Optimization Algorithms

Thomas Weise (汤卫思)

February 22, 2026

Abstract

Optimization is the art of finding good solutions for problems that are hard to solve. From Logistics, the scheduling of work in a factory or the lectures of students, the packing of objects into bins, over the design of rotor wings to the synthesis of dynamic controllers for engines – all involve, implicitly or explicitly, optimization. In this book we explore this very wide field. We look at it from the perspective of a programmer. We try to find define a pattern common to most optimization problems and then we explore how algorithms can be designed on basis of this pattern. We step-by-step investigate more and more complicated optimization approaches and learn lessons along the way. This book follows a very practical approach. We focus less on theory and more on learning-by-doing and trial-and-error ... which, coincidentally, is very similar to how many of the algorithms that we will study work as well.

Contents

Preface

After writing "*Global Optimization Algorithms – Theory and Application*" [71] as PhD student a long time ago, I now want to write a more *practical* guide to optimization and metaheuristics. Currently, this book is in an early stage of development and work-in-progress, so expect many changes.

This book tries to introduce optimization in an accessible way for an audience of undergraduate and graduate students without background in the field. It tries to provide an intuition about how optimization algorithms work in practice, how to recognize optimization problems and the basic structure behind them, what things to look for when solving an optimization problem, and how to get from a simple, working, "proof-of-concept" approach to an efficient algorithm for a given problem.

We follow a "learning-by-doing" approach by trying to solve one practical optimization problem as example theme throughout the book. All algorithms are directly implemented and applied to that problem after we introduce them. This allows us to discuss their strengths and weaknesses based on actual results. We learn how to compare the performance of different algorithms. We try to improve the algorithms step-by-step, moving from very simple approaches, which do not work well, to efficient metaheuristics.

We use concrete examples and algorithm implementations written in `Python` [72]. The source code is freely as part of the `moptipy` package under the GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007. The code listings in the book will usually be abridged excerpts.

In order to fully understand the code examples, we recommend the reader to familiarize themselves with `Python`, `NumPy`, and `Matplotlib`. Of course, if you just read this book to learn about algorithms, you can ignore the source code examples.

Prof. Dr. Thomas Weise (汤卫思教授)
at the School of Artificial Intelligence and Big Data (人工智能与大数据学院)
of Hefei University (合肥大学),
in Hefei, Anhui, China (中国安徽省合肥市)

Contact me via email to tweise@hfuu.edu.cn with CC to tweise@ustc.edu.cn.

[book pdf]

[course website]

https://thomasweise.github.io/oa

This book was built using the following software:

```
Alpine Linux 3.23.2
Linux 6.11.0-1018-azure x86_64

python: 3.12.12
texgit_py: 0.9.11
texgit_tex: 0.9.7
pycommons: 0.8.88
pdflatex: pdfTeX 3.141592653-2.6-1.40.28 (TeX Live 2025)
biber: 2.21
makeglossaries: 4.7 (2025-05-14)
ghostscript: 10.06.0 (2025-09-09)

date: 2026-02-22 09:30:00 +0000
```

Chapter 1

Introduction

Algorithms influence a bigger and bigger part of both of our daily, private, and work life. They suggest interesting movies for us to watch or products to purchase. They help us find efficient routes when driving by car or match us to the next available nearby taxi. They control advertisement campaigns and suggest product pricing policies [50]. They support us by suggesting good decisions in a variety of fields, ranging from engineering, timetabling and scheduling, product design, to logistic planning. They will be the most important element of the transition of our industry to smarter manufacturing and intelligent production, where they can automate a variety of tasks, as illustrated in Figure 1.1.

Optimization and Operations Research (OR) provide us with algorithms that propose good solutions to a very wide range of questions. These solutions achieve a predefined goal while minimizing (at least) one resource requirement, be it costs, energy consumption, space, the time requirement, and so on. Besides saving direct costs, the reduction of resource consumption often is also good for the environment. Therefore, optimization can help us to become more efficient both economically and ecologically. We can thus already list three obvious reasons why optimization will be a key technology for the next century:

1. The automation of production can improve the work life by reducing manual work while increasing productivity and product quality. However, any form of intelligent production or smart manufacturing needs automated decisions. Since these decisions should be *intelligent,* they can only come from a process which involves optimization in one way or another.

2. All branches of industry, all service sectors, as well as cities and regions face both global and

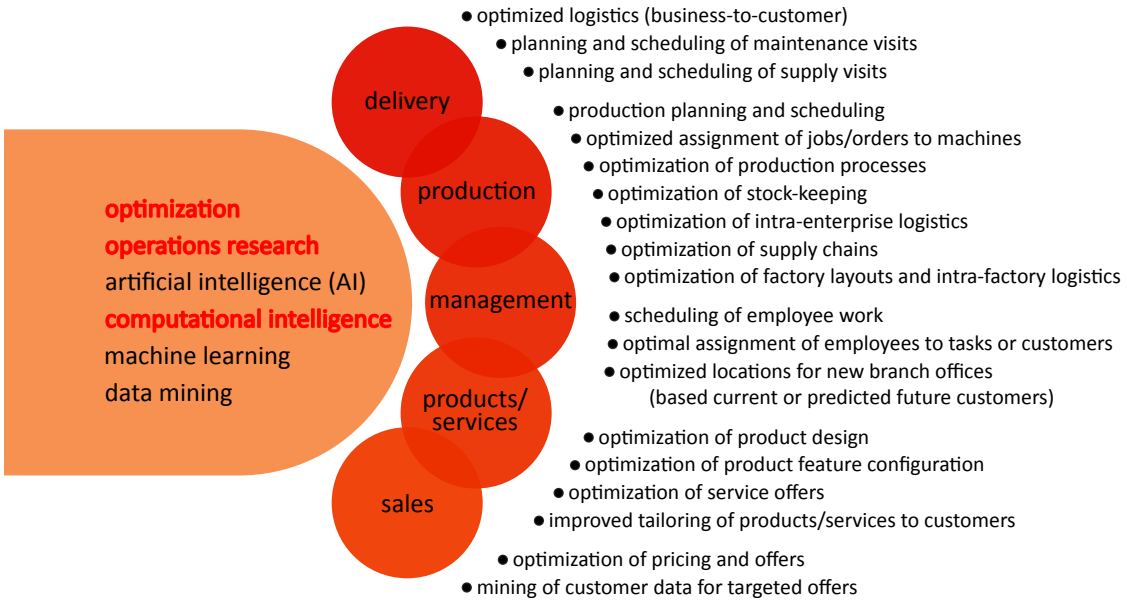

Figure 1.1: Examples for applications of optimization, computational intelligence, machine learning techniques in five fields of smart manufacturing: the production itself, the delivery of the products, the management of the production, the products and services, and the sales level.

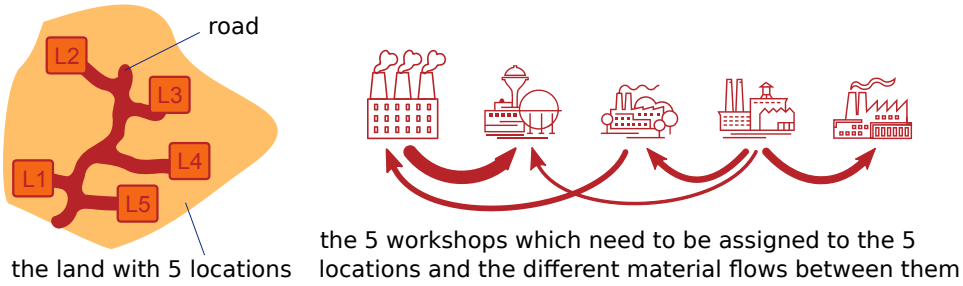

Figure 1.2: Illustrative sketch of a quadratic assignment scenario, where different buildings of a factory need to be laid out on a plot of land.

local competition. Those who can reduce their resource consumption and costs while improving product, production, or service quality and efficiency will have the edge. One key technology for achieving this is better planning via optimization.

