thesis, more benchmark problems are needed.

Assumption 3 was that Quantum Annealer outperforms Classic Computers with growing numbers of decision variables.

An observation was that EAs were not only improved by Simulated Annealing, but also the other way around can improve Simulated Annealing. Recalling that assumption 1 was that Quantum Annealing is at least as good as Simulated Annealing and also referring assumption 3, this observation *might* not be true for Quantum Annealing. EAs will not necessarily improve Quantum Annealing in a situation where Quantum Annealers are capable of solving QUBO of arbitrary size.

Finally, I conclude that Quantum Annealers can be used to improve EAs when solving QUBO problems, while EAs not necessarily improve a Quantum Annealer of arbitrary size. The improvement the EA is to the Quantum Annealer is given by the capacity limitations and will probably shrink with Quantum Hardware improvements, until eventually Quantum Computer are large enough to solve real-world problems without a classic algorithm as a vehicle. However, due to the scalability problems that were mentioned in the introduction, it is not yet clear if this will ever be the case. Unit then, the suggested algorithm is an option to take advantage of Quantum Computing.

References

[1] Dennis Willsch et. al. *Benchmarking Advantage and D-Wave 2000Q quantum annealers with exact cover problems*. arXiv. https://arxiv.org/pdf/2105.02208.pdf. 2022.

[2] Edward Farhi et. al. *A Quantum Approximate Optimization Algorithm*. Massachusetts Institute of Technology. 2014.

[3] Kelly Boothby et. al. *Next-Generation Topology of D-Wave Quantum Processors*. D-Wave. https://www.dwavesys.com. 2019.

[4] Sanjay Jangid et. al. *Evolutionary Algorithms: A Critical Review and its Future Prospects*. International Journal of Trend in Research and Development (IJTRD). ISSN: 2394-9333. 2015.

[5] Thomas Bartz-Beielstein et. al. *Evolutionary Algorithms*. WIREs Data Mining and Knowledge Discovery. https://doi.org/10.1002/widm.1124. 2014.

[6] John N. Tsitsiklis Dimitris Bertsimas. *Introduction to Linear Optimization*. Massachusetts Institute of Technology. 2019.

[7] DWave. *D-Wave Cloud solution*. 2023. URL: https://www.dwavesys.com/.

[8] Anna Kobylinska und Filipe Pereira Martins. *Was ist Quanten-Annealing?* data-insider.de. https://www.datacenter-insider.de/. 2018.

[9] Josef Stoer Florian Jarre. *Optimierung. Einführung in mathematische Theorie und Methoden*. Springer Spektrum. 2019.

[10] Du Glover Kochenberger. *A Tutorial on Formulating and Using QUBO Models*. arXiv. https://doi.org/10.48550/arXiv.1811.11538. 2019.

[11] Thomasz Dominik Gwiazda. *Genetic Algorithms Reference*. Thomasz Gwiazda e-books. http://www.tomaszgwiazda.com/Genetic$_a lgorithms_r eference_f irst_4 0_p ages.pdf$. 2006.

[12] Thomasz Dominik Gwiazda. *Genetic Algorithms Reference*. Thomasz Gwiazda e-books. http://www.tomaszgwiazda.com/Genetic$_a lgorithms_r eference_f irst_4 0_p ages.pdf$. 2006.

[13] Kuk-Hyun Han and Jong-Hwan Kim. *Quantum-Inspired Evolutionary Algorithm for a class of combinatorial optimiyation*. IEEE Transactions on Evolutionary Computation. 2002.

[14] Ahmad Hassanat et al. *Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach*. Information. https://www.mdpi.com/2078-2489/10/12/390. 2019.

[15] University of Heidelberg. *Discrete and Combinatorial Optimization*. [Online; accessed 29-August-2023]. URL: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/XML-TSPLIB/instances/.

[16] University of Heidelberg. *Known Solutions*. [Online; accessed 29-August-2023]. URL: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/STSP.html.

[17] Matthias Homeister. *Quantum Computing verstehen*. Springer Vieweg. ISBN 978-3-658-36433-5. 2022.

[18] Shan Huang et al. *Quantum-accelerated algorithms for generating random primitive polynomials over finite fields*. 2023. arXiv: 2203.12884 [quant-ph].

[19] IBM. *Solving combinatorial optimization problems using QAOA*. IBM. https://learn.qiskit.org/course/applications/solving-combinatorial-optimization-problems-using-qaoa. 2023.

