Bauhaus Universität Weimar
Faculty of Media
Degree Programme Medieninformatik

Bio-Inspired Algorithms for Combinatorial Optimisation Problems

Metaheuristic Characteristics in Ant Colony Optimisation, Artificial Bee Colony and Firefly Algorithm

Bachelor's Thesis

Karoline Brehm
Born October 3rd, 1997 in Berlin-Spandau

Matriculation Number: 117190

1. Referee: PD Dr. Andreas Jakoby
2. Referee: Prof. Dr. Benno Stein

Submission Date: September 16, 2020

Abstract

Metaheuristics provide an approach to the hard optimization problems researchers and professionals face in almost every field in science and engineering. Nature- and bio-inspired metaheuristics have been successfully used in real-world applications, and there is a great interest in this topic. To take a closer look at the mechanics in such algorithms in this thesis, the Artificial Bee Colony Algorithm, Ant Colony Optimization and Firefly Algorithm were selected as representatives of population-based metaheuristics taking inspiration from swarm intelligence. The algorithms were implemented and analysed in respect to the the components characteristic for metaheuristics, their difficulty, and their performance on instances of the Traveling Salesperson Problem.

Contents

Abbreviations

ACO Ant Colony Optimisation

ABC Artificial Bee Colony Algorithm

CABC Combinatorial Artificial Bee Colony Algorithm

FA Firefly Algorithm

CI Computational Intelligence

SI Swarm Intelligence

CO Combinatorial Optimisation

TSP Travelling Salesperson Problem

Chapter 1

Introduction

In this thesis we will approach the field of *bio-inspired algorithms* by means of analysing the components and performance of three algorithms, which were selected as representatives of this group: the *Ant Colony Optimisation* (ACO), *Firefly Algorithm* (FA) and *Artificial Bee Colony Algorithm* (ABC). These algorithms are *metaheuristics*, i.e. optimization strategies for hard to solve problems.

Approaching this field for the first time, a large number of topics related to metaheuristics and bio-inspired algorithms present themselves, and some terms are used ambiguously or are loosely defined. Research fields are often interdisciplinary or interconnected. Thus, in chapter 2 we provide insight into the topic of metaheuristics and optimization. First, combinatorial optimization problems are discussed, including the *Traveling Salesperson Problem* (TSP), which the algorithms performance was tested on. Secondly, the concept of metaheuristics and related fields of research are described. Chapter 3 then goes into detail about the structure and characteristic components of metaheuristics, as well as some fundamental search metaheuristics. This provides the basis for the understanding and analysis of the ABC, ACO and FA. The methods for the analysis and testing are described in chapter 4. Finally, in chapter 5, the algorithms are described, as well as the analysis of their components, difficulty and performance.

1.1 Motivation and Context

Metaheuristics are used because the problems faced by researchers and professionals in almost every field of science and engineering are not solvable directly. There is a need for some kind of artificial, *computation intelligence* (CI) to produce solutions to these problems. Traditional mathematical methods fail when there is no known function that could be solved, but rather a black box, where there is no principled approach. The scope is too large for a brute force in acceptable time, and there may not ever by information available about what a solution could look like, or where to search for it. Bozorg-Haddad et al. [1] note that the field of computational intelligence has expanded in the last decades to now cover almost every field in science and engineering.

In particular, Yang in [2] states that contemporary optimisation algorithms are

metaheuristic. Special attention is paid by researchers to the group of nature- and bio-inspired algorithms, which is pointed out by state-of-research reviews [3], [4] and [5]. While a trend of publishing only seemingly innovative algorithms is criticised, the success of bio-inspired algorithms in real-world applications and the great interest in the topic justifies a closer look at the mechanics in such algorithms.

Del Ser et al. in [3] in a 2019 literature review on bio-inspired computation give an overview of the state of the art and open challenges in the field as of 2019. The authors state that bio-inspired algorithms pose an approach to solving optimisation problems and provide a tools for solving real world problems in a vast area of fields. An important topic related to bio-inspired computation is *Swarm Intelligence* (SI), which is "a branch of bio-inspired computation based on the emergence of collective intelligence from large populations of agents with simple behavioural patterns for communication and interaction." According to the authors there has been a steady increase in yearly publication numbers on SI for two decades.

These and other strands of algorithms attempt to emulate biological systems and processes to mimic their advances, such as the emergence of fit individuals in evolution. For this purpose, inspiration is taken from nature: From fish and insects to "intelligent water drops". However, this "gold rush" for new strands of algorithms and "exoticism" is critiqued for the lack of scientific value by the Del Ser et al.. It is empathised that a standardized vocabulary and use of a solid design rationale based on theoretical insight are critical for new research.

The 2017 book by Patnaik2017 et al. [6] presenting the state of the art in nature-inspired computing, SI is identified as a tool to solve nonlinear design problems with real-world applications where one faces NP-hard problems, infinite numbers of potential solutions and limited computation resources. The flexibility and versatility, as well as self-learning capability, and adaptability to external variations are listed as selling points of SI algorithms.

Chapter 2

Background

To understand the "metaheuristic characteristics" of the specific algorithms discussed in this thesis, we will begin with a description of combinatorial optimization problems, particularly the TSP, in section 2.1, and discuss how bio-inspired algorithms are related to the field of metaheuristic algorithms in section 2.2.

2.1 Combinatorial Optimization Problems

Many optimization problems aim to find the best configuration of a set of variables to achieve some goal. There is a natural divide into real-valued and discrete problems, depending on how the solutions are encoded.

In [7] and [8] combinatorial optimization problems (COP) are described as discrete problems in which one searches for an object from a finite (or possibly countable infinite) set. This object is typically an integer number, subset, permutation or graph structure.

A general model of a combinatorial optimization problem $P = \{S, \Omega, f\}$ consists of:

- the *search space* S defined over a finite set of discrete decision variables X_i that take values in the variable domains $D_i = \{v_i^1, ..., v_i^n\}$, $i = 1...n$
- a set of *constraints* Ω among the variables
- an *objective function* $f : D_1 \times ... \times D_n \rightarrow \mathbb{R}^+$ to be minimized

A *feasible solution* $s \in S$ is a complete assignment of values to the decision variables X_i that satisfies all constraints in Ω, so the set of all feasible solutions is $S = \{(x_1, v_1), ..., (x_n, v_n) | v_i \in D_i\}$

A solution $s^* \in S$ is called a *global optimum* if and only if $f(s^*) \leq f(s)$ for all $s \in S$. The set of globally optimal solutions $S^* \subset S$.

2.1.1 Traveling Salesperson Problem

The traveling salesperson problem (TSP) is one of the most studied combinatorial problems, and the most famous NP-hard optimization problem. It was defined in

the eighteenth century, and since 1930 has received attention in artificial intelligence, computational mathematics and optimization theory. [9], [10] It is used to model natural applications, and is also a benchmark for optimization methods due to it's simple concept but computational difficulty. For this reason it was used for testing the algorithms in this thesis.

The TSP aims to find a minimal tour for the "salesperson" visiting a number of cities exactly once, starting and finishing at the same location.

Chopard et al. [11] comment that there are many different approaches that have been taken to solve the TSP, one of which is metaheuristic algorithms, and referring to specialized metaheuristics developed for solving large and difficult instances of TSP. In the scope of this thesis, we will therefore use TSP only to benchmark the algorithms, knowing that there exist algorithms much better adapted to this problem.

Many heuristic algorithms for the TSP exist, including the Nearest Neighbour Heuristic [10], the 2-Opt, 3-Opt, k-Opt and Lin-Kerningham heuristics. [6]

The Nearest Neighbour Heuristic is shown in algorithm 1.

Algorithm 1 The Nearest Neighbour Heuristic

```
1: Select random city.
2: while There are unvisited cities do
3:     Add nearest unvisited city to tour.
4: end while
5: Return tour.
```

2.1.2 Solving Combinatorial Optimisation Problems

There are different ways of approaching a CO problem:

Exact algorithms (also called complete algorithms) guarantee to find an optimal solution in bounded time for every finite size instance of the problem. However, if a CO problem is NP-hard (assuming $P \neq NP$) no polynomial time algorithm exists, and computation time might be exponential.

Approximate methods offer a compromise, as they do not guarantee an optimal solution, but might get close enough to satisfy the requirements. Talbi [12] distinguishes approximation algorithms which at least guarantee obtaining a solution within a bound of the optimum, and heuristic algorithms which do not give such a guarantee. It is added that approximation algorithms are more of a tool for studying the difficulty of the problem, and less for practical use, as attainable results are too far from the global optimum.

The third approach is the use of metaheuristics, including bio-inspired algorithms such as those discussed in this thesis. With complete and approximation algorithms not available, or not quick or precise enough, metaheuristics are another approach to finding acceptable solutions: Metaheuristics are approximate algorithms that employ diverse search methods to find reasonably good solutions in reasonable time and are independent of the problem or problem instance. According to Luke in [13], the problems, which metaheuristics are used on, fulfil two criteria. One being that

there is little knowledge on what the optimal solution looks like, and the second that there is no good principled approach. However, a candidate solutions can be assessed in reasonable time. Talbi [12] gives guidelines on determining whether to use a metaheuristic for an application or if better or equal results may be achievable through other methods.

2.2 Metaheuristics and Related Concepts

We have seen in section 2.1 that metaheuristics belong to the group of approximate optimization algorithms. In this section we will further discuss what metaheuristics are, and how they are related to other concepts and fields, such as heuristics and bio-inspired algorithms.

Regarding the difference between heuristics and metaheuristics, Saka et al. [14] conclude that a heuristic "exploits problem dependent information to find a sufficiently good solution to a specific problem, while metaheuristic is a general-purpose algorithm that can be applied to almost any type of optimization problem".

2.2.1 Metaheuristics

Blum and Roli in [7] discuss various definitions of the term metaheuristic, summarizing that metaheuristics are "high-level strategies for exploring search spaces by using different methods". Strategies that may be included are mechanisms to escape local minima, heuristics making use of domain-specific knowledge and the use of search experience to guide the search. Metaheuristics used can range from simple local search procedures to complex learning processes.

Because certain simple metaheuristics are useful for finding local optima but do not sufficiently explore the search space to find global optima, they may be incorporated in complex algorithms to improve exploitation in the area of a candidate solution. Such fundamental strategies are described in section 3.2. They can be seen both as metaheuristic and components of metaheuristics.

Sörensen et al. in [15] outline the history of metaheuristics, noting that the use of high-level strategies is inherent in human problem-solving and that first formal studies on heuristics were done around 1940. They state that the term "metaheuristic" can refer to a problem-independent algorithmic framework (called metaheuristic framework), as well as a problem-specific implementation of an algorithm in accordance with the guidelines in the framework (called metaheuristic algorithm).

Luke [13] states that metaheuristics are a subfield of stochastic optimisation, which is a general class of algorithms and techniques using randomness to a degree to find (near) optimal solutions to hard problems. Contrarily, Talbi in [12] notes that there are stochastic and deterministic metaheuristics, such as tabu search. According to Blum et al. [7] metaheuristics usually employ stochastic elements.

2.2.2 Related Concepts

The algorithms discussed in this thesis belong to the broad field of CI. Kramer in [16] states that CI originated as a branch of artificial intelligence, implementing biologically inspired models algorithmically. This includes artificial neural networks, fuzzy-logic, multi-agent systems and optimisation algorithms.

Furthermore, many metaheuristics, including the algorithms discussed in this thesis belong to the field of nature- and bio-inspired computation. Among those, some are also based on SI.

SI describes the phenomenon of collective behaviour of organisms of a species for carrying out specific tasks (see Yang, Deb et al. in [4]). This has inspired many optimisation methods and in [17] Yang and Karamanoglu give an overview of swarm intelligence based algorithms, which features ABO, ABC and FA. Del Ser et al. in [3] summarize that in bio-inspired computation researchers "emulated intelligent bio-inspired processes in the form of computational algorithms in an attempt to mimic the inherent advances of such biological systems to address complex modelling, simulation and optimization problems". They state that in this field, evolutionary algorithms and swarm intelligence are the most prominent and distinctive approaches. Nature-inspired algorithms incorporate bio-inspired algorithms as well as methods inspired by physical processes and social phenomena.

While the classification of metaheuristics as nature-inspired or non-nature inspired is common, Blum and Roli [7] state that it is "not very meaningful", as the classification is not always precisely possible, and hybrid algorithms might fall under both categories. Nevertheless, it is a widely used category and according to Del Ser et al. in [3] bio-inspired computation it is one of the most studied branches of artificial intelligence during the last decades. However, the need for a unified description and "metaphor-free vocabulary" of bio-inspired algorithms is stressed.

Additional to the mentioned fields of research, metaheuristics are often categorized by their algorithmic characteristics and their components, for example use of memory vs. memoryless search and deterministic vs. stochastic nature. A general divide of metaheuristics can be observed, dependent on the use of either the presence of a single solution (trajectory methods, also called single-point-search or *single-state-methods*) or a set of solutions (*population methods*). [7], [13]

Finally, more generally, Dökeroglu et al. [18] in their recent survey on fourteen distinguished *new-generation metaheuristics* distinguish between classic and new-generation algorithms. It is noted that most state-of-the-art metaheuristics such as Genetic Algorithms, Particle Swarm Optimization, ACO and many more, have been developed before 2000, and that there has been a "new wave of metaheuristic approaches" in the last two decades.

2.2.3 Why are new metaphors explored?

The *No Free Lunch Theorem* states that for certain types of mathematical problems the computational cost of finding a solution averaged over all problems in the class is the same for every solution method. This means that there is no "better than all the others" algorithm, only ones that are more suited to certain problems in that class, and can therefore achieve better results in that specific case for a given finite set of problems and algorithms. [2]

Chapter 3

Metaheuristic Strategies and Algorithm Components

In this section, the characteristics of metaheuristic algorithms will be described. This is the basis for the analysis and understanding of the algorithms discussed later on.

First, the general structure of metaheuristics is described in section 3.1. There exist fundamental search strategies, which in themselves can be considered metaheuristics, but are also used as components of other metaheuristic algorithms. These will be described in section 3.2. Section 3.3 then gives a comprehensive overview of characteristic components.

For this chapter I have referred mainly to the following textbooks and overviews of the topic:

Blum and Roli, Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison [7]

Chopard and Tomassini, An Introduction to Metaheuristics for Optimization [11]

Luke, Essentials of Metaheuristics [13]

Talbi, Metaheuristics from Design to Implementation [12]

Much of the information relevant to this chapter is of course covered in multiple of these books, and is not original to these authors. To improve readability, these above mentioned sources will therefore not be cited in the text every time, but only to highlight differences.

3.1 General Structure of Metaheuristics

There are many different kinds of metaheuristics, and the design of the search processes can differ greatly. Therefore, we will first identify the general structure of metaheuristics, which can be single-solution-based or population-based. While the algorithms discussed in this thesis use populations of solutions, basic search metaheuristics may be incorporated in them to locally improve solutions. Those basic strategies operate on single solutions. Therefore, we will provide general descriptions

for both types.

Secondly, we will discuss some high-level components and their representation, also taking into regard combinatorial optimization problems.

Lastly we will discuss the balancing of the search that must take place in metaheuristics to efficiently find global optima, before we go onto the search methods and components this is achieved by, in section 3.2 and 3.3.