3. Our world suffers from both depleting resources and too much pollution. Optimization can "give us more while needing less." This often leads to more environmentally friendly processes.

But how can algorithms help us to find solutions for hard problems in a variety of different fields? How general are these algorithms? How can they help us to make good decisions? How can they help us to save resources?

In this book, we will try to answer all of these questions. We will explore quite a lot of different optimization algorithms. We will look at their actual implementations and we will apply them to example problems to see what their strengths and weaknesses are.

1.1 Examples

Let us first look at some typical use cases of optimization.

1.1.1 Example: Layout of Factories

There are both dynamic and static aspects of intelligent production, as well as all sorts of nuances in between. The question of where to put which facility in a factory is a rather static, but quite important aspect. Let us assume we own a company and bought a plot of land to construct a new factory. Of course we know which products we will produce in this factory. We also know which facilities we need to construct, i.e., the workshops, storage depots, and maybe an administrative building. What we need to decide is where to place them on our land, as illustrated in Figure 1.2.

Let us assume we have n locations on our plot of land that we can use for our n facilities. In some locations, there might be already buildings, in others, we may need to construct them anew. For each facility and location pair, a different construction cost may arise. A location with an existing shed might be a good solution to put a warehouse. However, if we want to put the administration building there, we would first need to demolish the shed. These are the static costs.

But placing the facilities also has a very big impact on the running costs.

For every possible plan, costs also arise from the relative distances between the facilities that we wish to place. Maybe there is a lot of material flow between two of the workshops. Finished products and raw material may need to be transported between a workshop and the storage depot. Between the administration building and the workshops, on the other hand, there will usually be no material flow. Of course, the distance between two facilities will depend on the locations we pick for them. For each pair of facilities that we place on the map, flow costs will arise as a function of the amount of material to be transported between them and the distance of their locations.

The total cost of an assignment of facilities to locations is therefore the sum of the resulting base costs and flow costs. Our goal would be to find the assignment (i.e., the plan) with the smallest possible total cost.

This scenario is called Quadratic Assignment Problem (QAP) [6]. It has been subject to research since the 1950s [4]. QAPs appear in wide variety of scenarios such as the location of facilities on a plot of land or the placement of work stations on the factory floor. Even if we need to place components

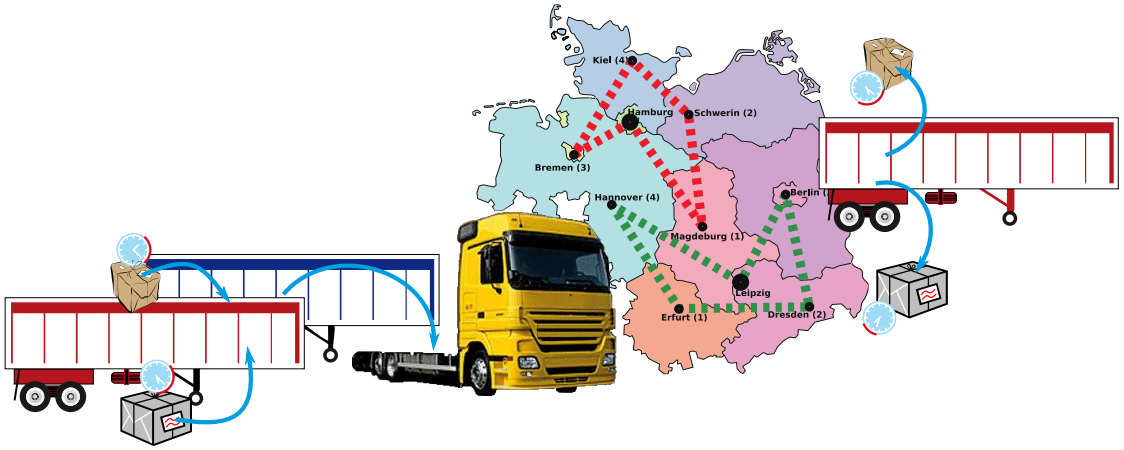

Figure 1.3: Illustrative sketch of logistics problems: Orders require us to pick up some items at source locations within certain time windows and deliver them to their destination locations, again within certain time windows. We need to decide which containers and vehicles to use and over which routes we should channel the vehicles.

on a circuit board in a way that minimizes the total wire length, we basically have a QAP, too [65]! Despite being relatively simple to understand, the QAP is hard to solve [61].

1.1.2 Example: Route Planning for a Logistics Company

Another, more dynamic application area for optimization is logistics. Let us look at a typical real-world scenario from this field [74, 75]: the situation of a logistics company that fulfills delivery tasks for its clients. A client can order one or multiple containers to be delivered to her location within a certain time window. She will fill the containers with goods, which are then to be transported to a destination location, again within a certain time window. The logistics company may receive many such customer orders per day, maybe several hundreds or even thousands. The company may have multiple depots, where containers and trucks are stored. For each order, it needs to decide which container(s) to use and how to get them to the customer, as sketched in Figure 1.3. The trucks it owns may have different capacities and could, e.g., carry either one or two containers. Besides using trucks, which can travel freely on the map, it may also be possible to utilize trains. Trains have higher capacities and can carry many containers. Different from trucks, they must follow specific schedules. They arrive and depart at fixed times to/from fixed locations. For each possible vehicle, different costs could occur. Containers can be exchanged between different vehicles at locations such as parking lots, depots, or train stations.

The company could have the goals to fulfill all transportation requests *at the lowest cost*. Actually, it might seek to maximize its profit, which could even mean to outsource some tasks to other companies. The goal of optimization then would be to find the assignment of containers to delivery orders and vehicles and of vehicles to routes, which maximizes the profit. And it should do so within a limited, feasible time.

Of course, there is a wide variety of possible logistics planning tasks. Besides our real-world example above, a classical task is the TSP [2, 27, 39], where the goal is to find the shortest round-trip tour through n cities, as sketched in Figure 1.4. Many other scenarios can be modeled as such logistics questions, too: If a robot arm needs to several drill holes into a circuit board, finding the shortest tour means solving a TSP and will speed up the production process, for instance [26].

1.1.3 Example: Packing, Cutting Stock, and Knapsack

Let's say that your family is moving to a new home in another city. This means that you need to transport all of your belongings from your old to your new place, your PC, your clothes, maybe some furniture, a washing machine, and a refrigerator, as sketched in Figure 1.5. You cannot pack everything into your car at once, so you will have to drive back and forth a couple of times. But how often will you have to drive? Packing problems [21, 62] aim to package sets of objects into containers as efficient as possible, i.e., in such a way that we need as few containers as possible. Your car can be thought of

Figure 1.4: A Traveling Salesperson Problem (TSP) through eleven cities in China, including the beautiful city of Hefei (合肥).

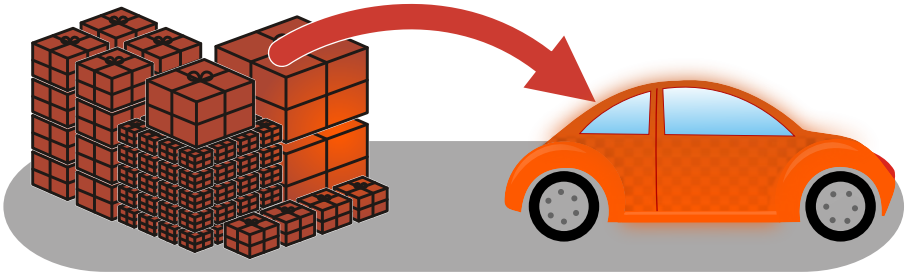

Figure 1.5: A sketch illustrating a packing problem.

as a container and whenever it is filled, you drive to the new flat. If you need to fill the container four times, then you have to drive back and forth four times.