[20] Manuel Lozano, Francisco Herrera, and José Ramón Cano. "Replacement strategies to preserve useful diversity in steady-state genetic algorithms". In: *Information Sciences* 178.23 (2008). Including Special Section: Genetic and Evolutionary Computing, pp. 4421–4433. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2008.07.031. URL: https://www.sciencedirect.com/science/article/pii/S0020025508002867.

[21] C. E. Miller, A. W. Tucker, and R. A. Zemlin. "Integer Programming Formulation of Traveling Salesman Problems". In: *J. ACM* 7.4 (Oct. 1960), pp. 326–329. ISSN: 0004-5411. DOI: 10.1145/321043.321046. URL: https://doi.org/10.1145/321043.321046.

[22] Peter W. Shor. *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*. SIAM J.Sci.Statist.Comput. 26. 1996. URL: https://doi.org/10.1137/S0097539795293172.

[23] Guillermo Alonso-Linaje Sau l Gonz alez Bermejo and Parfait Atchade-Adelomou. *GPS: A new TSP formulation for its generalizations type QUBO*. arXiv. 2021.

[24] Alexander Schrijver. *A Course in Combinatorial Optimization*. CWI. 2006.

[25] Marin Golub Stjepan Picek and Domagoj Jakobovic. *Evaluation of Crossover Operator Performance in Genetic Algorithms with Binary Representation*. International Conference on Intelligent Computing. 2011.

[26] A.J. Umbarkar and P.D. Sheth. *CROSSOVER OPERATORS IN GENETIC ALGORITHMS: A REVIEW*. ICTACT JOURNAL ON SOFT COMPUTING. 2015.

[27] Jean-Paul Watson et al. "The Traveling Salesrep Problem, Edge Assembly Crossover, and 2-opt". In: vol. 1498. Sept. 1998, pp. 823–834. ISBN: 978-3-540-65078-2. DOI: 10.1007/BFb0056924.

[28] Wikipedia. *Problem des Handlungsreisenden — Wikipedia, die freie Enzyklopädie*. [Online; Stand 10. August 2023]. 2023. URL: https://de.wikipedia.org/w/index.php?title=Problem_des_Handlungsreisenden&oldid=232879820.

[29] Wikipedia contributors. *Combinatorial optimization — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-June-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Combinatorial_optimization&oldid=1147380347.

[30] Wikipedia contributors. *Crossover (genetic algorithm) — Wikipedia, The Free Encyclopedia*. [Online; accessed 20-July-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Crossover_(genetic_algorithm)&oldid=1161837084.

[31] Wikipedia contributors. *Evolutionary algorithm — Wikipedia, The Free Encyclopedia*. [Online; accessed 15-August-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Evolutionary_algorithm&oldid=1166508185.

[32] Wikipedia contributors. *Lagrangian relaxation — Wikipedia, The Free Encyclopedia*. [Online; accessed 10-August-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Lagrangian_relaxation&oldid=1149103170.

[33] Wikipedia contributors. *List of quantum processors — Wikipedia, The Free Encyclopedia*. [Online; accessed 16-August-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=List_of_quantum_processors&oldid=1168428670.

[34] Wikipedia contributors. *Mathematical optimization — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Mathematical_optimization&oldid=1167578530. [Online; accessed 6-August-2023]. 2023.

[35] Wikipedia contributors. *Noisy intermediate-scale quantum era — Wikipedia, The Free Encyclopedia*. [Online; accessed 16-August-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Noisy_intermediate-scale_quantum_era&oldid=1169928167.

[36] Wikipedia contributors. *Quantum computing — Wikipedia, The Free Encyclopedia*. [Online; accessed 15-August-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Quantum_computing&oldid=1170316181.

[37] Wikipedia contributors. *Selection (genetic algorithm) — Wikipedia, The Free Encyclopedia*. [Online; accessed 26-August-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Selection_(genetic_algorithm)&oldid=1167055349.

[38] Wikipedia contributors. *Tournament selection — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-July-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Tournament_selection&oldid=1160627612.

[39] Wikipedia contributors. *Travelling salesman problem — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Travelling_salesman_problem&oldid=1166057133. [Online; accessed 8-August-2023]. 2023.

[40] Mitsuo Gen Xinjie Yu. *Introduction to Evolutionary Algorithms*. Springer Science Business Media. ISBN 978-1-84996-128-8. 2010.