3.1.1 Single-Solution and Population-based Metaheuristics

Generally an optimization problem is defined by a search space S and a objective function f. (See section 2.1 for a more detailed description of combinatorial optimization problems.)

The *search space* S represents the set of *feasible solutions* s.

The *objective function* f formulates the goal to achieve. It associates with each solution $s \in S$ a real value that describes the *quality* of the solution $f : S \to \mathbb{R}$. A solution $s*$ is a *global optimum* if $f(s*) \leq f(s), \forall s \in S$ (assuming a minimization problem. Maximizing f is equivalent to minimizing $-f$)

To find those optimal solutions (or at least reasonably good solutions), a metaheuristic explores the search space S.

It is important to note that in different sources, the terminology may differ. The search space is also called configuration space or state space, solutions are also called configurations, states, individuals or agents, the objective function is also called fitness, cost, evaluation, utility or energy function, and the quality of a solution is also called its fitness or cost. Which term is used may depend on the context, e.g., in evolutionary strategies, the terms fitness and fitness function are commonly used as derived from the biological concept of survival of the fittest.

Additionally, descriptions of bio-inspired algorithms may use the vocabulary of the metaphor, e.g. "bee" instead of "candidate solution" for ABC. However, this practice has received heavy critique, as pointed out by Del Ser et al. [3], who call for a metaphor-free description.

Single-Solution-based Metaheuristics

In single-solution-based methods (also called single point search/ trajectory methods/ single-state methods), there is one solution present at a time and the search process describes a trajectory in the search space. This includes local search metaheuristics, some of which are described in section 3.2.

According to Yang et al. [17] we can describe a generic iterative optimization algorithm as:

$$x^{t+1} = g(x^t, \alpha, OP) \tag{3.1}$$

where x^t is the solution at iteration t. Starting with an initial guess x^0, this will form a piecewise trajectory in the search space, which depends on the parameter α and the problem OP or its objective function $f(x)$.

To explore the search space, a *neighbourhood* $N(x)$ is defined. For a solution $x \in S$ its neighbourhood $N(x)$ is the set of solutions y that one can reach from x in one step and includes x itself. The neighbourhood is specified implicitly by defining a set of transformations T_i that generate y from x. $y = T_i(x)$, $y \in N(x)$. In the search process, the next solution x_{t+1} is chosen from the neighbourhood so that the process becomes: $x_0 \to x_1 \in N(x_0) \to x_2 \in N(x_1) \to \ldots$.

The transformation and selection strategy depend on the metaheuristic and generally make use of the quality of neighbour solutions and incorporate stochastic elements.

A high-level template for single-solution-based metaheuristics is shown in algorithm 2, as given by Talbi et al. [12].

Algorithm 2 Single-Solution-based Metaheuristic

1: Initialize solution x_0
2: **while** Stopping Criteria not satisfied **do**
3: Generate set of candidate solutions $C(s)$ (partial or complete neighbourhood) from the current solution s_t
4: Select a solution s_{t+1} from $C(s)$
5: **end while**
6: Return current or best solution found

Population-based Metaheuristics

In contrast to single-solution-based methods, in population-based metaheuristics a set of solutions (often called candidate solutions) is present, which is called *population* (or swarm/sample). The search process describes an iterative improvement of this set. The present candidate solutions affect how other solutions evolve, which distinguishes population-based from parallel single-solution methods.

For population-based metaheuristics, Yang et al. [17] describes the search process as:

$$[x_1, \ldots, x_n]^{t+1} = g([x_1, \ldots, x_n]^t, [\alpha_1, \ldots, \alpha_k]^t, [\rho_1, \ldots, \rho_m]^t, OP) \tag{3.2}$$

where the population size is n and the algorithm depends on k different algorithm-dependent parameters α_i and m different random variables ρ_i. At each iteration, n solutions are derived from the previous state of the population. We also write $P_t = [x_1, \ldots, x_n]^t$ for the population at iteration t.

A high-level template for population-based metaheuristics is shown in algorithm 3, as given by Talbi et al. [12].

Algorithm 3 Population-based Metaheuristic

1: Initialize population P_0
2: **while** Stopping Criteria not satisfied **do**
3: Generate population of new candidate solutions P'_t
4: Select new population P_{t+1} from $(P_t \cup P'_t)$
5: **end while**

Population-based metaheuristics can be differentiated according to their method of manipulating the current population to generate the next generation:

Luke [13] distinguishes *resampling techniques* as used by evolutionary algorithms and "directed mutation methods" such as in Particle Swarm Optimization. The latter modify the population, but no resampling occurs.

Talbi [12] distinguishes *evolution and blackboard strategies*. In evolution strategies the solutions are reproduced using variation operators, like mutation and recombination on the solutions' representation. The next generation is therefore constructed from the different attributes of solutions belonging to the current population. Different selection strategies for the choice of solutions for the next generation exist, such as elitist selection, which picks the best from the union of current and newly generated solutions. In blackboard strategies the population constructs a shared memory, which is the main input for generating the new population. The search memory can be of the form of a probabilistic model, such as the pheromone matrix in ACO.

The group of bio-inspired metaheuristics based on swarm intelligence, which we will focus on in the scope of this thesis, is population-based and generate new populations of solutions by directed mutation and blackboard methods.

3.1.2 Search Space, Objective Function and Neighbourhood

In this section we will go into more detail about the search space and objective function in respect to metaheuristics, and the concept of *neighbourhoods* in the search space.

Search Space

Chopard and Tomassino [11] describe the qualities of the search space S of an optimization problem.

A solution $x \in S$ is often a quantity specified by N degrees of freedom, represented as N-dimensional vector $x = (x_1, x_2, \ldots, x_N)$. As described in section 2.1, in combinatorial optimization problems this is typically an integer number, subset, permutation or graph structure.

N defines the *size* of the optimization problem, but not the size of the search space: $|S|$ equals the number of solutions that the search space contains.

In combinatorial optimization problems with solutions x_i from the discrete countable space, the search space is finite, but $|S|$ may be exponential to the N degrees of freedom.

An important space for combinatorial optimization problems is the permutation space of n objects. For a general $n \leq 1$ the number of permutations is $n! = n(n-1)(n-2)...1$. The size of such a search space grows exponentially with n.

A *permutation* of a set of n elements a_i, $i = 1...n$ is a linear order of the elements. It can be represented in different ways, such as a list or dictionary data structure: In a list $(a_{i_1}, a_{i_2}, ..., a_{i_n})$ where i_k is the index of the element at place k, positions

are first-order quantities. In a dictionary data structure we indicate for each object at which position it will occur.

The transformation operators must be defined adjusted to the representation. A swap operator (i, j) for example, which swaps object i and j in a permutation, produces different outcomes on the list vs. the dictionary, and it is necessary to choose an operator that is suitable.

Objective Function

The objective function f of an optimization problem associates with each solution of the search space a real value that describes the quality of the solution $f : S \to R$. The f may or may not be equal to the problem formulation.

In many routing problems such as TSP and vehicle routing problems, the formulated objective is to minimize a given global distance. For instance in the TSP, f corresponds to the total distance of the Hamiltonian tour:

$$f(s) = \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)} \tag{3.3}$$

π represents a permutation encoding of a tour and n the number of cities.

This means, the objective function to optimize by the metaheuristic is equal to the objective function specified in the problem formulation. Such objective functions are called *self-sufficient* objective functions.

Neighbourhood

A neighbourhood function $N(s)$ is a mapping $N : S \to 2^S$ that assigns to each solution $s \in S$ a set of solutions $N(s) \subset S$. A solution s' in the neighbourhood of s, i.e. $s' \in N(s)$), is called a neighbour of s.

A solution $s \in S$ is called a local optimum if it has a better quality than all its neighbours, ie.$f(s) \leq f(s')$ for all $s' \in N(s)$.

The neighbourhood is specified implicitly by defining a set of transformations T_i that generate y from x. $y = T_i(x)$, $y \in N(x)$. This means, a neighbour is generated by the application of a transformation operator m that performs a small perturbation to the solution s.

The neighbourhood definition is strongly dependent on the representation of the problem. The effect of a transformation applied to a solutions representation is called *locality*. In the case of strong locality, a small change to the representation ideally should correspond to a small change in the solution. Extremely weak locality will result in the search converging toward random search, as there is no meaningful relation between neighbouring solutions that could be exploited by the algorithm.

3.1.3 Diversification and Intensification

Another important high-level concept is *diversification* and *intensification*.

Except from some very basic search strategies, all metaheuristics incorporate some method of designing the search process in a way that promotes finding the global optimum. This requires two things: Intensively exploring areas of high quality solutions and exploring unvisited areas of the search space. The concepts related to these goals are called intensification and diversification.

The terms *exploitation* for intensification, and *exploration* for diversification are often used synonymously. Sörensen et al. [15] refer to Glover and Laguna [19] who state that exploration and exploitation are terms from control theory, while diversification and intensification are concepts first described in tabu search.

According to Glover and Laguna, [19] (qtd. in [15]), exploration and exploitation refers to following a particular (generally memoryless) rule until it fails in the exploitation phase, then making random changes in the exploration phase, before returning to the rule. In contrast, diversification and intensification are strategies of guiding the search.

Blum [7] states that exploitation and exploration often refer to short term strategies using randomness, while intensification and diversification are long term strategies and make use of the search history.

Sörensen et al. [15] note that the "emphasis on incorporating strategies to achieve the goals of intensification and diversification" underlies most modern metaheuristics and while seeming natural and inevitable today, in the 1980s "contrasted with perspectives that prevailed [...] which relied chiefly on randomization and gave little attention to the relevance of memory (and hence learning) within metaheuristics".

Intensification strategies reinforce attributes of high quality solutions found in the past. In a neighbourhood context, this means focusing on the region where such solutions have been found. [15] According to Yang [17], [20] intensification strategies promote convergence by finding better solutions in the local area.

In contrast, diversification strategies generate solutions that incorporate attributes which have not been incorporated in past solutions. From a neighbourhood perspective, the search is focused on diverging into new regions. [15] According to Yang [17], [20], this means generating diverse solutions, which increases the probability of finding a globally optimal solution.

Yang [17], [20] states that an unbalanced search that leans too much into either diversification or intensification, can respectively lead to either premature convergence to a local optimum and possibly insuffcient solution, or a slowed down search process. Diversification components "attempt to thoroughly investigate the decision space and divert the solutions to be trapped in local optima", while the intensification components "ensures the convergence of the algorithm to what is more likely to be the global optimum solution." It is stated that global optimality can usually be ensured through a good combination of these components.

As in this description, diversification and intensification are sometimes seen as opposing forces. However, as Blum et al. note[7], algorithm components may have an diversifying effect, an intensifying effect, or both. This put the notion of "opposing

forces" into question. Blum et al. [7] propose a framework for intensification and diversification, in which they are considered the effects of algorithm components: The framework positions components depending on their use of randomness, objective function and one or multiple other functions as a guide for the search. Memory may be incorporated by one of the "other" functions as well. Not using the objective function indicates pure diversification, whereas exclusively using the objective function indicates pure intensification.

3.2 Fundamental Search Metaheuristics

In this section we will discuss a number of important fundamental search strategies, which can be considered metaheuristics. They are relevant because they may be incorporated in other, more complex, metaheuristics.

Local search, in context of bio-inspired algorithms, refers to strategies such as those described in the following. These may be used to improve upon a solution like a single-solution metaheuristic, i.e. not considering the rest of the population.

The term local search is ambiguously used, and in other sources may also refer to:

- a simple metaheuristic method called hill climbing
- modified versions of hill climbing
- single-solution metaheuristics

3.2.1 Accepting Negative Moves and Multi-Start

These following two strategies are simple concepts found in the local search algorithms.

Lones [21] identifies *accepting negative moves* as a basic metaheuristic strategy. It prevents convergence to local optima, as moves to solutions worse than the current one are allowed sometimes.

A local search may be enhanced with a technique called *multi-start*, where the search process is "restarted in a different region once it has converged at a local optimum." [21]

3.2.2 Hill Climbing

According to Talbi et al. [12], *hill climbing* is likely the oldest and simplest metaheuristic method. Lones [21] summarizes the concept: "Follow a sequence of local improvements in order to find a locally optimal solution. A single move is performed at each step. If this leads to a better solution, the algorithm then moves on to explore a variant of this new solution, otherwise it remains at the original point and considers a different move."

Algorithm 4 Hill-Climbing Metaheuristic

```
1: Initialize solution x
2: while time left and ideal solution not found do
3:    Generate candidate neighbour x'
4:    if Quality(x') > Quality(x) then
5:       replace x with x'
6:    end if
7: end while
8: Return x
```

Steepest Ascent / Descent

In the *steepest ascent/descent* variant of hill climbing, a number of neighbours is considered at each iteration, and the best option is picked for the move.

Algorithm 5 Steepest Ascent Hill-Climbing Metaheuristic

```
1: Initialize solution x
2: while time left and ideal solution not found do
3:    Generate candidate neighbour x'
4:    for n-1 times do
5:       Generate candidate neighbour y
6:       if Quality(y) > Quality(x') then
7:          replace x' with y
8:       end if
9:    end for
10:   if Quality(x') > Quality(x) then
11:      replace x with x'
12:   end if
13: end while
14: Return x
```

Probabilistic Hill Climbing

In *propabilistic hill climbing*, randomized moves are introduced by selecting the the next solution in the neighbourhood with a probability $p(x')$.

$$p(x') = \frac{f(x')}{\sum_{y \in N(x)} f(y)} \tag{3.4}$$

for positive f(x), so that the probability of x' being selected is proportional to its fitness, instead of performing a steepest ascent/descent. The higher the quality of a candidate solution, the more likely it is to be selected. [11].

3.2.3 Metropolis Algorithm and Simulated Annealing

Simulated Annealing (see algorithm 7) is a metaheuristic which has had a major impact on the field for its simplicity and efficiency when it was first employed for optimization problems in the 80s. Annealing is a technique in which a substance is heated and slowly cooled down to achieve a strong structure, as in metallurgy. Its basis comes from the *Metropolis Algorithm* (see algorithm 6), a model for simulation of energy changes in a system at a temperature, which was proposed by Metropolis et al. in 1953, based on principles of statistical mechanics. Due to this historical background of physics, the term energy E, energy function and system state x (instead of quality and solution) are used.

Both algorithms always accept a better solution for the next step, similar to steepest ascent. However, a negative move may be accepted with a probability p. p is determined by the "energy variation" $\Delta E = E_{x'} - E_x$ and a control parameter T called temperature, following the Boltzmann distribution:

$$P(\Delta E, T) = e\frac{-\Delta E}{T} \tag{3.5}$$

The larger the difference in energy (quality), the smaller the probability of accepting a worse solution. Given a ΔE, the likelihood of accepting a worsening move increases with a high temperature. This can prevent the search from getting stuck in local optima.

In the Metropolis Algorithm, a constant k and temperature T are fixed.

Algorithm 6 Metropolis Algorithm

```
1: Initialize solution x
2: Choose constants k and T
3: while Stopping criteria unsatisfied do
4:    Generate random neighbour x'
5:    ΔE = f(x') − f(x)
6:    if ΔE ≤ 0 then
7:       replace x with x'
8:    else
9:       with probability e^(−ΔE/kT) replace x with x'
10:   end if
11: end while
12: Return x
```

In Simulated Annealing a *cooling schedule* $g(T)$ is used to update the temperature parameter and progressively lower it to achieve convergence. There exist different strategies and extensions to the basic simulated annealing. To give an example, the temperature may be lowered in stages, where at each temperature level the process is allowed to converge towards a value of energy, until the solution cannot be improved upon any further.