Such Bin Packing Problems (BPPs) exist in many variants and are very related to cutting stock problems [21]. They can be one-dimensional [18], for example if we want to transport dense/heavy objects with a truck where the maximum load weight is limiting factor while there is enough space capacity. This is similar to having a company which puts network cables into people's homes and therefore bulk purchases reels with 100m of cables each. Of course, each home needs a different required total length of cables and we want to cut our cables such that we need as few reels as possible.

A two-dimensional variant [44, 83], could correspond to printing a set of (rectangular) images of different sizes on (rectangular) paper. Assume that more than one image fits on a sheet of paper but we have too many images for one piece of paper. We can cut the paper after printing to separate the single images. We then would like to arrange the images such that we need as few sheets of paper as possible.

The three-dimensional variant then corresponds to our moving homes scenario. Of course, there are many more different variants — the objects we want to pack could be circular, rectangular, or have an arbitrary shape. We may also have a limited number of containers and thus may not be able to pack all objects, in which case we would like to only package those that give us the most profit (arriving at a task called knapsack problem [48]).

1.1.4 Example: Job Shop Scheduling Problem

Another typical optimization task arises in manufacturing, namely the assignment ("scheduling") of tasks ("jobs") to machines in order to optimize a given performance criterion ("objective"). Scheduling [58, 59] is one of the most active areas of operational research for more than six decades.

In the Job Shop Scheduling Problem (JSSP) [5, 10, 38, 66], we have a factory ("shop") with several

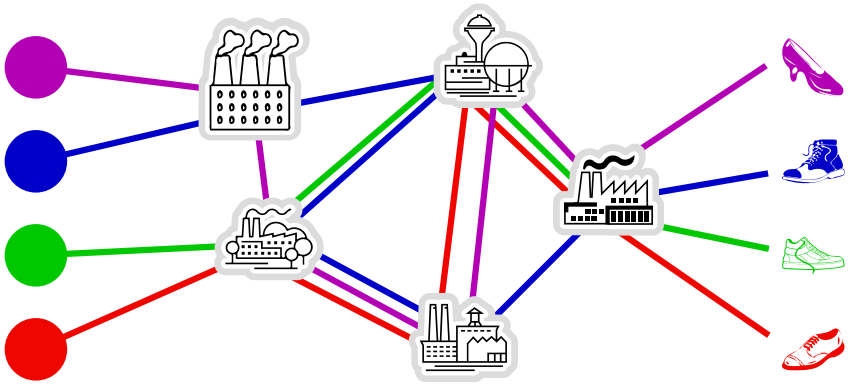

Figure 1.6: Illustrative sketch of a JSSP scenario with four jobs where four different types of shoe should be produced, which require different workshops ("machines") to perform different production steps.

machines. We receive a set of customer orders for products which we have to produce. We know the exact sequence in which each product/order needs to pass through the machines and how long it will need at each machine. Each production job has one production step ("operation") for each machine on which it needs to be processed. These operations must be performed in the right sequence. Of course, no machine can process more than one operation at a time. While we must obey these constraints, we can decide about the time at which each of the operations should begin. Often, we are looking for the starting times that lead to the earliest completion of all jobs, i.e., the shortest makespan.

Such a scenario is sketched in Figure 1.6, where four orders for different types of shoe should be produced. The resulting jobs pass through different workshops (or machines, if you want) in different order. Some, like the green sneakers, only need to be processed by a subset of the workshops.

This general scenario encompasses many simpler problems. For example, if we only produce one single product, then all jobs would pass through the same machines in the same order. Customers may be able to order different quantities of the product, so the operations of the different jobs for the same machine may need different amounts of time. This is the so-called Flow Shop Scheduling Problem (FSSP) — and it has been defined back in 1954 [35]!

Clearly, since the JSSP allows for an *arbitrary* machine order per job, being able to solve the JSSP would also enable us to solve the FSSP, where the machine order is fixed. We will introduce the JSSP in detail and use it as the main example in this book on which we will step-by-step exercise different optimization methods.

1.1.5 Summary

The examples we have discussed so far are, actually, related to each other. They all fit into the broad areas of operational research and smart manufacturing [15, 30]. The goal of smart manufacturing is to optimize development, production, and logistics in the industry. Therefore, computer control is applied to achieve high levels of adaptability in the multi-phase process of creating a product from raw material. The manufacturing processes and maybe even whole supply chains are networked. Product lot sizes are small and a high flexibility is required to adapt production processes to customer wishes. This creates the requirement for a large degree of automation and automatic intelligent decisions. The key technology necessary to propose such decisions are optimization algorithms. In a perfect world, the whole production process as well as the warehousing, packaging, and logistics of final and intermediate products would take place in an *optimized* manner. No time or resources would be wasted as production gets cleaner, faster, and cheaper while the quality increases.

1.2 Metaheuristics: Why do we need them?

The main topic of this book will be metaheuristic optimization (although I will eventually also discuss some other methods (remember: work in progress). So why do we need metaheuristic algorithms? Why should you read this book?

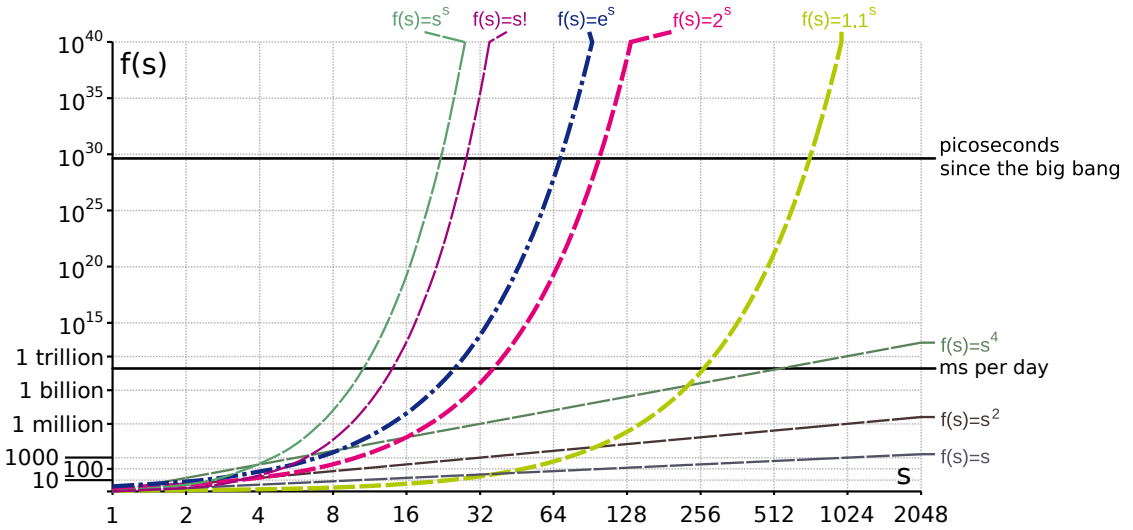

Figure 1.7: The growth of different functions in a log-log scaled plot. Exponential functions grow very fast. This means that an algorithm which needs $\sim 2^s$ steps to solve an optimization problem of size s quickly becomes infeasible if s grows.

1.2.1 Good Solutions within Acceptable Runtime

The first and foremost reason is that they can provide us good solutions within reasonable time. It is easy to understand that there are some problems which are harder to solve than others. Every one of us already knows this from the mathematics classes in school. Of course, the example problems discussed before cannot be attacked as easily as solving a single linear equation. They require algorithms, they require computer science.