Algorithm 7 Simulated Annealing

Initialize solution x
Set starting temperature T_{max}
while Stopping criteria unsatisfied **do**
  while Equilibrium condition unsatisfied **do**
    Generate random neighbour x'
    $\Delta E = f(x') - f(x)$
    if $\Delta E \leq 0$ **then**
      replace x with x'
    else
      with probability $e^{\frac{-\Delta E}{kT}}$ replace x with x'
    end if
  end while
  Update temperature $T = g(T)$
end while
Return best solution found

3.3 Components of Metaheuristic Algorithms

We have seen the general structure of metaheuristics, and some basic metaheuristic algorithms and strategies. In this section we will now give a comprehensive overview of other components found in metaheuristic algorithms.

3.3.1 Initialisation

In population-based metaheuristics, the search begins with an initial population. Talbi [12] states that this population is crucial in the effectiveness and efficiency of the search. Different initialization strategies exist, from uniformly and randomly sampling the search space, to using a diversification strategy to ensure a good distribution of samples, or using a heuristic. It is noted that the initialization in itself can be seen as a possibly difficult optimization problem of finding the optimum of a "diversification criterion", which would promote finding the global optimum.

3.3.2 Randomisation

Yang et al. [17] state that randomization is an efficient component for global search algorithms. This is contained in the fundamental search metaheuristics described section 3.2, in the form of random restarts and the use of a probability distribution for selection of the next solution. The same concepts are used in complex metaheuristics as well.

As described in section 3.1.3, diversification is associated with randomization, and may be part of different components. It is therefore relevant to identify components that introduce randomness.

Formally, a random walk can be written as modifying an existing solution x_n at step n by a perturbation w_n, so that $x_{n+1} = x_n + w_n$ where w_n is a vector of random numbers drawn from a known probability distribution. Using the Gaussian distribution leads to a behaviour called normal diffusion where the average distance moved is proportional to $\sqrt{n}$. Another distribution is the Lévy distribution. Lévy flights are used in several bio-inspired metaheuristics for continuous problems to improve the exploration of the search space. [2]

3.3.3 Selection

The selection of solutions forming the next population is a concept prevalent in algorithms based on evolution. A selection strategy may be used to pick high quality solutions, ensuring convergence, while maintaining some level of diversification, for example by including some random or lower quality solutions.

3.3.4 Memory

Blum [7] notes that the use of memory is "one of the fundamental elements of a powerful metaheuristics". It is differentiated between short term and long term memory. *Short term memory* includes tracking recently performed moves, visited solutions and decisions taken. A common example is the "tabu list" of Tabu search, which in its simplest form is a Steepest Ascent strategy where the tabu list of previously visited solutions is used to prohibit moves to them, which allows escaping from local minima. *Long term memory* is "accumulation of synthetic parameters" about the search.

Lones [21] identifies the following two basic metaheuristic strategies that leverage information gathered about the search space during the search:

Adaptive Memory Programming is a generalization of the ideas of tabu search. Past search experience is used to guide future search. A FIFO tabu list is the simplest form, but complex variants have been developed.

Search Space Mapping refers to the concept of constructing a map to guide the search process. This can take the form of a probabilistic overlay (also called probabilistic model) that points towards productive regions, as in Ant Colony Optimization.

Not all metaheuristic use memory however. For example, simulated annealing (see section 3.2.3) is a memory-less method as no information extracted from the search is used to guide the search. Only the best solution found overall is remembered and returned as the result of the search. Still, simulated annealing is used successfully.

3.3.5 Stopping Criteria

A component not introduced yet is the end of the search. Different *stopping criteria* exist which control when to stop the iterative optimization process. Talbi [12] differentiates between *static* and *adaptive* procedures, where the end of the search is known or not known beforehand respectively. Static procedures may use a number of iterations, number of objective function evaluations or limit of CPU resource to determine when to stop. Adaptive procedures may use a number of iterations without improvement, or reaching an optimum or satisfactory solution if such a property can be observed.

Chapter 4

Methods

To gain insight into bio-inspired metaheuristics the Firefly Algorithm (FA), Artificial Bee Colony (ABC) metaheuristic and Ant Colony Optimisation (ACO) metaheuristics were chosen for analysis and testing. Section 4.1 discusses the evaluation metrics used for the analysis of the algorithms' components, performance as well as their difficulty. Section 4.2 describes the experimental design.

4.1 Evaluation Metrics

4.1.1 Analysis of Algorithm Structure and Components

The algorithms will be analysed and compared according to the components typical of metaheuristics and bio-inspired algorithms described in section 3.3, particularly considering the following aspects:

- The population of candidate solutions and its structure and development
- Strategies for the generation of new candidate solutions
- Selection strategies
- Stochastic components
- Diversifying and intensifying components
- Memory

4.1.2 Performance

The performance analysis of metaheuristics is a difficult task. This is due to random choices which lead to varying results in each run and parameters that control the search, but for which no single correct value can be determined. The books by Talbi [12] and Chopard and Tomassini [11] provide details on the experimental design for metaheuristics.

In the following we describe the evaluation metrics used, why they are used and how they are measured.

To assess the *algorithm performance*, we consider the relation of the quality of the solution found by the algorithm to the resources used. Additionally we consider algorithm behaviour, such as convergence of the solutions and indicators for exploitation vs. exploration behaviour, which may allow drawing conclusions about the algorithm performance.

The solution quality is measured by recording the quality development of solutions produced by the algorithm per iteration, as well as the development of the best solution so far seen in the search. Additionally, it is recorded at which iteration the final solution was first found, as this can indicate how quickly the algorithm converges: If an algorithm produces bad quality solutions at an early iteration, this could indicate an imbalance in the search, where there is a lack of exploration to counteract the convergence towards local optima. Respectively, finding high quality results at an early iteration indicates an efficient algorithm.

Next to the solution quality, it is of course important how long the optimization takes. There are different options for measuring the resources needed by an algorithm. One is to measure the number of calls to the objective function, so for TSP calculating the length of a given tour. This however allows only limited insight when considering three quite different algorithms, as the objective function is not necessarily the most time-consuming task. This was suspected from the algorithms' structure and found true in preliminary tests. Therefore, only the overall runtime was taken into consideration.

Compared to an exploratory approach, where time constrains are lesser of a concern, the implementation and hardware is crucial for the runtime in an application. Optimisation of the implementation, choice of programming language and parallelisation may all be used to improve runtime, but will not be considered in this thesis.

In the tests performed for this thesis, the number of iterations was limited to 2000 per run because there is no guarantee on the error of the solution the algorithm will return. It is therefore necessary to limit the number of iterations, as the algorithm may get stuck in a local optimum above the desired error and run forever. (See also section 3.3.5 about stopping criteria.) In preliminary tests a limit of 2000 iterations for the maximum number of iterations seemed to allow enough steps for decent convergence within reasonable time.

In the analysis we consider the following metrics for performance:

- Quality of results
  - Relative error after the maximum number of iterations
- Reliability of the algorithm
  - Success rate, defined as the number of runs where a solution with an error $\leq 7.5\%$ (determined through preliminary tests) is found within the maximum number of iterations
- Convergence

- Iteration at which the best solution was found and its quality
- Number of Iterations after which a certain quality is reached

4.1.3 Difficulty

The "difficulty" of the metaheuristics will be considered, as it impacts how quickly an algorithm can be implemented and in the correct way intended by its creators. Even if an existing library is used, understanding the algorithm is crucial for working with it. Namely, being able to deliberately adjust parameters, choose an appropriate algorithm for a problem at hand and possibly add or adjust steps of the optimisation to improve performance. This is a subjective criterion, and we will use the following key questions to assess the ease of understanding and use:

- In the paper describing the algorithm variant to be implemented, was a "metaphor-free" description given? (see section 3.1.1 for critique on using the vocabulary of the biological inspiration for the metaheuristic)
- In the paper describing the algorithm variant to be implemented, were the structure and components unambiguously defined and their purpose explained?
- Does the algorithm variant for TSP honour the concept of the original algorithm?
- Are the components of the algorithm complex or simple?

Interesting is also the number of parameters, as a large number of parameters is more difficult to tune. Appropriate values need to be found, as they control the search. With a small number of parameters, it may also be easier to recognize how important a parameter is to the search behaviour though testing.

4.2 Testing of the Algorithms

4.2.1 Implementation

The instances used to test the algorithms are from the "TSPLIB" library of TSP instances, as this library is widely used for benchmarking. It is freely available to reproduce tests and instances as well as optimal tours provided. A detailed description is provided by the documentation, which can be found on the project website, as well as more information and references to other TSP instance libraries. [22]

The algorithms were implemented in Python 3. The "TSPLIB95" library for working with the "TSPLIB95" file format was used. [23]

The tour determined by the nearest-neighbour heuristic (see 1) was used for the initial population in FA and ABC, and for initial update of the pheromone matrix in ACO. This approach is adapted from the paper by Karaboga and Gorkemli [24],

for fair comparison of two variants of the combinatorial ABC algorithm, to allow all algorithm to start from a common initial solution. However, other initialisation strategies exist which may improve performance (see section 3.3.1).

Random numbers play an important role in metaheuristic algorithms. In the implementation, the "default random number generator" provided in the "numpy.random" module was used, which uses the "PCG64" generator. The seed is generated from entropy of the operating system by the generator. [25] [26]

4.2.2 Parameter Settings

The parameter tuning of a metaheuristic algorithm is in itself an optimisation problem, where no setting will be optimal for all instances. As a result, which parameter setting can likely achieve good results cannot be answered easily, but only through extensive testing with a variety of problem instances of different sizes.

Therefore, for the experimental comparison of algorithms in this case, the parameter settings were adopted from the respective papers describing the specific algorithm. In those papers, parameter tests were performed and well-performing parameter settings are proposed.

4.2.3 Experiment Setup

To measure and compare the performance of metaheuristic algorithms, it is necessary to perform multiple optimisation runs for a configuration of instance and parameter setting. This is necessary due to the random choices in the algorithm which will lead to varying results. Additionally, instances of different sizes and topology need to be tested. Algorithms may scale differently well and any two algorithms may be not be suited equally well to a certain problem, as stated by the Free Lunch Theorem (see section 2.2.3).

For the tests performed, ten runs were performed for each combination of algorithm and instance, for the following four instances of symmetric TSP shown in table 4.1 (a total of 120 runs).

Table 4.1: TSP instances for algorithm testing.

Problem	Number of Cities	Optimal Tour Length
eil51	51	426
eil76	76	538
kroA100	100	21282
gr202	202	40160

These are instances of the symmetric TSP for which optimal tours and their lengths are known and provided in the "TSPLIB" library. This allows computation of the error of solutions and also the visual comparison of the best tour found by the algorithm to the optimal tour. The instances were chosen for a regular increase in the dimension from 51 to 76 to 100 cities, plus one instance of higher dimension.

Chapter 5

Analysis of Three Bio-Inspired Metaheuristics

In this chapter, we will take a closer look at the three bio-inspired metaheuristics. We chose ACO, ABC and FA, as they are are good examples in the group of population-based bio-inspired algorithms. ACO is arguably the first algorithm of this kind to adopt swarm intelligence and is widely used and studied. In the "new generation" ABC and FA stand out as they have received most attention by researchers (see Dökeroglu et al. [18], Patnaik [6], Hussain et al. [5])

In the first part of this chapter, the ACO, ABC and FA are described (sections 5.1, 5.2, 5.3), considering their inspiration from nature and the specific algorithm variant implemented for this experiment. In the second part, the algorithms' components and performance are analysed and compared according to the metrics defined in section 4 (sections 5.4, 5.5, 5.6).

5.1 Artificial Bee Colony

The "Artificial Bee Colony Algorithm" (ABC) was proposed in 2005 by Karaboga [27] for "solving unimodal and multi-modal numerical optimization problems". Please note that in the following, we will use the term ABC to refer to the metaheuristic, not to the numerical optimization algorithm.

The *Combinatorial Artificial Bee Colony Algorithm* (CABC) is a variant for TSP published in 2011 by Karaboga and Gorkemli [28]. It was further developed into the *quick Combinatorial Artificial Bee Colony Algorithm* (qCBC) in 2013 [29] and a comparison of CABC and qCABC can be found in the 2019 paper by the same authors [24].

Another approach to adapting ABC to the TSP is presented by Kiran [30]. A comprehensive survey of ABC was published by Karaboga et al. [31].

For this thesis the CABC was implemented and tested.

5.1.1 Inspiration from Nature

The ABC mimicks the foraging behaviour of honey bees in a swarm. Anguluri et al. in [32] state that ABC "has received huge attention from both practitioners and researchers on intelligent optimization". However, it is not the only or first metaheuristic to take inspiration from these insects. The authors identify different aspects of honey bee behaviour and social order that have inspired algorithms, from division of labour in the bee hive, their spatial memory and navigation, to foraging and communication.

In nature, a swarm of honey bees is divided into the female queen laying eggs, male drones which mate with a queen during the mating flight, and female non-mating worker bees which make up 98% of the colony. The latter exhibit the collective intelligence inspiring optimization: Worker bees forage for nectar from flowers, and both observe other bees to decide which flowers to visit next and communicate the location of food sources to other bees in the hive via "waggle dance", which is a figure eight dance behaviour that signals distance and direction. This foraging behaviour with communication is the inspiration for the ABC. [32]

5.1.2 The Artificial Bee Colony Metaheuristic

ABC considers three groups of artificial foraging bees: *employed bees*, *onlooker bees* and *scout bees*. Employed bees search for good food sources, which are communicated to the onlooker bees. Onlooker bees then select food sources probabilistically depending on their quality. Lastly, if a food source is abandoned, its employed bee becomes a scout bee searching for food sources randomly.

Those artificial bees do not represent a population of candidate solutions (see section 3.1.1 for the structure of population-based metaheuristics). The food sources represent candidate solutions and their nectar contents represent the solutions quality. While each employed bee technically is associated with exactly one food source and turn into scout bees and then back into employed bees, what is actually modeled is a population of food sources which are iteratively improved upon by the three different mechanisms represented by each type of foraging bee. The number of onlooker bees in referenced papers was generally chose equal to the number of food sources.

Lones in [21] gives a very concice description of the core concept of ABC without the obfuscating vocabulary of the bee metaphor: "[ABC] explores a fixed number of regions within the search space at any one time. The number of search processes in each region is determined by relative fitness. Search continues in a region whilst fitter solutions continue to be found; otherwise, a new region is randomly sampled."

The basic structure of ABC is shown in algorithm 8. In the employed bee phase and onlooker bee phase, a new solution is generated. If the quality of the solution is better, the old food source is replaced with the new one. The onlooker bees select a food source according to a probability distribution. An important parameter for ABC is the number of attempt at improving a solution before abandoning it, which is called "limit".

Algorithm 8 Artificial Bee Colony Algorithm

1: Initialize food sources randomly.
2: **while** Stopping criteria unsatisfied **do**
3: *Employed Bee Phase* Employed Bees search the food sources' neighbourhoods for better food sources.
4: *Onlooker Bee Phase* Onlooker Bees probabilistically select food sources according to quality and search their neighbourhoods for better food sources.
5: *Scout Bee Phase* If no better solutions can be found in the neighbourhood of a food source after a number of attempts in the employed bee and onlooker bee phase, it is abandoned and a scout bee randomly selects a new food source.
6: Remember the current best solution.
7: **end while**
8: Return best solution found.

5.1.3 The Combinatorial Artificial Bee Colony Algorithm

In the following, the details for CABC are described.

The probability function for the selection of a food source i by onlooker bees is:

$$p_i = \frac{0.9 * fit_i}{fit_{best}} + 0.1 \tag{5.1}$$

The fitness fit_i of each tour is given by:

$$fit_i = \frac{1}{1 + f_i} \tag{5.2}$$

where f_i is the length of the solution that the food source represents.

The limit l is calculated by:

$$l = \frac{n * d}{L} \tag{5.3}$$

where L is a control parameter, n is the population size (number of food sources) and d is the dimension of the problem instance.

To adapt the ABC to the TSP, an appropriate neighbourhood search mechanism is needed. In the CABC the Greedy Sub Tour Mutation (GSTM) is used, which is a successful mutation operator proposed by Albayrak and Allahverdi [33] for Genetic Algorithms for TSP. The algorithm is given in algorithm 9

The GSTM is controled by the following parameters:

Table 5.1: Parameters of Greedy Sub Tour Mutation

P_{RC}	Reconnection Probability
$P_{P_{CP}}$	Correction and Perturbation Probability
P_L	Linearity Probability
L_{MIN}	Minimum Sub Tour Length
L_{MAX}	Maximum Sub Tour Length
NL_{MAX}	Size of the Neighbourhood List

Algorithm 9 Greedy Sub Tour Mutation

1: Given is a tour. Let *random* be a random value uniformly distributed over the intervall [0,1].
2: Select a random segment of length $l \in [L_{MAX}, NL_{MAX}]$.
3: **if** $random \leq P_{RC}$ **then**
4: {*Reconnection*}
5: Compute quality of all possible reconnection mutations.
6: Return the best mutation.
7: **else**
8: **if** $random \leq P_{CP}$ **then**
9: **while** the open tour is not empty **do**
10: **if** $random \leq P_L$ **then**
11: {*Perturbation*}
12: Select city by rolling.
13: **else**
14: Select city by mixing.
15: **end if**
16: Add city to mutated segment and remove from open tour.
17: **end while**
18: Replace selected segment in the original tour with the mutated segment and return.
19: **else**
20: {*Correction*}
21: Greedily select rotation for startpoint of segment and update tour.
22: In the updated tour, greedily select rotation for the endpoint of the segment and update tour.
23: Return mutated tour.
24: **end if**
25: **end if**

A segment of the tour is randomly selected, and the edge between its outer neighbours is added to the tour, such that a *closed tour and open tour* are created (more precicely a cycle and a path). Probabilistically, a tour is mutated by either the *reconnection*, *correction* or *perturbation* scheme, all of which create a new tour from these two trails.

The reconnection scheme inserts the open tour between adjacent cities such that the tour length is minimal.

The perturbation scheme reinserts the mutatated open tour in the original position. The mutated open tour is built by incrementally adding cities from the open tour by probabilistically performing either *rolling* or *mixing* for each step, depending on P_L. Rolling adds a random city to the mutatated tour, then removing it from the open tour. Mixing adds the last city in the open tour to the mutated tour, then removing it. In this way, the linearity probability controls whether the mutated open tour will be resemble the inversion or the random permutation of the segment more closely.

In the correction scheme the neighbourhood lists of the cities are used, which include the NL_{MAX} closest neighbours. For an endpoint R of the segment and neighbours

N in its neighbourhood list, a rotation of the segment with the rotation edge (R, N) is greedily selected. This replaces the edges $(R, R+1)$, $(N, N+1)$ with (R, N), $(R+1, N+1)$, where $R+1$, $N+1$ are the successors of R and N in the tour. (Given a path $v_1, v_2...v_i, v_{i+1}...v_k$ the rotation of a path is defined as the path $v_1, v_2...v_i, v_k...v_{i+1}$ for rotation edge (v_k, v_i). [34])

The gain of a point R G_R can therefore be computed by the equation 5.4, and the first rotation with $G_R > 0$ is selected. The search is restricted to the neighbourhood list and there may not be such a rotation.

$$G_R = d(R, R+1) + d(N, N+1) - d(R, N) + d(R+1, N+1) \tag{5.4}$$

where $d(x, y)$ is the distance between two cities.

The pseudocode is for the CABC given in algorithm 10.

Algorithm 10 Combinatorial Artificial Bee Colony

```
Initialize population of food sources (candidate solutions).
Compute Neighbourhood Lists.
Remember the best solution.
while Stopping criteria not satisfied do
    {Employed Bee Phase}
    for each employed bee do
        Generate a new candidate solution in the neighbourhood of the associated
        solution using GSTM. Replace if the new solution is better.
    end for
    {Onlooker Bee Phase}
    for each onlooker bee do
        Select a tour according to the distribution of the probability values p_i.
        Generate a new candidate solution in the neighbourhood of the associated
        solution using GSTM. Replace if the new solution is better.
    end for
    {Scout Bee Phase}
    For all candidate solutions which could not be improved within l attempts,
    replace it with a random tour.
    Remember the current best solution.
end while
Return best solution found.
```

5.2 Ant Colony Optimisation

Ant Colony Optimisation is an umbrella term for different metaheuristic algorithms inspired by the foraging behaviour. The idea was originally proposed in 1992 by Dorigo [35] for solving the TSP and later formalized into a metaheuristic by Dorigo, Di Caro and Gambardella in 1999 [36]. Many variants of ABC exist, including the original Ant System, the MIN-MAX Ant System and the Ant Colony System which are stated to be the two most successful variants by Dorigo et al. in 2006 [8]. The Ant Colony System was introduced in 1997 by Dorigo and Gambardella [37] and will be the focus of the following discussion.

5.2.1 Inspiration from Nature

Ants use pheromones as a mean of indirect communication: As they forage for food, they leave a chemical trail on the ground. This accumulates when numerous ants use the path, and evaporates with time if few ants use the path. Consequently, trails that are used by many ants are more likely to be chosen by other ants and this results in efficient transport of food to the nest. This concept is known as stigmergy: The communication between the individual ants is self-organized, indirect and non-symbolic, and the information is local, on the ground the ant walks on. [8], [38]

5.2.2 Ant Colony System

The basic stucture of ACS is shown in algorithm 11.

Algorithm 11 The Ant Colony System

1: Set parameters, initialize pheromone trails
2: **while** Stopping criteria unsatisfied **do**
3: Construct Ant solutions by incrementally adding cities to the solution according to the state transition rule and appling the local pheromone update rule.
4: (Apply local search.)
5: Apply global pheromone update.
6: **end while**
7: Return best solution found.

In every iteration, ants construct solutions to the TSP by starting from a random city and incrementally adding cities to a partial solution s^p according to the so-called pseudorandom proportional rule. This allows control over the degree of greedyness in the choice: either a greedy selection or a selection according to a probability distribution is applied.

The evaluation of the possible choices depends on the pheromone τ associated with the edge from the latest city to cities in its neighbourhood $N(s^p)$, and the heuristic information η which is defined as:

$$\eta_{ij} = \frac{1}{d_{ij}} \tag{5.5}$$

where d_{ij} is the distance between cities.

A pseudorandom proportional rule is used, where q is a random variable uniformly distributed over $[0, 1]$, and q_0 is the control parameter.

If $q \leq q_0$, the best candidate is chosen according to the following equation:

$$j = \arg\max_{c_{il} \in N(s^p)} \tau_{il} * {\eta_{il}}^{\beta} \tag{5.6}$$

Else, a stochastic selection mechanism is used to determine the next city for an ant k with the latest city i:

$$p_{ij} = \begin{cases} \frac{\tau_{ij} * {\eta_{ij}}^{\beta}}{\sum_{c_{il} \in N(s^p)} \tau_{ij} * {\eta_{ij}}^{\beta}} & \text{if } c_{ij} \in N(s^p) \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

The pheromone associated with the edge between two cities τ_{ij} is initialized with a value τ_0, and pheromone is deposited by ants during the construction (local/online pheromone update). Additionally, pheromone is deposited for the best tour found in the iteration (global/offline pheromone update). Some pheromone also evaporates when it is updated. The amount of deposited pheromone depends on the quality of the partial solution.

The local pheromone update function is:

$$\tau_{ij} = (1 - \varphi) * \tau_{ij} + \varphi * \tau_0 \tag{5.8}$$

where $\varphi \in (0, 1]$ is the pheromone decay coefficient. The local update causes some diversification, as subsequent ants are less likely to choose the same edge.

For the global pheromone update the "best tour" is the best tour found either in the iteration or so far in the search.

$\Delta\tau_{ij}$ is the pheromone amount deposited by an ant:

$$\Delta\tau_{ij} = \frac{1}{L_{best}}. \tag{5.9}$$

The pheromone value is update accoring to the following rule:

$$\tau_{ij} = \begin{cases} (1 - \rho) * \tau_{ij} + \rho * \Delta\tau ij & \text{if the edge (i,j) belongs to the best tour,} \\ \tau_{ij} & \text{otherwise} \end{cases} \tag{5.10}$$

where ρ is the evaporation rate.

Lastly, applying some local search to improve upon the construted solutions is optional but commonly used.

5.3 Firefly Algorithm

The Firefly Algorithm (FA) was first proposed by Yang in 2007 for continuous optimization problems. Descriptions can be found in the following sources by Yang (et al.): [2], [17], [20], [39].

Fister et al. [40] provide a comprehensive review of FA and its variants and applications. The Discrete Firefly Algorithm adapted is a variant adapted to the TSP proposed by Jati et al. in 2013 [9]. This is the variant implemented for this thesis.

5.3.1 Inspiration from Nature

The FA takes inspiration from bioluminescent Fireflies which flash their light as a "courthip signal", attracting mates. In most species, males and females communicate via flash signal patterns, with females preferring brighter flashing males.

From this behaviour, some basic rules are extracted:

The FA is based on the flashing patterns and behaviour of fireflies, from which some idealized rules are extracted:

- All fireflies are unisex and have the same behaviour.
- The light intensity of a firefly is determined by the quality of the candidate solution it represents.
- The attractiveness of a firefly is proportional to its perceived brightness, which decreases with distance.
- A firefly will either move into the direction of a brighter firefly it perceives as most attractive, or move randomly if there is no firefly more attractive than itself.

5.3.2 The Firefly Algorithm Metaheuristic

The basic structure of FA is shown in algorithm 12.

Algorithm 12 Firefly Algorithm

```
1: Initialize population of fireflies.
2: while Stopping criteria unsatisfied do
3:     Compute the brightness of each firefly.
4:     For each firefly compute the attractiveness of all other fireflies and move it
       towards a brighter firefly or randomly.
5:     Remember the current best solution.
6: end while
7: Return the best solution.
```

Light intensity and attractiveness control the movement of fireflies. The light intensity of a fireflies depends on the quality of the candidate solution it represents and is absolute. The attractiveness lies in the eye of the beholder, as it depends on the light intensity of the other firefly and the distance between the fireflies.

The light intensity is usually defined as proportional to the objective function:

$$I(x) \propto f(x) \tag{5.11}$$

The light intensity perceived by the onlooker $I(r)$ varies with the distance r monotonically and exponentially:

$$I(r) = I_0 e^{-\gamma r} \tag{5.12}$$

where γ is the light absorption coefficient I_0 the light intensity of x_i. This way, γ controls the visibilty of the fireflies. For $\gamma \to 0$, visibility is independent of the distance, so that all fireflies can be seen. For $\gamma \to \infty$ visibility is close to zero, so that other fireflies cannot be seen.

From this the function for attractiveness $\beta(r)$ is derived:

$$\beta(r) = \beta_0 e^{-\gamma r} \tag{5.13}$$

where β_0 ist the attractiveness at $r = 0$.

The distance r between two fireflies is determined by some distance metric, dependent on the optimization problem.

5.3.3 The Discrete Firefly Algorithm for Traveling Salesperson Problem

In the following the details for the discrete FA for TSP as proposed by Jati in [9] are described. The algorithm is the same as described in algorithm 12.

The distance $d(i, j)$ between two fireflies is defined as:

$$d(i, j) = \frac{10 * A}{n^2} \tag{5.14}$$

where A is the number of different edges and d is the dimension of the problem instance. This scales the distance to the interval [0, 10].

The parameter β_0 in equation 5.13 is defined as the light intensity of the other fireflies.

Fireflies move either towards another firefly according to an edge-based movement scheme proposed by the author, or randomly.

To move one firefly closer to another, the total number of different edges must be reduced. The edge-based movement scheme ensures that edges which are contained in both tours are not removed, and one edge of the attractor is added to the attracted firefly, thus decreasing the distance between the two. The steps of the movement scheme are given in algorithm 13.

Algorithm 13 Movement Scheme

1: Given are fireflies i and j, where j is the attractor.
2: Randomly select an edge (x,y) from the tour of j.
3: For cities x and y, find the segments S_x, S_y around the respective city in the tour of firefly i, which contains only edges that are also part of the tour of j. Such segments expend only in only direction as (x,y) is not contained in the tour of i.
4: Randomly choose one of four ways of merging the segments such that x and y are adjacent in the new tour: Insert S_x before or after S_y, S_y before or after S_x. This may require inverting a segment.
5: Return the new tour.

To move randomly, inversion mutation is used in m different positions generating m new solutions which are added to the population. The m worst solutions in the population are then removed such that the population size stays the same. Inversion mutation simply inverts a segment of random length $l \in [2, d]$, replacing two edges of the tour.

5.4 Algorithm Components

In this section the metaheuristic components found in the ABC, FA and ACO and their expression in the implemented algorithm variants are identified and compared.

The initialization and stopping criterion were simple random initialization and a maximum number of iterations in all three algorithms. As described in 4.2, in the testing the nearest neighbour heuristic was used to enable starting from the same initial solution in the test runs.

Overall, the characteristic components of metaheuristics can be found in different forms in all three algorithms. Thus, understanding the general structure and components described in 3 did provide a good basis for working with these algorithms and assessing differences and similarities between them.