Ever since primary school, we have learned many problems and types of equations that we can solve. Unfortunately, theoretical computer science shows that for many problems, the time we need to find the best-possible solution can grow *exponentially* with the number of involved variables in the worst case. The number of involved variables here could be the number of cities in a TSP, the number of jobs or machines in a JSSP, or the number of objects to pack in a, well, packing problem. A big group of such complicated problems are called $\mathcal{NP}$-hard [8, 38]. Unless some unlikely breakthrough happens in terms of which problems can be solved efficiently and which not [11, 36], there will be many problems that we cannot always solve exactly within reasonable time. Each and every one of the example problems discussed belongs to this type!

As sketched in Figure 1.7, the exponential function rises very quickly. One idea would be to buy more computers for bigger problems and to simply parallelize the computation. Well, parallelization can provide a linear speed-up at best. If we have two CPU cores, we can solve the problem in half of the time and if we have three CPU cores, we can do it in one third of the time, and so on ... if the problem lends itself to parallelization (and not all of them do). Either way, we are dealing with problems where the runtime requirements may double every time we add a single new decision variable. And no: Quantum computers are not the answer. Most likely, they cannot even solve these problems qualitatively faster either [1].

So what can we do to solve such problems? The exponential time requirement occurs if we make *guarantees* about the solution quality, especially about its optimality, over all possible scenarios. What we can do, therefore, is that we can trade-in the *guarantee* of finding the best possible solution for lower runtime requirements. We can use algorithms from which we *hope* that they find a good *approximation* of the optimum, i.e., a solution which is very good with respect to the objective function, but which do not *guarantee* that this result will be the best possible solution. We may sometimes be lucky and even find the optimum, while in other cases, we may get a solution which is close enough. And we will get this within acceptable time limits.

In Figure 1.8 we illustrate this idea on the example of the Traveling Salesperson Problem (TSP) [2, 27, 39] briefly mentioned in Section 1.1.2. The goal of solving the TSP is to find the shortest round trip tour through n cities. The TSP is $\mathcal{NP}$-hard [23, 27]. Today, it is possible to solve many large instances of this problem to optimality by using sophisticated *exact* algorithms [12, 14]. Yet, finding

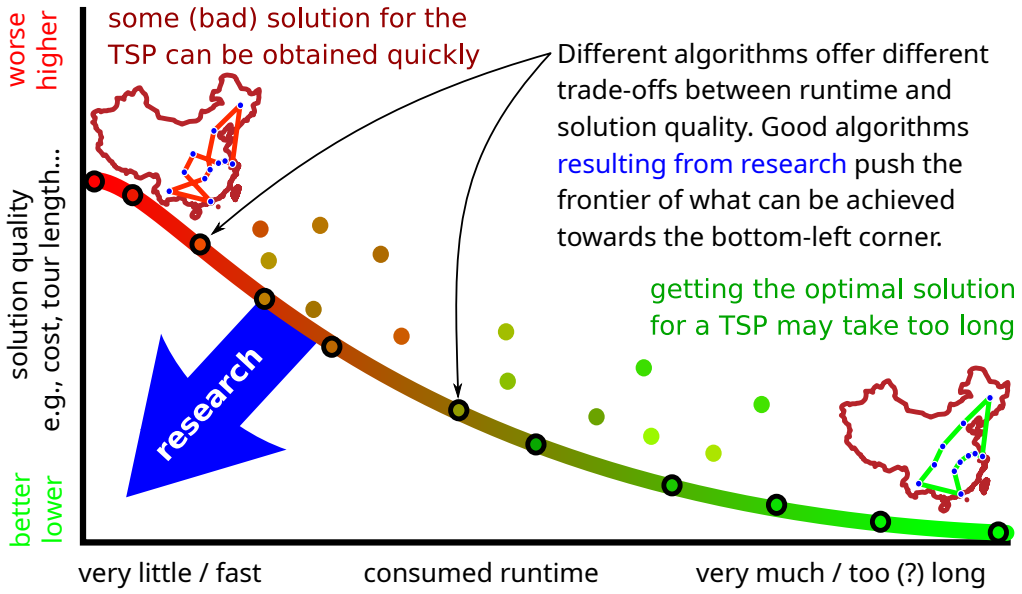

Figure 1.8: The trade-off between solution quality and runtime.

the *shortest possible tour* for a particular TSP might still take many years if you are unlucky. Finding just *one tour* is, however, very very easy: I can write down the cities in any particular order. Of course, I can visit the cities in an arbitrary order. That is an entirely valid solution, and I can obtain it basically in 0 time. This "tour" would probably be very bad, very long, and generally not a good idea.

In the real world, we need something in between. We need a solution which is as good as possible as fast as possible. Heuristic and metaheuristic algorithms offer different trade-offs of solution quality and runtime. Different from exact algorithms, they do not guarantee to find the optimal solution and often make no guarantee about the solution quality at all. Still, they often allow us to get very good solutions for computationally hard problems in short time. They may often still discover them (just not always, not guaranteed).

1.2.2 Good Solutions within Acceptable Development Time

Saying that we need a good algorithm to solve a given problem is very easy. Developing a good algorithm to solve a given problem is not, as any graduate student in the field can probably confirm. Before, I stated that great exact algorithms for the TSP exist [12, 14], that can solve many TSPs quickly (although not all). There are years and years of research in these algorithms. Even the top heuristic and metaheuristic algorithm for the TSP today result from many years of targeted research [29, 53, 79] and their implementation from the algorithm specification alone can take months [78]. Unfortunately, if you do not have plain TSP, but one with some additional constraints — say, time windows to visit certain cities — the optimized, state-of-the-art TSP solvers are no longer applicable. And in a real-world application scenario, you do not have years to develop an algorithm. What you need are simple, versatile, general algorithm concepts that you can easily adapt to your problem at hand. Something that can be turned into a working prototype within a few weeks.

Metaheuristics are the answer [51, 71]. They are general algorithm concepts into which we can plug problem-specific modules. General metaheuristics are usually fairly easy to implement and deliver acceptable results. Once a sufficiently well-performing prototype has been obtained, we could go and integrate it into the software ecosystem of the customer. We also can try to improve its performance using different ideas . . . and years and years of blissful research, if we are lucky enough to find someone paying for it.

Part I

The Structure of Optimization Problems

Chapter 2

Introduction

From the examples that we have seen, we know that optimization problems come in all kinds of different shapes and forms. Without practice, it is not directly clear how to identify, define, understand, or solve them. Moreover, it is not really clear how we can solve such a wide range of problems using the same kind of methods.

The goal of this part of the book is to bring some order into this mess. We will approach an optimization task step-by-step by formalizing its components, which will then allow us to apply efficient algorithms to it. This *structure of optimization* is a blueprint that can be used in many different scenarios as basis to apply different optimization algorithms.

We will approach this domain from the perspective of a programmer. Imagine you are a programmer and your job would be to, well, make a program that solves optimization problems. Now, an "optimization problem" seems to be a very general and amorphous thing. The first thing you would try to do is to discover some components that commonly occur in all of the optimization problems you can think of. If you can manage to specify how such components look like, what information and functionality they provide, then you are a step closer to fulfilling your task. Then, you can develop algorithms that use these information and functionality to find solutions. This is what we will do.

First, let us clarify what *optimization problems* actually are.

Definition 2.1: Optimization Problem I

An *optimization problem* is a situation $\mathcal{I}$ which requires deciding for one choice from a set of possible alternatives in order to reach a predefined/required goal at minimal costs.