Table 5.2: Comparison of metaheuristic components

Component	**ABC**	**ACO**	**FA**
Search Strategy	Employed bees exploit food sources and communicate quality to onlooker bees, onlooker bees choose food sources to exploit according to their quality, scout bees randomly explore the search space	Ants construct solutions according to pheromone associated with edges	Fireflies move towards attractive other fireflies or randomly
Development of Solutions	Iterative improvement; Random reinitialization	Construction of new solutions in every iteration	Movement towards attractive solutions; Random movement
Neighbourhood	The GSTM Operator produces solutions in different neighbourhoods, using the neighbourhood lists for the correction step.	All unvisited cities are considered during the construction process	All solutions in the population are considered by each firefly
Generation of new Solutions	Greedy Sub Tour Mutation Operator which performs reconnection, correction or perturbation (evolution strategy)	Construction of solutions in every iteration, guided by pheromone and heuristic information (blackboard strategy)	Edge-based movement scheme, guided by quality of solutions in the population; Inversion mutation for random movement (evolution strategy)
Selection Strategies	–	–	Removal of worst solutions in the population in the random mutation step

Continued on next page

Table 5.2 – continued from previous page

Component	ABC	ACO	FA
Diversifying Components	Scout Bee Phase; GSTM Perturbation	Local pheromone update	Random movement of fireflies
Intensifying Components	GSTM Reconnection and Correction	Global pheromone update	Movement of fireflies towards better solutions
Balancing the Search	Probabilistic choice of onlooker bees	Probabilistic selection of cities in construction of solution	Attractiveness measure dependent on distance
Local Search	GSTM Reconnection and Correction	Optional additional step	–
Memory	Remembering of the best solution seen so far (does not guide the search)	Pheromone represents accumulated knowledge about the search space, guiding the search (Search Space Mapping)	Remembering of the best solution seen so far (does not guide the search)
Guidance of Search through "Communication"	Probabilistic Choice of onlooker bees depending on quality, mimicking the communication of bees by "waggle dance")	Indirect Communication though pheromones (Stigmergy)	Light intensity of Fireflies
Randomness	GSTM Operator; Random reinitialization by scout bees	Pseudorandom Proportional Rule; Probabilistic choice of cities in construction	Random movements; Random choices in the edge-based movement

5.5 Algorithm Difficulty

The papers referenced for the ABC, ACO and FA variants implemented in this thesis all provided descriptions using both the terminology of the metaphor and of optimization and metaheuristics. I.e. for FA, candidate solutions were generally called fireflies in the description, and the tour is also referred to as *chromosome*. This however did not impact the comprehensibility. In contrast, the description of the ABC was more difficult to understand. Candidate solutions are represented by food sources, but often the algorithm is explained in a way that suggests that the employed bees, onlooker bees and scout bees represent candidate solutions. This is not the case, and in fact the types of bee actually represent the steps in the manipulation of the population. Consequently, it is unclear whether the values given for population size refer to the number of food sources or number of bees. Lastly, ACO too is described in terms of the metaphor as well, but this does not prohibit the understanding of the algorithm.

Overall, while completely "metaphor-free" descriptions were not used, this generally did not impact the understanding, unless the description did not correspond to the structure as described in section 3.1, as was the case with ABC.

The general structure of all three algorithms is fairly simple, and complexity arises only in the details. ABC and FA iteratively mutate the population. The ABC relies on the GSTM for the generation of new solutions, which is a much more complex operator than the edge-based movement and inversion mutation used in the FA. In comparison, ACO has a different structure, where the population of solutions is newly constructed in every iteration, based on the pheromone and heuristic information. Here, the difficulty arises from understanding the global and local update function.

As discussed in 4, the tuning of the search parameters is a crucial step in applying the algorithms. Two parameters that are variable for all algorithms are the population size and maximum number of iterations, i.e. the stopping criterion.

ABC has a large number of parameters controlling the search: hte number of onlooker bees, limit l as well as six parameters controlling the GSTM. ACO relies on the initial pheromone value τ_0, influence of the heuristic information β, pheromone decay coefficientΦ and evaporation rate ρ. In contrast, FA relies only on the light absorption coefficient γ and number of option generated for random movement m.

Consequently, FA is easiest to tune, and the impact the parameters have on the search is very clear, whereas in ABC and ACO, the meaning of the parameters is not as straightforward.

Overall, this meant that the FA was most easy to understand, and thus to correctly implement.

5.6 Results of Performance Testing

The results of the performance testing are shown in this section.

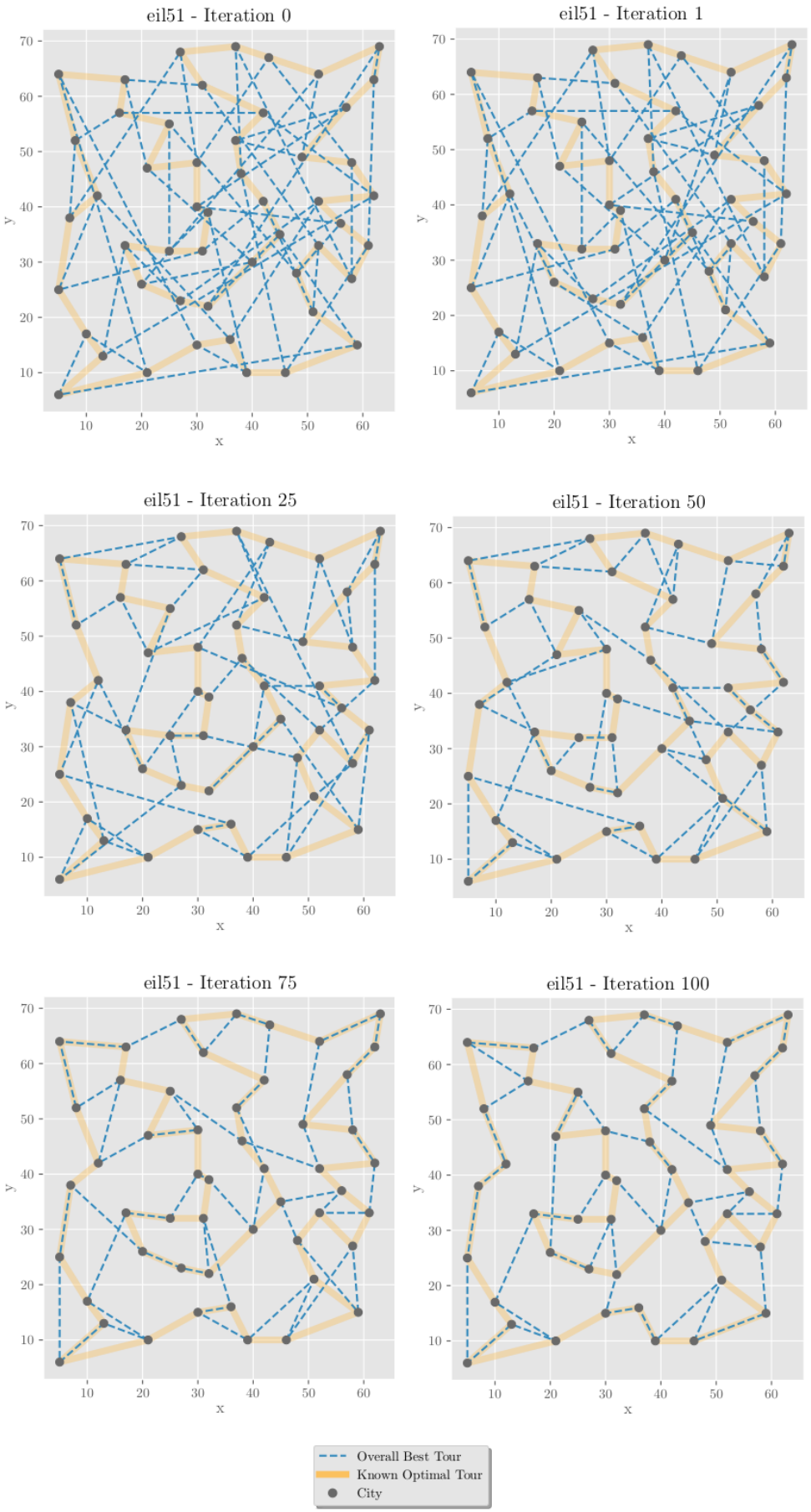
The following outcomes of the ten runs per problem instance are shown in the respective tables 5.3 (ABC), 5.4 (ACO) and 5.5 (FA), where:

- L_O is the length of the optimal solution.
- $\min(L_F)$ is the length of the best solution found in all runs.
- The error is given as the mean relative error in % rounded to one decimal place.
- $It(L_F)$ is the average earliest iteration at which the best solution was found in a run.
- The success rate is the number of runs where the relative error $\leq 7.5\%$, divided by the total number of runs.
- The runtime per iteration is calculated by (Average Runtime/2000).

For each algorithm, the graphs for the convergence and development of the error are shown, as well as an example run on the eil51 problem to compare the development of the solutions per iteration and overall.

The effect of convergence is shown in figure 5.1 which illustrates the development of the optimal solution in an example run of the FA on the eil51 instance. This visual representation was very useful for gaining insight into the behaviour of optimization algorithms.

Figure 5.1: Results of an exemplary 100 iteration run of FA on the eil51 instance



5.6.1 Artificial Bee Colony

The ABC did not perform well in the tests: Its runtime was extremely long, between one and fourteen hours depending on the problem size, and for instance sizes larger than 50 cities, the relative error of the results was between 17 and 40 % on average. Even eil51, only half the runs found solutions within the 7.5% error margin. The convergence (see figure 5.2) for eil51, eil76 and kroA100 til around iteration 250, then slows down. For gr202, the solutions converge slowly, and it seems that more iterations would be needed to further improve the results. The slower convergence seems to be due to the relatively good initial solution (around 50% error compared to around 800% for kroA100), which can be seen in figure 5.4.

We can see in figure 5.3 that the iteration best and overall best solutions do not differ, as would be expected. Per iteration, small changes to the population are applied and the solutions are overall improved.

Table 5.3: ABC Results

Instance	L_O	$\min(L_F)$	Error in %	$It(L_F)$	Sucess Rate	Average Runtime	Runtime per Iteration in s
eil51	426	447	7	1840	0.5	00:52:13	1.6
eil76	538	617	17.7	1924	0	02:08:14	3.8
kroA100	21282	28020	39.3	1980	0	03:04:00	5.5
gr202	40160	51082	28.9	1857	0	14:02:20	25.3