Definition 2.1 presents an economical point of view on optimization in a rather informal manner. But the points come across: We want to reach a certain goal, e.g., visit all cities in a TSP, process all jobs in a JSSP, pack all the items into bins in a BPP, or assign all factories to locations in the QAP. We have several possible ways to reach that goal and we need to choose one among them. Each of these possible choices has an associated cost. Since all of them lead to reaching the goal, we want to pick the one choice with the minimal costs. We can refine this situation into the more mathematical formulation given in Definition 2.2.

Definition 2.2: Optimization Problem II

The goal of solving an *optimization problem* is finding an input $y \in \mathbb{Y}$ for which a function $f : \mathbb{Y} \mapsto \mathbb{R}$ takes on a value as small as possible.

From these definitions, we can already deduce a three necessary components that make up such an optimization problem. We will look at them from the perspective of a programmer:

1. The first obvious component is a data structure $\mathbb{Y}$ representing possible solutions to the problem. This one half of the output of the optimization software.
2. Then, there is the so-called objective function $f : \mathbb{Y} \mapsto \mathbb{R}$, which rates the quality of the candidate solutions $y \in \mathbb{Y}$. It basically returns the cost of a solution and we usually want to minimize it.

3. Finally, there must be some way to instantiate the data structure $\mathbb{Y}$, i.e., to sample one y from $\mathbb{Y}$.

Backmatter

Glossary

$(1+1)$ **EA** The $(1+1)$ EA is a local search algorithm that retains the best solution x_c discovered so far during the search [7, 20]. In each step, it applies a unary search operator to this best-so-far solution x_c and derives a new solution x_n. If the new solution x_n is *better or equally good* when compared with x_c, i.e., not worse, then it replaces it, i.e., is stored as the new x_c. If the search space are bit strings of length n, then the $(1+1)$ EA uses a unary search operator that flips each bit independently with probability m/n, where usually $m = 1$. This operator is the main difference to randomized local search (RLS). The $(1+1)$ EA is a special case of the $(\mu+\lambda)$ evolutionary algorithm ($(\mu+\lambda)$ EA) where $\mu = \lambda = 1$.

EA An *evolutionary algorithm* is a metaheuristic optimization method that maintains a population of candidate solutions, which undergo selection (where better solutions are chosen with higher probability) and reproduction (where mutation and recombination create a new candidate solution from one or two existing ones, repectively) [3, 71].

$(\mu+\lambda)$ **EA** The $(\mu+\lambda)$ EA is an evolutionary algorithm (EA) where, in each generation, λ offspring solutions are generated from the current population of μ parent solutions. The offspring and parent populations are merged, yielding $\mu+\lambda$ solutions, from which then the best μ solutions are ratained to form the parent population of the next generation. If the search space is the bit strings of length n, then this algorithm usually applies a mutation operator flipping each bit independently with probability $1/n$.

BPP The *Bin Packing Problem* [21, 62] is one of the classical problems from OR. The goal is to pack a set of objects into containers, using as few containers as possible. Different flavors of BPPs can be distinguished, e.g., by the number of dimensions [18, 44] that the objects and containers have, and/or whether the objects can be rotated [83], as well as other constraints and limitations.

$f(y)$ The *objective function* $f : \mathbb{Y} \mapsto \mathbb{R}$ can compute a cost or rating for each candidate solution y in the solution space $\mathbb{Y}$. The smaller this value, the better the solution, i.e., in the context of this work, objective functions are subject to minimization. Objective functions are not necessarily continuous and it is also quite common that they have integer or natural numbers as results, e.g., we often have $f : \mathbb{Y} \mapsto \mathbb{N}_0$ instead of $f : \mathbb{Y} \mapsto \mathbb{R}$..

FFA *Frequency Fitness Assignment* is a algorithm plugin for optimization methods applied to discrete or combinatorial problems with not-too-many different possible objective values. It replaces the objective values in all comparisons with their absolute encounter frequency so far during the search. FFA has successfully been applied to the QAP [9].

FSSP The *Flow Shop Scheduling Problem* is a variant of a JSSP where all products go through the same sequence of machines [35].

Git is a distributed Version Control Systems (VCS) which allows multiple users to work on the same code while preserving the history of the code changes [64, 68]. Learn more at `https://git-scm.com`.

GitHub is a website where software projects can be hosted and managed via the Git VCS [57, 68]. Learn more at `https://github.com`.

$\mathcal{I}$ the problem instance data, i.e., the data that is the input of the optimization algorithm. The solution space $\mathbb{Y}$ and the objective function f as well as all components building on top of them

are defined based on the concrete values in $\mathcal{I}$. For example, in an *instance* of the TSP could define the number n of cities and their distances. If our solutions are permutations of cities, then n is also the dimension of $\mathbb{Y}$. The objective function f accesses the city distances to compute the tour lengths..

JSSP The *Job Shop Scheduling Problem* [5, 10, 38] is one of the most prominent and well-studied scheduling tasks. In a JSSP instance, there are k machines and m jobs [66]. Each job must be processed once by each machine in a job-specific sequence and has a job-specific processing time on each machine. The goal is to find an assignment of jobs to machines that results in an overall shortest makespan, i.e., the schedule which can complete all the jobs in the shortest time. The JSSP is $\mathcal{NP}$-complete [8, 38].

Matplotlib is a `Python` package for plotting diagrams and charts [32–34, 55]. Learn more at at `https://matplotlib.org` [33].

ML Machine Learning, see, e.g., [63]

moptipy is the *Metaheuristic Optimization in `Python`* library [76]. It has been used in several different research works, including [9, 41–43, 67, 81–83]. Learn more at `https://thomasweise.github.io/moptipy` and `https://thomasweise.github.io/moptipyapps`.

$\mathbb{N}_0$ the set of the natural numbers *including* 0, i.e., 0, 1, 2, 3, and so on. It holds that $\mathbb{N}_0 \subset \mathbb{Z}$.

$\mathcal{NP}$ is the class of computational problems that can be solved in polynomial time by a non-deterministic machine and can be verified in polynomial time by a deterministic machine (such as a normal computer) [25].

$\mathcal{NP}$-complete A decision problem is $\mathcal{NP}$-complete if it is in $\mathcal{NP}$ and all problems in $\mathcal{NP}$ are reducible to it in polynomial time [25, 60]. A problem is $\mathcal{NP}$-complete if it is $\mathcal{NP}$-hard and if it is in $\mathcal{NP}$.

$\mathcal{NP}$-hard Algorithms that guarantee to find the correct solutions of $\mathcal{NP}$-hard problems [8, 11, 38] need a runtime that is exponential in the problem scale in the worst case. A problem is $\mathcal{NP}$-hard if all problems in $\mathcal{NP}$ are reducible to it in polynomial time [25].

NumPy is a fundamental package for scientific computing with `Python`, which offers efficient array datastructures [19, 28, 34]. Learn more at `https://numpy.org` [54].

OR Operations Research (or Operational Research) is the application of sciences such as mathematics and computer science to the management and organization of systems, organizations, enterprises, factories, or projects. It encompasses the development and application of problem-solving methods and techniques (such as mathematical optimization, simulation, queueing theory and other stochastic models) with the goal to improve decision-making and efficiency [17].

Python The `Python` programming language [31, 40, 46, 72], i.e., what you will learn about in our book [72]. Learn more at `https://python.org`.

QAP The *Quadratic Assignment Problem* is an optimization problem where the goal is to assign a set of n facilities to a set of n locations [6, 9, 37, 45]. Such an assignment can be represented as a permutation x of the first n natural numbers, where x_i specifies the location where facility i should be placed. For each QAP, a distance matrix D is given, where D_{pq} specifies the distance from location p to location q, as well as a flow matrix F, where F_{ij} is the amount of material flowing from facility i to facility j. The objective function f then rates a permutation x as $f(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} D_{x_i x_j} F_{ij}$. The QAP is $\mathcal{NP}$-complete [61].

$\mathbb{R}$ the set of the real numbers.

RLS Randomized local search retains the best solution x_c discovered so far during the search and, in each step, it applies a unary search operator to this best-so-far solution x_c and derives a new solution x_n. If the new solution x_n is *better or equally good* when compared with x_c, i.e., not worse, then it replaces it, i.e., is stored as the new x_c. If the search space are bit strings of length n, then RLS uses a unary search operator that flips exactly one bit. This operator is the main difference to $(1+1)$ evolutionary algorithm ($(1+1)$ EA).

SciPy is a `Python` library for scientific computing [34, 70]. Learn more at `https://scipy.org`.

SGA The Standard Genetic Algorithm [3, 16, 24, 49, 52, 71] was the first population EA. It maintains a population of solutions and applies mutation and crossover to generate offspring solutions. It uses fitness proportionate selection to choose which solutions should "survive" into the next generation, which today is considered a very bad design choice, see, e.g., [80].

TSP In an instance of the *Traveling Salesperson Problem*, also known as *Traveling Salesman Problem*, a set of n cities or locations as well as the distances between them are defined [2, 27, 39, 41, 73, 77]. The goal is to find the shortest round-trip tour that starts at one city, visits all the other cities one time each, and returns to the origin. The TSP is one of the most well-known $\mathcal{NP}$-hard combinatorial optimization problems [23, 27].

TTP The *Traveling Tournament Problem* (TTP) is the combinatorial optimization problem of both efficiently and fairly organizing a tournament of n teams that play against each other in a pairwise fashion [22, 81]. The efficient part boils down to arranging the games such that the total travel length is short, which is somewhat similar to the classical TSP. Initially, each team is at its home location. On each day, a team needs to travel if its scheduled game is not at its present location. On the last day, each team may need to travel back home unless their last game is a home game. The total travel length sums up the lengths of all travels over all teams. The fair part is represented in several constraints, such as doubleRoundRobin, compactness, maxStreak, and noRepeat. The TTP is $\mathcal{NP}$-hard [69].

VCS A *Version Control System* is a software which allows you to manage and preserve the historical development of your program code [68]. A distributed VCS allows multiple users to work on the same code and upload their changes to the server, which then preserves the change history. The most popular distributed VCS is Git.

$\mathbb{Y}$ the set of candidate solutions of an optimization problem is called the *solution space* $\mathbb{Y}$. The solution space corresponds to the data structure holding the candidate solutions y. These are the elements that are passed to the user after the optimization process is completed. They are also the parameters of the objective function(s) $f : \mathbb{Y} \mapsto \mathbb{R}$..

y a candidate solution $y \in \mathbb{Y}$ is one possible solution to an optimization problem. After the optimization process is completed, one or multiple such solutions are passed to the user..

$\mathbb{Z}$ the set of the integers numbers including positive and negative numbers and 0, i.e., . . . , -3, -2, -1, 0, 1, 2, 3, . . . , and so on. It holds that $\mathbb{Z} \subset \mathbb{R}$.

Bibliography

[1] Scott Aaronson. "The Limits of Quantum Computers". *Scientific American* 298(3):62–69, Mar. 2008. New York, NY, USA: Scientific American, Inc., a division of Springer Nature Limited. ISSN: 1946-7087. doi:10.1038/scientificamerican0308-62. URL: http://www.cs.virginia.edu/~robins/The_Limits_of_Quantum_Computers.pdf (visited on 2021-08-10) (cit. on p. 6).

[2] David Lee Applegate, Robert E. Bixby, Vašek Chvátal, and William John Cook. *The Traveling Salesman Problem: A Computational Study*. 2nd ed. Vol. 17 of Princeton Series in Applied Mathematics. Princeton, NJ, USA: Princeton University Press, 2007. ISBN: 978-0-691-12993-8 (cit. on pp. 3, 6, 14).

[3] Thomas Bäck, David B. Fogel, and Zbigniew "Zbyszek" Michalewicz, eds. *Handbook of Evolutionary Computation*. Bristol, England, UK: IOP Publishing Ltd and Oxford, Oxfordshire, England, UK: Oxford University Press, 1997. ISBN: 978-0-7503-0392-7 (cit. on pp. 12, 14).

[4] Martin Beckmann and Tjalling Charles Koopmans. "Assignment Problems and the Location of Economic Activities". *Econometrica* 25(1):53–76, Jan. 1957. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd. ISSN: 0012-9682 (cit. on p. 2).

[5] Jacek Błażewicz, Wolfgang Domschke, and Erwin Pesch. "The Job Shop Scheduling Problem: Conventional and New Solution Techniques". *European Journal of Operational Research* 93(1):1–33, Aug. 1996. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/0377-2217(95)00362-2 (cit. on pp. 4, 13).

[6] Rainer E. Burkard, Eranda Çela, Panos Miltiades Pardalos, and Leonidas S. Pitsoulis. "The Quadratic Assignment Problem". In: *Handbook of Combinatorial Optimization*. Ed. by Panos Miltiades Pardalos, Ding-Zhu Du, and Ronald Lewis Graham. 1st ed. Boston, MA, USA: Springer, 1998, pp. 1713–1809. ISBN: 978-1-4613-7987-4. doi:10.1007/978-1-4613-0303-9_27 (cit. on pp. 2, 13).

[7] Eduardo Carvalho Pinto and Carola Doerr. *Towards a More Practice-Aware Runtime Analysis of Evolutionary Algorithms*. arXiv.org: Computing Research Repository (CoRR) abs/1812.00493. Ithaca, NY, USA: Cornell Universiy Library, Dec. 3, 2018. doi:10.48550/arXiv.1812.00493. URL: https://arxiv.org/abs/1812.00493 (visited on 2025-08-08). arXiv:1812.00493v1 [cs.NE] 3 Dec 2018 (cit. on p. 12).

[8] Bo Chen, Chris N. Potts, and Gerhard J. Woeginger. "A Review of Machine Scheduling: Complexity, Algorithms and Approximability". In: *Handbook of Combinatorial Optimization*. Ed. by Panos Miltiades Pardalos, Ding-Zhu Du, and Ronald Lewis Graham. 1st ed. Boston, MA, USA: Springer, 1998, pp. 1493–1641. ISBN: 978-1-4613-7987-4. doi:10.1007/978-1-4613-0303-9_25. See also pages 21–169 in volume 3/3 by Norwell, MA, USA: Kluwer Academic Publishers. (Cit. on pp. 6, 13).

[9] Jiayang Chen (陈嘉阳), Zhize Wu (吴志泽), Sarah Louise Thomson, and Thomas Weise (汤卫思). "Frequency Fitness Assignment: Optimization Without Bias for Good Solution Outperforms Randomized Local Search on the Quadratic Assignment Problem". In: *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. Nov. 20–22, 2024, Porto, Portugal. Ed. by Francesco Marcelloni, Kurosh Madani, Niki van Stein, and Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024, pp. 27–37. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0012888600003837 (cit. on pp. 12, 13).

[10] Philippe Chrétienne, Edward G. Coffman, Jan Karel Lenstra, and Zhen Liu, eds. *Scheduling Theory and Its Applications*. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., Sept. 1995. ISBN: 978-0-471-94059-3 (cit. on pp. 4, 13).

[11] Stephen Arthur Cook. "The Complexity of Theorem-Proving Procedures". In: *Third Annual ACM Symposium on Theory of Computing (STOC'1971)*. May 3–5, 1971, Shaker Heights, OH, USA. Ed. by Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman. New York, NY, USA: Association for Computing Machinery (ACM), 1971, pp. 151–158. ISBN: 978-1-4503-7464-4. doi:10.1145/800157.805047 (cit. on pp. 6, 13).