Figure 5.2: Convergence

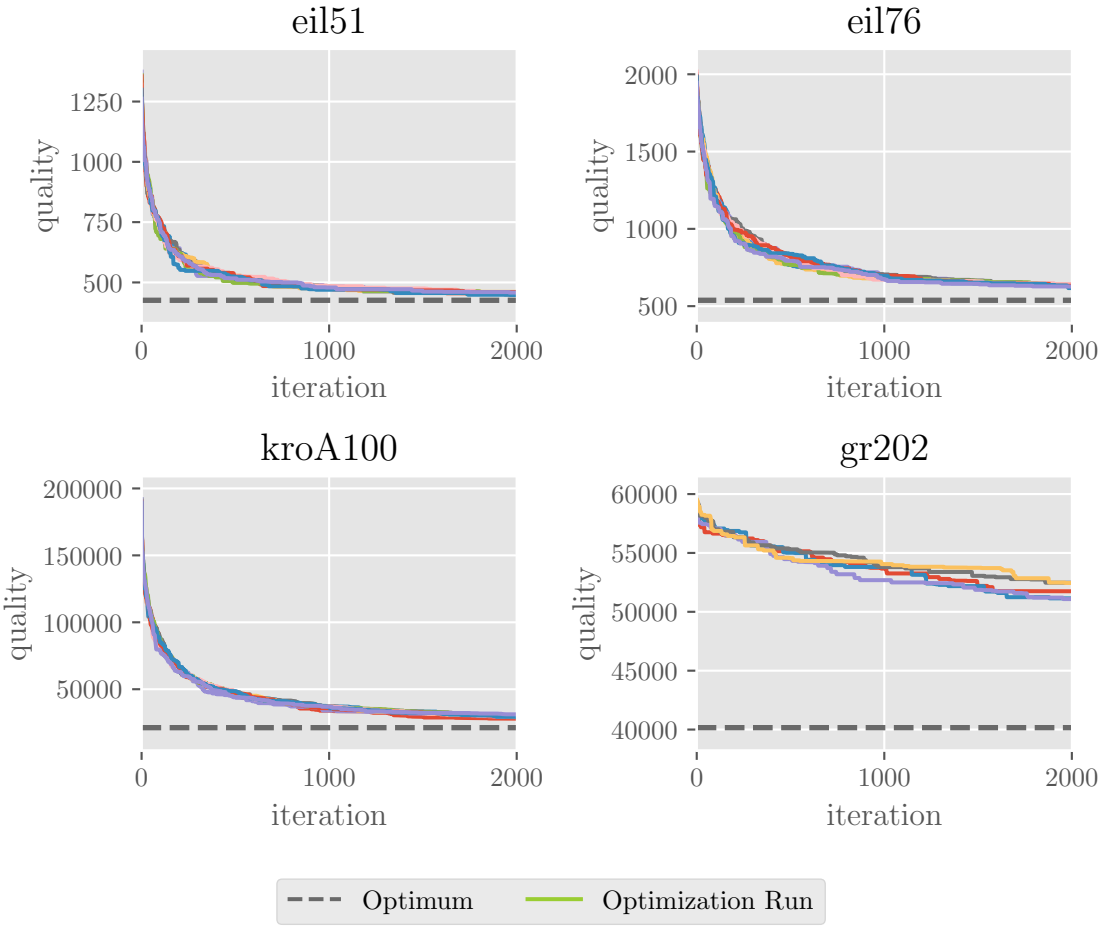


Figure 5.3: Development of iteration best and overall best solution

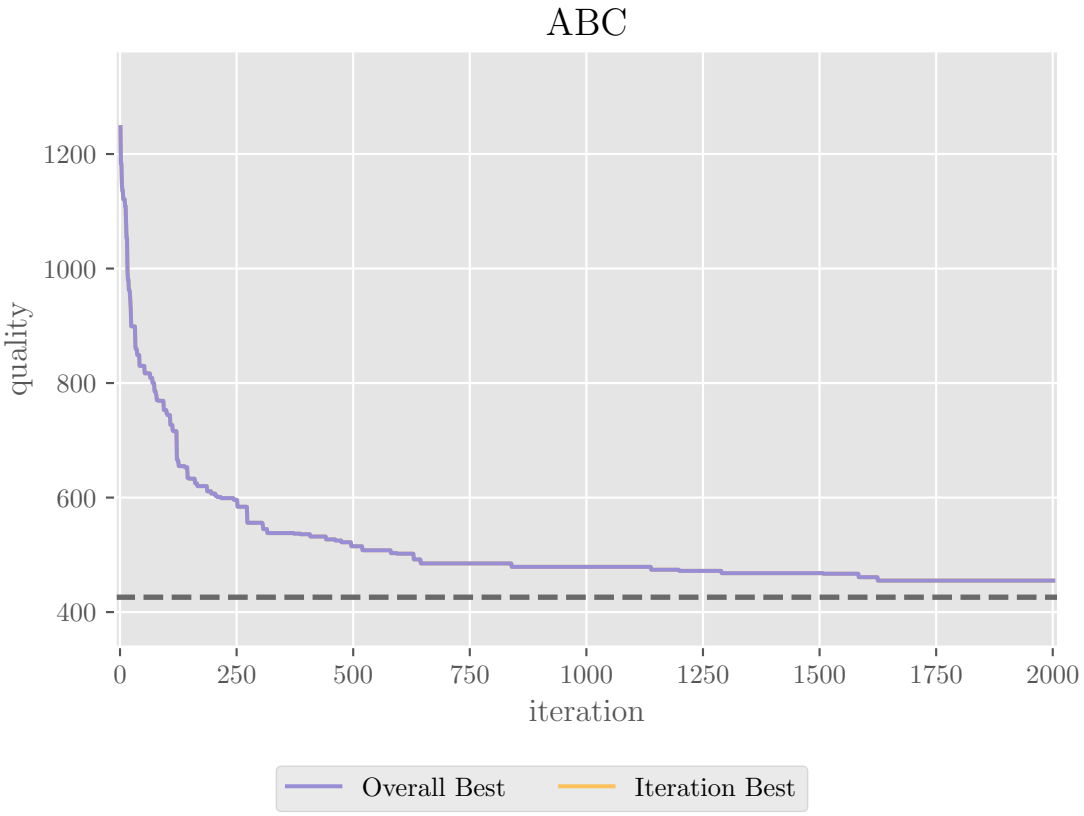
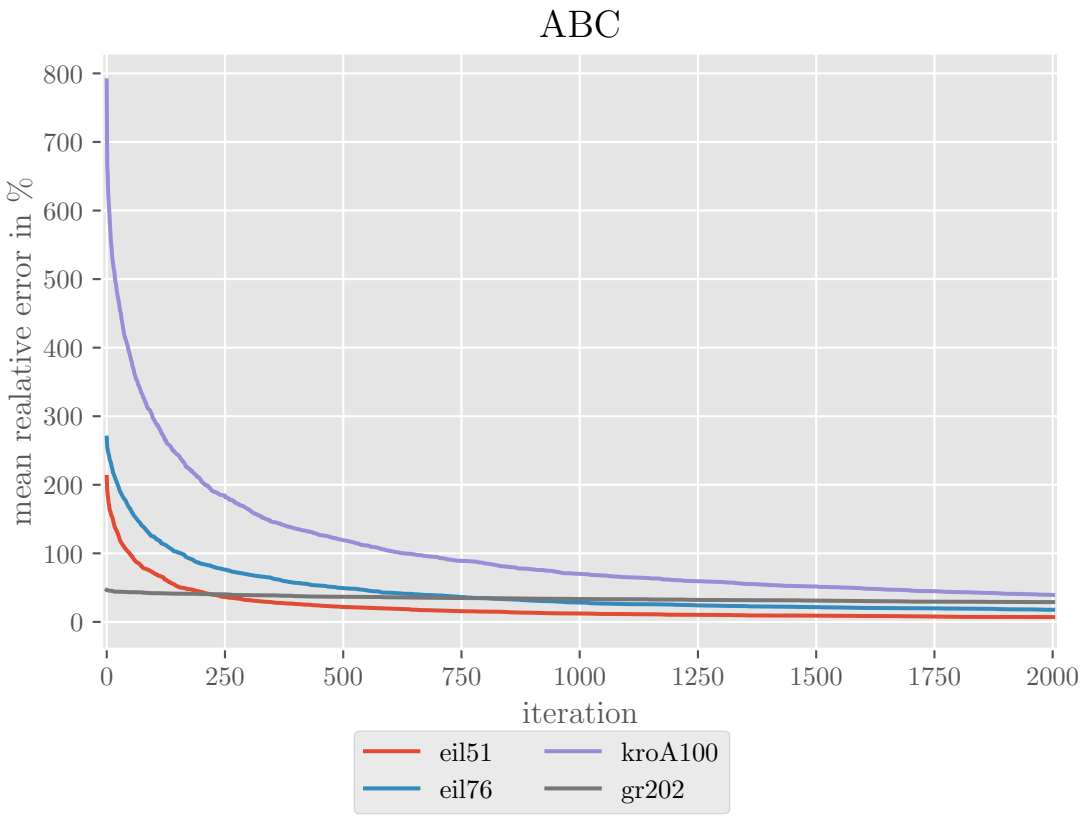


Figure 5.4: Mean error by problem instance



5.6.2 Ant Colony Optimization

The ACO did not provide sufficient any results within the 7.5% error margin. The mean relative error barely improves, as can se seen in figure 5.7. In figure 5.6, we can see how the best solutions per iteration do not closely follow the overall best, and it seems that better solutions are only found by chance, not thanks to the pheromone guided construction of solutions.

This may be due to unsuitable parameter settings, causing a lack of intesification in the search, or a fault in the implementation.

Table 5.4: ACO Results

Instance	L_O	$\min(L_F)$	Error in %	$It(L_F)$	Sucess Rate	Average Run-time	Runtime per Iteration in s
eil51	426	1218	197.8	1211	0	00:11:28	0.3
eil76	538	1853	254.5	1154	0	00:27:10	0.8
kroA100	21282	159258	685.6	918	0	00:31:25	0.9
gr202	40160	58388	54.7	448	0	02:25:45	4.4

Figure 5.5: Convergence

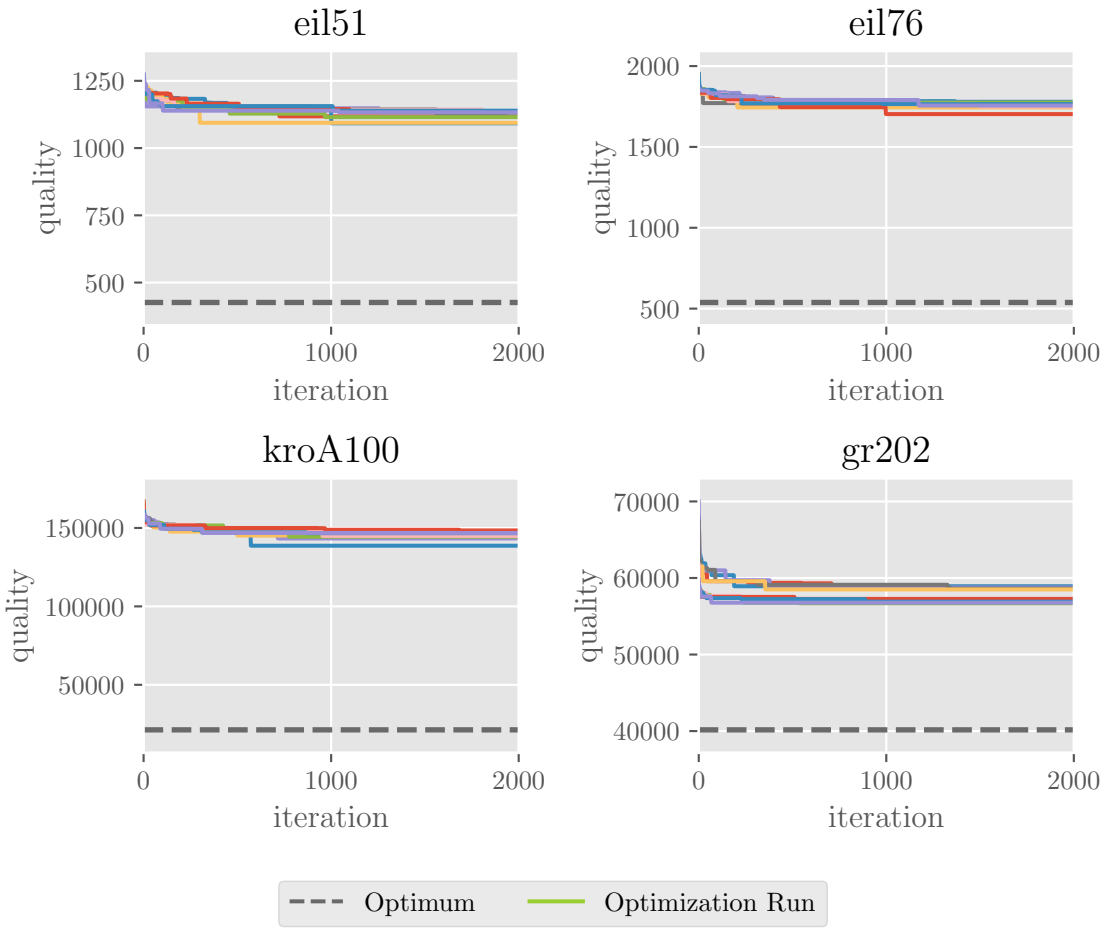


Figure 5.6: Development of iteration best and overall best solution

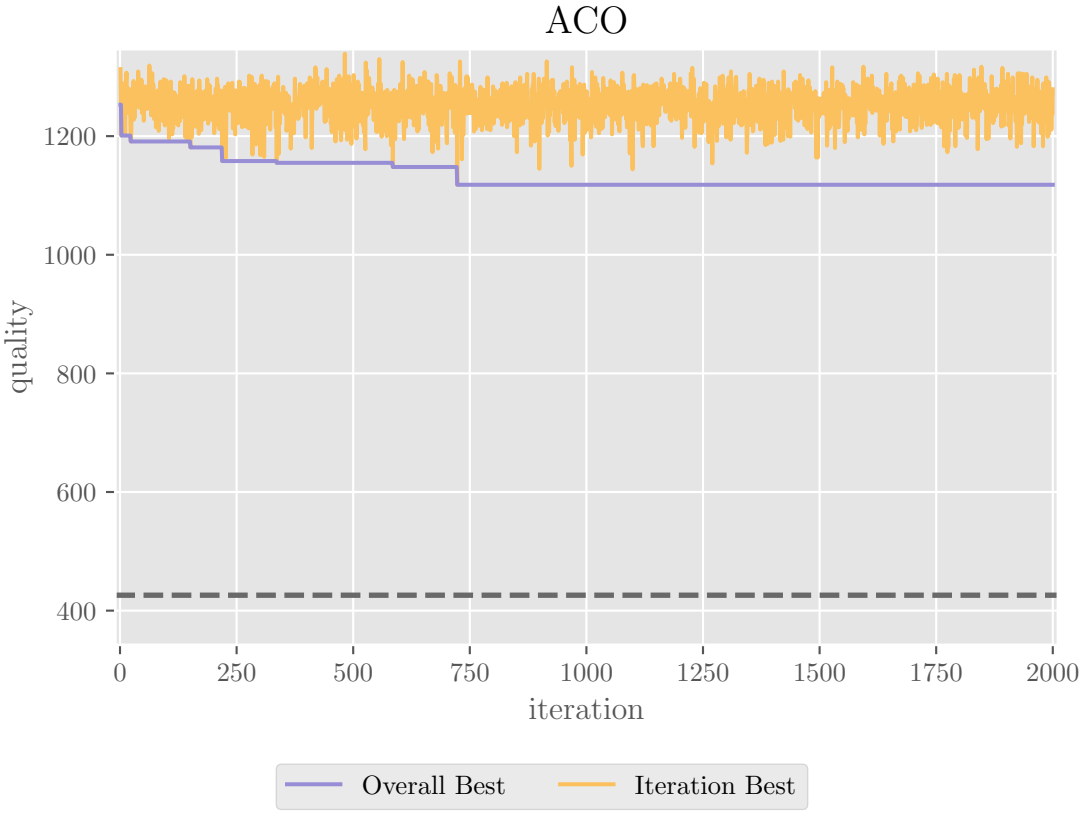
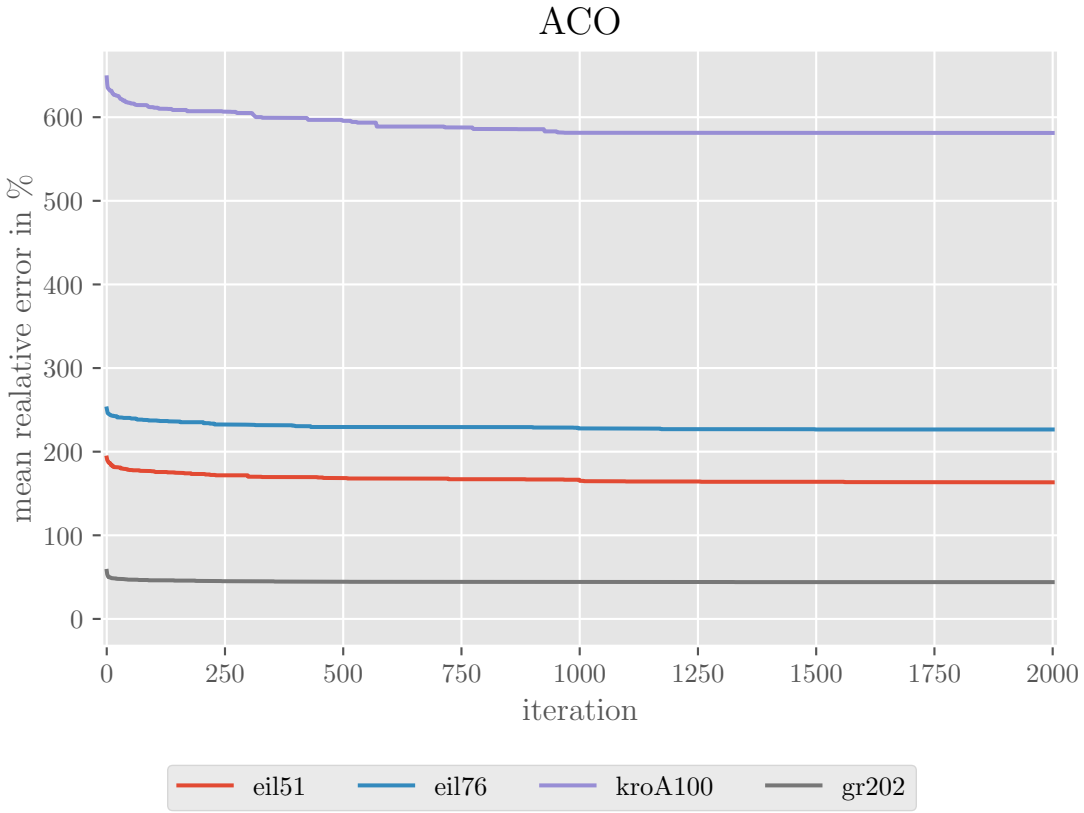


Figure 5.7: Mean error by problem instance



5.6.3 Firefly Algorithm

The FA provided good results in relatively short time. Solutions with a relative error between 7.4% and 10.3% were found in under 5 minutes for instances of up to 100 cities. In figure 5.8 we can see that the solutions converge quickly for about the first half of the search and then more slowly as they approach the optimum. In about 18 minutes, solutions with an average of 8% error were found for gr202. For this problem, again, the initial solution is relatively good, so the convergence is much slower throughout. The success rate was between 20% and 80%.