[12] William John Cook. *World TSP*. Waterloo, ON, Canada: University of Waterloo, 2013–Oct. 25, 2025. URL: http://www.math.uwaterloo.ca/tsp/world (visited on 2026-01-03) (cit. on pp. 6, 7).

[13] William John Cook, Daniel G. Espinoza, and Marcos Goycoolea. *Computing with Domino-Parity Inequalities for the TSP*. Tech. rep. Atlanta, GA, USA: Georgia Institute of TechnologyIndustrial and Systems Engineering, 2005. URL: https://www.math.uwaterloo.ca/~bico/papers/DP_paper.pdf (visited on 2026-01-03). See also [13]. (Cit. on p. 16).

[14] William John Cook, Daniel G. Espinoza, and Marcos Goycoolea. "Computing with Domino-Parity Inequalities for the Traveling Salesman Problem (TSP)". *INFORMS Journal on Computing* 19(3):356–365, Aug. 2007. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 1091-9856. doi:10.1287/ijoc.1060.0204. See also [13]. (Cit. on pp. 6, 7).

[15] Jim Davis, Thomas F. Edgar, James Porter, John Bernaden, and Michael Sarli. "Smart Manufacturing, Manufacturing Intelligence and Demand-Dynamic Performance". *Computers & Chemical Engineering* 47:145–156, Dec. 2012. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0098-1354. doi:10.1016/j.compchemeng.2012.06.037 (cit. on p. 5).

[16] Kenneth Alan De Jong. *Evolutionary Computation: A Unified Approach*. Vol. 4 of Complex Adaptive Systems. Cambridge, MA, USA: MIT Press, 2006. ISBN: 978-0-262-04194-2. URL: https://www.researchgate.net/publication/220740669 (visited on 2025-08-08) (cit. on p. 14).

[17] *Definition of Operations Research*. University of Western Ontario, London, ON, Canada: International Federation of Operational Research Societies (IFORS), 2020. URL: https://www.ifors.org/what-is-or (visited on 2026-01-01) (cit. on p. 13).

[18] Maxence Delorme, Manuel Iori, and Silvano Martello. "Bin Packing and Cutting Stock Problems: Mathematical Models and Exact Algorithms". *European Journal of Operational Research* 255(1):1–20, Nov. 2016. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/j.ejor.2016.04.030 (cit. on pp. 4, 12).

[19] Justin Dennison, Cherokee Boose, and Peter van Rysdam. *Intro to NumPy*. Centennial, CO, USA: ACI Learning. Birmingham, England, UK: Packt Publishing Ltd, June 2024. ISBN: 978-1-83620-863-1 (cit. on p. 13).

[20] Stefan Droste, Thomas Jansen, and Ingo Wegener. "On the Analysis of the $(1+1)$ Evolutionary Algorithm". *Theoretical Computer Science* 276(1-2):51–81, Apr. 2002. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0304-3975. doi:10.1016/S0304-3975(01)00182-7 (cit. on p. 12).

[21] Harald Dyckhoff and Ute Finke. *Cutting and Packing in Production and Distribution: A Typology and Bibliography*. Contributions to Management Science. Heidelberg, Baden-Württemberg, Germany: Physica-Verlag, 1992. ISSN: 1431-1941. ISBN: 978-3-642-63487-1. doi:10.1007/978-3-642-58165-6 (cit. on pp. 3, 4, 12).

[22] Kelly Easton, George L. Nemhauser, and Michael A. Trick. "The Traveling Tournament Problem Description and Benchmarks". In: *7th International Conference on Principles and Practice of Constraint Programming (CP'01)*. Nov. 26–Dec. 1, 2001, Paphos, Cyprus. Ed. by Toby Walsh. Vol. 2239 of Lecture Notes in Computer Science (LNCS). Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 2001, pp. 580–584. ISSN: 0302-9743. ISBN: 978-3-540-42863-3. doi:10.1007/3-540-45578-7_43 (cit. on p. 14).

[23] Michael R. Garey and David Stifler Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman and Company, 1979. ISBN: 978-0-7167-1045-5 (cit. on pp. 6, 14).

[24] David Edward Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1989. ISBN: 978-0-201-15767-3 (cit. on p. 14).

[25] Michael T. Goodrich. *A Gentle Introduction to NP-Completeness*. Irvine, CA, USA: University of California, Irvine, Apr. 2022. URL: https://ics.uci.edu/~goodrich/teach/cs165/notes/NPComplete.pdf (visited on 2025-08-01) (cit. on p. 13).

[26] Martin Grötschel, Michael Jünger, and Gerhard Reinelt. "Optimal Control of Plotting and Drilling Machines: A Case Study". *Zeitschrift für Operations Research (ZOR) – Methods and Models of Operations Research* 35(1):61–84, Jan. 1991. Heidelberg, Baden-Württemberg, Germany: Physica-Verlag. ISSN: 0340-9422. doi:10.1007/BF01415960 (cit. on p. 3).

[27] Gregory Z. Gutin and Abraham P. Punnen, eds. *The Traveling Salesman Problem and its Variations*. Vol. 12 of Combinatorial Optimization (COOP). New York, NY, USA: Springer New York, May 2002. ISSN: 1388-3011. doi:10.1007/b101971 (cit. on pp. 3, 6, 14).

[28] Charles R. Harris, K. Jarrod Millman, Stéfan van der Walt, Ralf Gommers, Pauli "pv" Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. "Array programming with NumPy". *Nature* 585:357–362, 2020. London, England, UK: Springer Nature Limited. ISSN: 0028-0836. doi:10.1038/S41586-020-2649-2 (cit. on p. 13).

[29] Keld Helsgaun. "General k-opt Submoves for the Lin–Kernighan TSP Heuristic". *Mathematical Programming Computation* 1(2-3):119–163, July 2009. London, England, UK: Springer Nature Limited. ISSN: 1867-2949. doi:10.1007/s12532-009-0004-6 (cit. on p. 7).

[30] Mario Hermann, Tobias Pentek, and Boris Otto. "Design Principles for Industrie 4.0 Scenarios". In: *49th Hawaii International Conference on System Sciences (HICSS)*. Jan. 5–8, 2016, Koloa, HI, USA. Ed. by Tung X. Bui and Ralph H. Sprague Jr. Los Alamitos, CA, USA: IEEE Computer Society, 2016, pp. 3928–3937. ISBN: 978-0-7695-5670-3. doi:10.1109/HICSS.2016.488 (cit. on p. 5).

[31] John Hunt. *A Beginners Guide to Python 3 Programming*. 2nd ed. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2023. ISBN: 978-3-031-35121-1. doi:10.1007/978-3-031-35122-8 (cit. on p. 13).

[32] John D. Hunter. "Matplotlib: A 2D Graphics Environment". *Computing in Science & Engineering* 9(3):90–95, May–June 2007. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 1521-9615. doi:10.1109/MCSE.2007.55 (cit. on p. 13).

[33] John D. Hunter, Darren Dale, Eric Firing, Michael Droettboom, and The Matplotlib Development Team. *Matplotlib: Visualization with Python*. Austin, TX, USA: NumFOCUS, Inc., 2012–2025. URL: https://matplotlib.org (visited on 2025-02-02) (cit. on p. 13).

[34] Robert Johansson. *Numerical Python: Scientific Computing and Data Science Applications with NumPy, SciPy and Matplotlib*. New York, NY, USA: Apress Media, LLC, Dec. 2018. ISBN: 978-1-4842-4246-9 (cit. on pp. 13, 14).

[35] Selmer Martin Johnson. "Optimal Two- and Three-Stage Production Schedules with Setup Times Included". *Naval Research Logistics Quarterly* I(1):61–68, Mar. 1954. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd. ISSN: 0894-069X. doi:10.1002/nav.3800010110. URL: https://www.rand.org/content/dam/rand/pubs/papers/2008/P402.pdf (visited on 2023-12-06) (cit. on pp. 5, 12).