Table 5.5: FA Results

Instance	L_O	$\min(L_F)$	Error in %	$It(L_F)$	Sucess Rate	Average Runtime	Runtime per Iteration in s
eil51	426	444	7.4	182	0.8	00:03:24	0.1
eil76	538	559	9.6	419	0.2	00:04:20	0.1
kroA100	21282	22505	10.3	1024	0.2	00:04:54	0.1
gr202	40160	42523	8	1812	0.4	00:17:32	0.5

Figure 5.8: Convergence

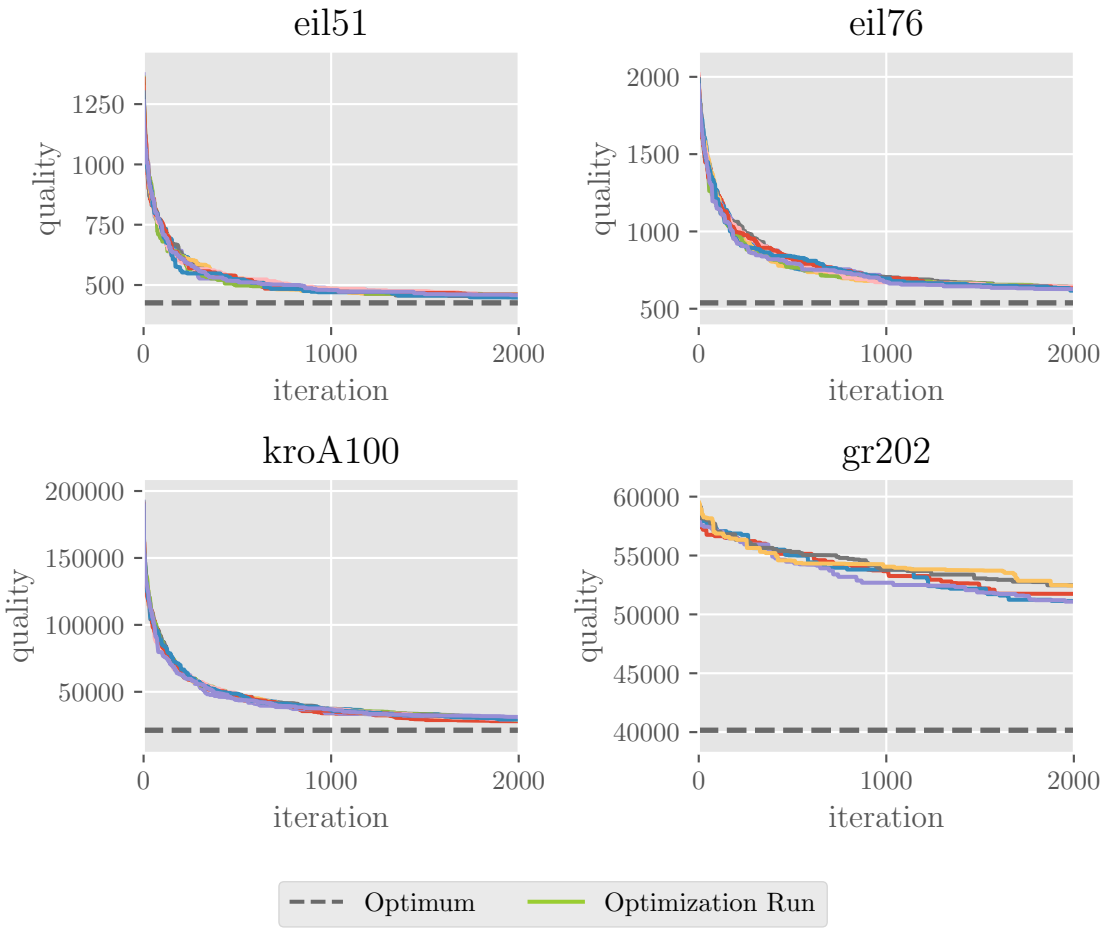


Figure 5.9: Development of iteration best and overall best solution

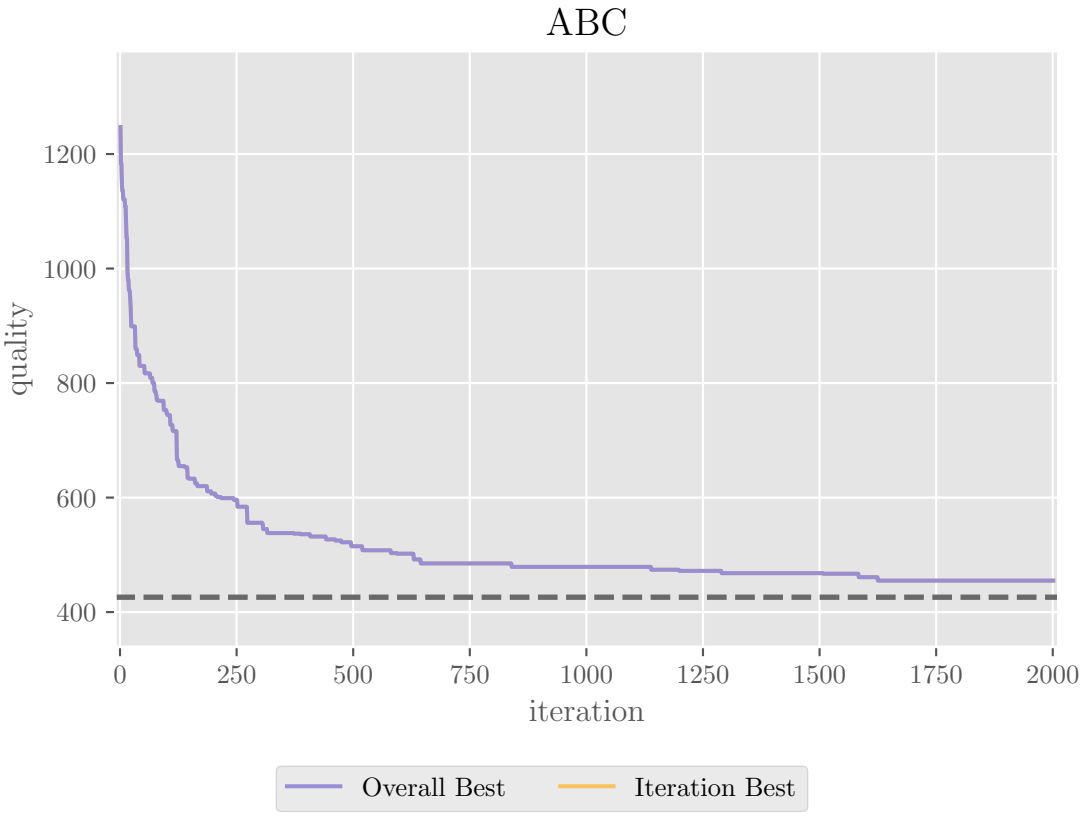
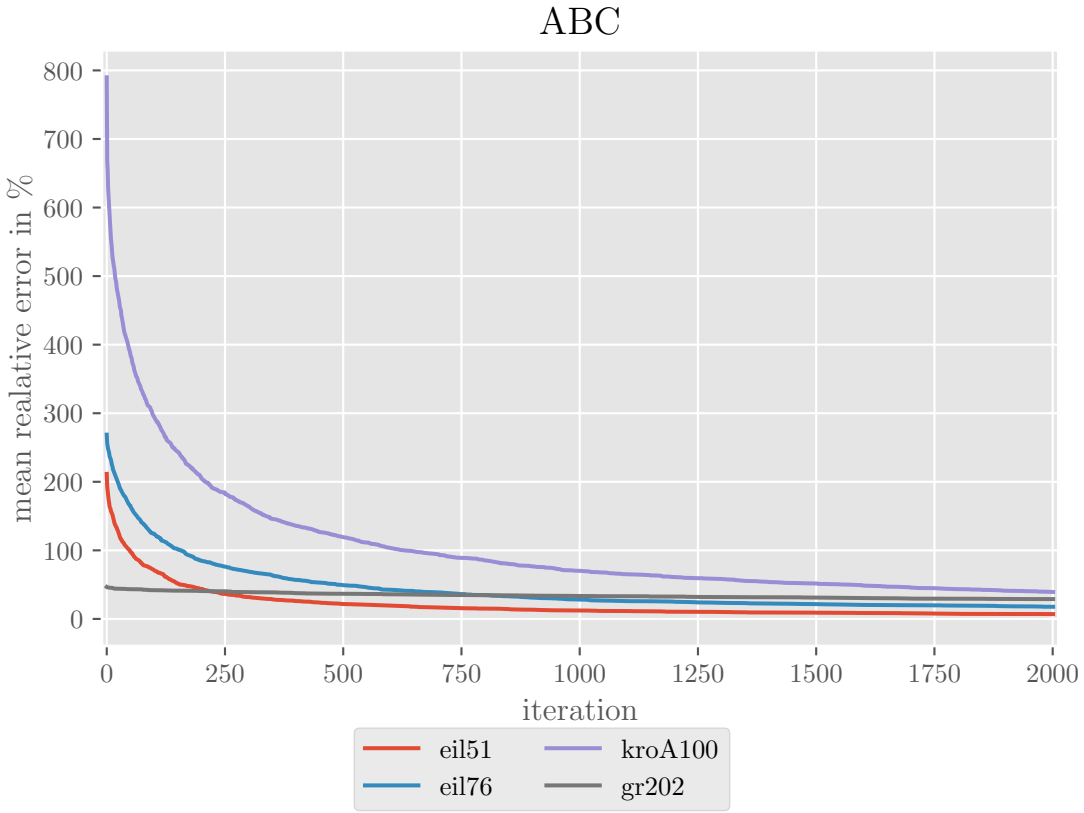


Figure 5.10: Mean error by problem instance



5.6.4 Comparison of Algorithms

To compare the algorithms, we consider the mean relative error (figure 5.11), success rate (figure 5.12) and average runtime (figure 5.13) for each TSP instance.

The FA was the clear winner in the performance comparison, as it is the only algorithm with provides reasonably good results within reasonable runtime. The ABC has somewhat good results for smaller problem instances, but extremely long runtime, and the ACO returned results only marginally better than those found by the nearest neighbour heuristic.

Only the FA succeeded in finding solutions within the 7.5% margin for all instances, while the ABC at least succeeded in 50% of cases for eil51, but not for larger numbers of cities.

While in the performance tests in this thesis, only one of three algorithms could provide reasonably good results, we know that all three algorithms have been used successfully in applications and are widely studied and well-known. Parameter Tuning is an essential step to achieving good results, and in this thesis, the parameters were not tuned to the specific problem instances, but rather adapted from the referenced literature. From this we conclude, that the results of the tests in this thesis are not representative for all applications, and different results may be achieved through parameter tuning and optimizing the implementation itself.

Figure 5.11: Comparison of algorithms by mean error.

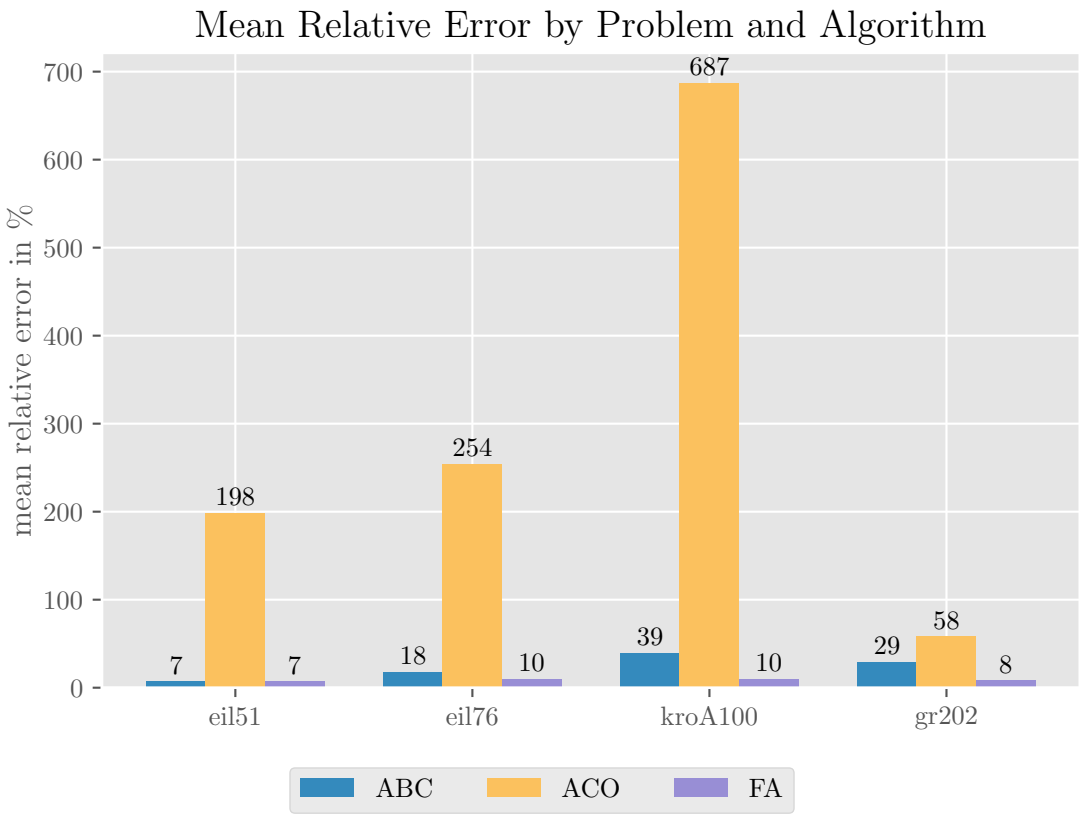


Figure 5.12: Comparison of algorithms by success rate.

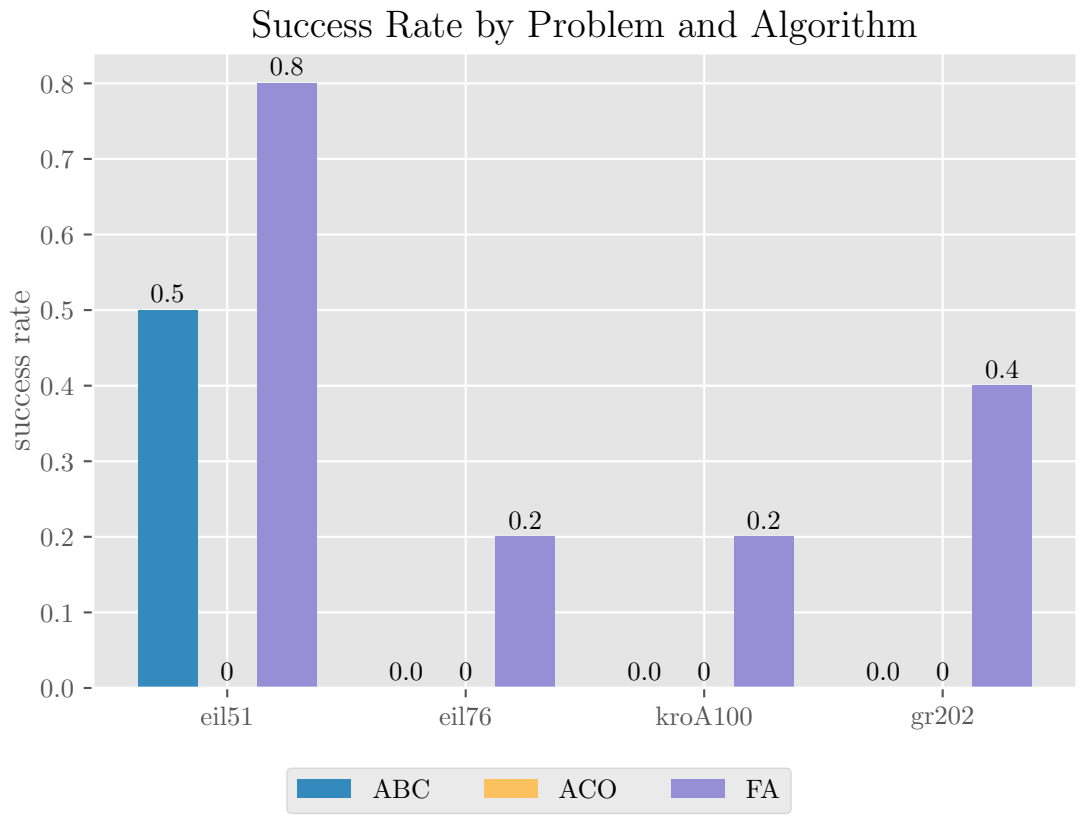
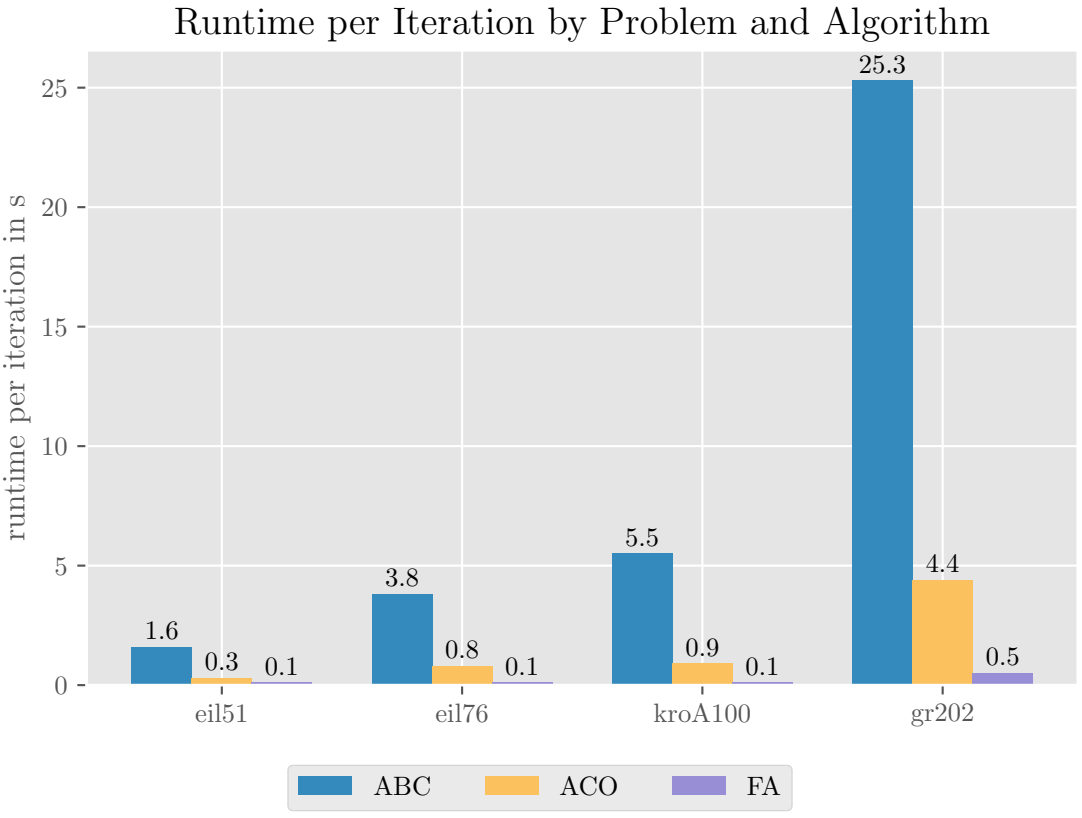


Figure 5.13: Comparison of algorithms by runtime per iteration.



Chapter 6

Conclusion

In this thesis, we gave an overview of the topic of metaheuristics, particularly bio-inspired algorithms such as Artifical Bee Colony, Ant Colony Optimization and Firefly Algorithm. These specific algorithms were chosen as representatives of bio-inspired algorithms, and have in common that they are population-based and are inspired by the collective behaviour of insects (a concept known as swarm intelligence).

These algorithms perform an iterative improvement of a population of solutions. While the ABC and FA use evolution strategies which employ variation operators, the ACO uses a blackboard strategy, where information about the search space is accumulated in a shared memory (the pheromone).

Other characteristic components can be identified in the algorithms as well: They employ strategies for generating new candidate solutions and various methods for guiding the search, such as the use of memory and selection strategies. Components can have diversifying and intensifying effects on the search, which must be balanced well to find global optima. Diversifying components include the use of randomisation, which is central to all metaheuristics. Intensifying components focus the search on a local region.

In the analysis and comparison of the algorithms, FA stood out with relatively good performance. It is also an easy to understand algorithm, which relies on only two control parameters. In contrast, the results of ACO were only marginally better than the heuristic solution, and ABC provided good results only for the smallest problem it was tested on, while having an extremely long runtime compared to FA.

However, as no parameter tuning to the specific problem instances was performed, and because we know that all three algorithms have been applied successfully in literature, the results of the tests are likely not representative and may be improved by parameter tuning and optimization of the implementation itself.

Finally, the algorithms themselves could be further adapted. In ACO, a local search step is commonly used to improve results. As seen in FA and ABC variants which were adopted from original versions for continuous problems to the combinatorial problem TSP, the search steps have to be represented by appropriate operators, such as the edge-based movement. This is another starting point for modifying the algorithms, often by incorporating parts of other metaheuristics, such as the GSTM,

which was proposed as an operator for genetic algorithms. This is a very common approach, and a large number of such hybrid metaheuristics can be found in the literature.

Bibliography

[1] O. Bozorg-Haddad, Ed., *Advanced Optimization by Nature-Inspired Algorithms*, en, Studies in Computational Intelligence, Springer Singapore, 2018, ISBN: 9789811052200. DOI: 10.1007/978-981-10-5221-7. [Online]. Available: https://www.springer.com/gp/book/9789811052200 (visited on 11/12/2019).

[2] X.-S. Yang, Ed., *Nature-Inspired Algorithms and Applied Optimization*, en, Studies in Computational Intelligence, Springer International Publishing, 2018, ISBN: 9783319676685. DOI: 10.1007/978-3-319-67669-2. [Online]. Available: https://www.springer.com/de/book/9783319676685 (visited on 11/12/2019).

[3] J. Del Ser, E. Osaba, D. Molina, X.-S. Yang, S. Salcedo-Sanz, D. Camacho, S. Das, P. Suganthan, C. Coello, and F. Herrera, "Bio-inspired computation: Where we stand and what's next," *Swarm and Evolutionary Computation*, vol. 48, Apr. 2019. DOI: 10.1016/j.swevo.2019.04.008.

[4] X.-S. Yang, S. Deb, Y.-X. Zhao, S. Fong, and X. He, "Swarm intelligence: Past, present and future," *Soft Computing*, vol. 22, no. 18, pp. 5923–5933, Sep. 2017, ISSN: 1433-7479. DOI: 10.1007/s00500-017-2810-5.

[5] K. Hussain, M. N. Mohd Salleh, S. Cheng, and Y. Shi, "Metaheuristic research: A comprehensive survey," *Artificial Intelligence Review*, vol. 52, no. 4, pp. 2191–2233, Jan. 13, 2018, ISSN: 1573-7462. DOI: 10.1007/s10462-017-9605-z.

[6] S. Patnaik, X.-S. Yang, and K. Nakamatsu, Eds., *Nature-Inspired Computing and Optimization: Theory and Applications*, en, Modeling and Optimization in Science and Technologies, Springer International Publishing, 2017, ISBN: 9783319509198. DOI: 10.1007/978-3-319-50920-4. [Online]. Available: https://www.springer.com/gp/book/9783319509198 (visited on 11/12/2019).

[7] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Computing Surveys*, vol. 35, no. 3, pp. 268–308, Sep. 2003, ISSN: 0360-0300. DOI: 10.1145/937503.937505.

[8] M. Dorigo, M. Birattari, and T. Stützle, "Ant colony optimization," *Computational Intelligence Magazine, IEEE*, vol. 1, pp. 28–39, Dec. 2006. DOI: 10.1109/MCI.2006.329691.

[9] G. K. Jati, R. Manurung, and Suyanto, "Discrete firefly algorithm for traveling salesman problem: A new movement scheme," in *Swarm Intelligence and Bio-Inspired Computation*, X.-S. Yang, Z. Cui, R. Xiao, A. H. Gandomi, and M. Karamanoglu, Eds., Oxford: Elsevier, 2013, ch. 13, pp. 295–312, ISBN: 978-0-12-405163-8. DOI: 10.1016/B978-0-12-405163-8.00013-2. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780124051638000132.

[10] G. Kizilates and F. Nuriyeva, "On the nearest neighbor algorithms for the traveling salesman problem," *Advances in Intelligent Systems and Computing*, vol. 225, pp. 111–118, Jan. 2013. DOI: 10.1007/978-3-319-00951-3_11.

[11] B. Chopard and M. Tomassini, *An Introduction to Metaheuristics for Optimization*, en, ser. Natural Computing Series. Springer International Publishing, 2018, ISBN: 9783319930725. DOI: 10.1007/978-3-319-93073-2. [Online]. Available: https://www.springer.com/gp/book/9783319930725.

[12] E.-G. Talbi, *Metaheuristics: From Design to Implementation*, en. John Wiley & Sons, May 2009, Google-Books-ID: SIsa6zi5XV8C, ISBN: 9780470496909.

[13] S. Luke, *Essentials of Metaheuristics*, second. Lulu, 2013, Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/. [Online]. Available: https://cs.gmu.edu/~sean/book/metaheuristics/.

[14] M. Saka, E. Doğan, and I. Aydogdu, "Analysis of swarm intelligence–based algorithms for constrained optimization," in *Swarm Intelligence and Bio-Inspired Computation*, X.-S. Yang, Z. Cui, R. Xiao, A. H. Gandomi, and M. Karamanoglu, Eds., Oxford: Elsevier, 2013, ch. 2, pp. 25–48, ISBN: 978-0-12-405163-8. DOI: 10.1016/B978-0-12-405163-8.00002-8. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780124051638000028.

[15] K. Sörensen, M. Sevaux, and F. Glover, "A history of metaheuristics," in *Handbook of Heuristics*, R. Martí, P. M. Pardalos, and M. G. C. Resende, Eds. Cham: Springer International Publishing, 2018, pp. 791–808, ISBN: 978-3-319-07124-4. DOI: 10.1007/978-3-319-07124-4_4.

[16] O. Kramer, *Computational Intelligence: Eine Einführung*, de, ser. Informatik im Fokus. Berlin Heidelberg: Springer-Verlag, 2009, ISBN: 9783540797388. DOI: 10.1007/978-3-540-79739-5. [Online]. Available: https://www.springer.com/gp/book/9783540797388.

[17] X.-S. Yang and M. Karamanoglu, "Swarm intelligence and bio-inspired computation: An overview," in *Swarm Intelligence and Bio-Inspired Computation*, X.-S. Yang, Z. Cui, R. Xiao, A. H. Gandomi, and M. Karamanoglu, Eds., Oxford: Elsevier, 2013, ch. 1, pp. 3–23, ISBN: 978-0-12-405163-8. DOI: 10.1016/B978-0-12-405163-8.00001-6. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780124051638000016.

[18] T. Dökeroğlu, E. Sevinç, T. Kucukyilmaz, and A. Cosar, "A survey on new generation metaheuristic algorithms," *Computers & Industrial Engineering*, Sep. 2019. DOI: 10.1016/j.cie.2019.106040.

[19] F. Glover and M. Laguna, *Tabu Search*. Kluwer Academic Publishers, Boston, 1997.

[20] X. S. Yang, "Firefly algorithm, stochastic test functions and design optimisation," *International Journal of Bio-Inspired Computation*, vol. 2, no. 2, p. 78, 2010. DOI: 10.1504/ijbic.2010.032124.

[21] M. A. Lones, "Metaheuristics in nature-inspired algorithms," in *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO Comp '14, Vancouver, BC, Canada: Association for Computing Machinery, 2014, pp. 1419–1422, ISBN: 9781450328814. DOI: 10.1145/2598394.2609841.

[22] G. Reinelt. (). "Tsplib 95," [Online]. Available: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/ (visited on 09/11/2020).

[23] R. Grant and M. Ritter. (). "Tsplib 95," [Online]. Available: https://tsplib95.readthedocs.io/en/stable/index.html (visited on 09/11/2020).

[24] D. Karaboga and B. Gorkemli, "Solving traveling salesman problem by using combinatorial artificial bee colony algorithms," *International Journal on Artificial Intelligence Tools*, vol. 28, no. 01, p. 1 950 004, 2019. DOI: 10.1142/S0218213019500040. eprint: https://doi.org/10.1142/S0218213019500040.

[25] The SciPy Community. (). "Random generator," [Online]. Available: https://numpy.org/doc/stable/reference/random/generator.html#numpy.random.default_rng (visited on 09/12/2020).

[26] ——, (). "Permuted congruential generator (64-bit, pcg64)," [Online]. Available: https://numpy.org/doc/stable/reference/random/bit_generators/pcg64.html (visited on 09/12/2020).

[27] D. Karaboga, "An idea based on honey bee swarm for numerical optimization, technical report - tr06," *Technical Report, Erciyes University*, Jan. 2005.

[28] D. Karaboga and B. Gorkemli, "A combinatorial artificial bee colony algorithm for traveling salesman problem," *2011 International Symposium on Innovations in Intelligent Systems and Applications*, pp. 50–53, 2011.

[29] D. Görkemli Beyza; Karaboga, "Quick combinatorial artificial bee colony -qcabc- optimization algorithm for tsp," in *2nd International Symposium on Computing in Informatics and Mathematics*, Dec. 19, 2013.

[30] M. S. Kıran, H. İşcan, and M. Gündüz, "The analysis of discrete artificial bee colony algorithm with neighborhood operator on traveling salesman problem," *Neural Computing and Applications*, vol. 23, no. 1, pp. 9–21, Jul. 1, 2013, ISSN: 1433-3058. DOI: 10.1007/s00521-011-0794-0.

[31] D. Karaboga, B. Gorkemli, C. Ozturk, and N. Karaboga, "A comprehensive survey: Artificial bee colony (abc) algorithm and applications," *Artificial Intelligence Review*, no. 42, pp. 21–57, Mar. 2012. DOI: 10.1007/s10462-012-9328-0.

[32] R. Anguluri, N. Lynn, S. Das, and P. Suganthan, "Computing with the collective intelligence of honey bees – a survey," *Swarm and Evolutionary Computation*, vol. 32, Jul. 2016. DOI: 10.1016/j.swevo.2016.06.001.

[33] M. Albayrak and N. Allahverdi, "Development a new mutation operator to solve the traveling salesman problem by aid of genetic algorithms," *Expert Syst. Appl.*, vol. 38, pp. 1313–1320, Mar. 2011. DOI: 10.1016/j.eswa.2010.07.006.

[34] A. Jakoby, *Discrete optimization*, en, Lecture Notes, Bauhaus-Universität Weimar, Oct. 2, 2019.

[35] D. M, "Optimization, learning and natural algorithms.," Ph.D. dissertation, Politecnico di Milano, Italy, 1992.

[36] M. Dorigo and G. Di Caro, "The ant colony optimization meta-heuristic," *New Ideas in Optimization*, Nov. 1999. [Online]. Available: https://www.researchgate.net/publication/2831286_The_Ant_Colony_Optimization_Meta-Heuristic.

[37] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, Apr. 1997, ISSN: 1941-0026. DOI: 10.1109/4235.585892.

[38] A. Chakraborty and A. K. Kar, "Swarm intelligence: A review of algorithms," in *Nature-Inspired Computing and Optimization: Theory and Applications*, S. Patnaik, X.-S. Yang, and K. Nakamatsu, Eds. Cham: Springer International Publishing, 2017, pp. 475–494, ISBN: 978-3-319-50920-4. DOI: 10.1007/978-3-319-50920-4_19.

[39] X. S. Yang and X. He, "Firefly algorithm: Recent advances and applications," *International Journal of Swarm Intelligence*, vol. 1, no. 1, p. 36, 2013. DOI: 10.1504/ijsi.2013.055801.

[40] I. Fister, I. Fister, X.-S. Yang, and J. Brest, "A comprehensive review of firefly algorithms," *Swarm and Evolutionary Computation*, vol. 13, pp. 34–46, Dec. 2013, ISSN: 2210-6502. DOI: 10.1016/j.swevo.2013.06.001.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien-oder Prüfungsleistung war.

Karoline Brehm