[36] Richard "Dick" Manning Karp. "Reducibility Among Combinatorial Problems". In: *Complexity of Computer Computations. Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*. Mar. 20–22, 1972. Ed. by Raymond E. "Ray" Miller and James W. "Jim" Thatcher. New York, NY, USA: Springer New York, 1972, pp. 85–103. ISBN: 978-1-4684-2003-6. doi:10.1007/978-1-4684-2001-2_9 (cit. on p. 6).

[37] Tjalling Charles Koopmans and Martin Beckmann. "Assignment Problems and the Location of Economic Activities". *Econometrica* 25(1):53–76, 1957. New Haven, CT, USA: The Econometric Society and Chichester, West Sussex, England, UK: John Wiley and Sons Ltd. ISSN: 0012-9682. doi:10.2307/1907742 (cit. on p. 13).

[38] Eugene Leighton Lawler, Jan Karel Lenstra, Alexander Hendrik George Rinnooy Kan, and David B. Shmoys. "Sequencing and Scheduling: Algorithms and Complexity". In: *Production Planning and Inventory*. Ed. by Stephen C. Graves, Alexander Hendrik George Rinnooy Kan, and Paul H. Zipkin. Vol. IV of Handbooks of Operations Research and Management Science. Amsterdam, The Netherlands: Elsevier B.V., 1993. Chap. 9, pp. 445–522. ISSN: 0927-0507. ISBN: 978-0-444-87472-6. doi:10.1016/S0927-0507(05)80189-6. URL: http://alexandria.tue.nl/repository/books/339776.pdf (visited on 2023-12-06) (cit. on pp. 4, 6, 13).

[39] Eugene Leighton Lawler, Jan Karel Lenstra, Alexander Hendrik George Rinnooy Kan, and David B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Estimation, Simulation, and Control – Wiley-Interscience Series in Discrete Mathematics and Optimization. Chichester, West Sussex, England, UK: Wiley Interscience, Sept. 1985. ISSN: 0277-2698. ISBN: 978-0-471-90413-7 (cit. on pp. 3, 6, 14).

[40] Kent D. Lee and Steve Hubbard. *Data Structures and Algorithms with Python*. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2015. ISBN: 978-3-319-13071-2. doi:10.1007/978-3-319-13072-9 (cit. on p. 13).

[41] Tianyu Liang (梁天宇), Zhize Wu (吴志泽), Jörg Lässig, Daan van den Berg, Sarah Louise Thomson, and Thomas Weise (汤卫思). "Addressing the Traveling Salesperson Problem with Frequency Fitness Assignment and Hybrid Algorithms". *Soft Computing* 28(17-18):9495–9508, July 2024. London, England, UK: Springer Nature Limited. ISSN: 1432-7643. doi:10.1007/S00500-024-09718-8 (cit. on pp. 13, 14).

[42] Tianyu Liang (梁天宇), Zhize Wu (吴志泽), Jörg Lässig, Daan van den Berg, and Thomas Weise (汤卫思). "Solving the Traveling Salesperson Problem using Frequency Fitness Assignment". In: *IEEE Symposium Series on Computational Intelligence (SSCI'2022)*. Dec. 4–7, 2022, Singapore. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 2022. ISBN: 978-1-6654-8769-6. doi:10.1109/SSCI51031.2022.10022296 (cit. on p. 13).

[43] Tianyu Liang (梁天宇), Zhize Wu (吴志泽), Matthias Thürer, Markus Wagner, and Thomas Weise (汤卫思). "Generating Small Instances with Interesting Features for the Traveling Salesperson Problem". In: *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. Nov. 20–22, 2024, Porto, Portugal. Ed. by Francesco Marcelloni, Kurosh Madani, Niki van Stein, and Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024, pp. 173–180. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0012888800003837 (cit. on p. 13).

[44] Andrea Lodi, Silvano Martello, and Michele Monaci. "Two-Dimensional Packing Problems: A Survey". *European Journal of Operational Research* 141(2):241–252, Sept. 2002. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/S0377-2217(02)00123-6 (cit. on pp. 4, 12).

[45] Eliane Maria Loiola, Nair Maria Maia de Abreu, Paulo Oswaldo Boaventura-Netto, Peter M. Hahn, and Tania Querido. "A Survey for the Quadratic Assignment Problem". *European Journal of Operational Research* 176(2):657–690, 2007. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/j.ejor.2005.09.032 (cit. on p. 13).

[46] Mark Lutz. *Learning Python*. 6th ed. Sebastopol, CA, USA: O'Reilly Media, Inc., Mar. 2025. ISBN: 978-1-0981-7130-8 (cit. on p. 13).

[47] Francesco Marcelloni, Kurosh Madani, Niki van Stein, and Joaquim Filipe, eds. *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. Nov. 20–22, 2024, Porto, Portugal. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0000195000003837.

[48] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley Series in Discrete Mathematics and Optimization. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., 1990. ISBN: 978-0-471-92420-3. URL: http://www.or.deis.unibo.it/knapsack.html (visited on 2023-12-07) (cit. on p. 4).

[49] Zbigniew "Zbyszek" Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 1996. ISBN: 978-3-540-58090-4. doi:10.1007/978-3-662-03315-9 (cit. on p. 14).

[50] Zbigniew "Zbyszek" Michalewicz, Leonardo Arantes, and Matthew Michalewicz. *The Rise of Artificial Intelligence: Real-world Applications for Revenue and Margin Growth*. Ormond, VIC, Australia: Hybrid Publishers, 2021. ISBN: 978-1-925736-62-5 (cit. on p. 1).

[51] Zbigniew "Zbyszek" Michalewicz and David B. Fogel. *How to Solve It: Modern Heuristics*. 2nd ed. Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 2004. ISBN: 978-3-540-22494-5. doi:10.1007/978-3-662-07807-5 (cit. on p. 7).

[52] Melanie Mitchell. *An Introduction to Genetic Algorithms*. Complex Adaptive Systems. Cambridge, MA, USA: MIT Press, Feb. 1998. ISBN: 978-0-262-13316-6. URL: http://boente.eti.br/fuzzy/ebook-fuzzy-mitchell.pdf (visited on 2025-08-08) (cit. on p. 14).

[53] Yuichi Nagata and Shigenobu Kobayashi. "A Powerful Genetic Algorithm using Edge Assembly Crossover for the Traveling Salesman Problem". *INFORMS Journal on Computing* 25(2):346–363, Spr. 2013. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 1091-9856. doi:10.1287/ijoc.1120.0506 (cit. on p. 7).

[54] NumPy Team. *NumPy*. San Francisco, CA, USA: GitHub Inc and Austin, TX, USA: NumFOCUS, Inc. URL: https://numpy.org (visited on 2025-02-02) (cit. on p. 13).

[55] Ashwin Pajankar. *Hands-on Matplotlib: Learn Plotting and Visualizations with Python 3*. New York, NY, USA: Apress Media, LLC, Nov. 2021. ISBN: 978-1-4842-7410-1 (cit. on p. 13).

[56] Panos Miltiades Pardalos, Ding-Zhu Du, and Ronald Lewis Graham, eds. *Handbook of Combinatorial Optimization*. 1st ed. Boston, MA, USA: Springer, 1998. ISBN: 978-1-4613-7987-4.

[57] Yasset Pérez-Riverol, Laurent Gatto, Rui Wang, Timo Sachsenberg, Julian Uszkoreit, Felipe da Veiga Leprevost, Christian Fufezan, Tobias Ternent, Stephen J. Eglen, Daniel S. Katz, Tom J. Pollard, Alexander Konovalov, Robert M. Flight, Kai Blin, and Juan Antonio Vizcaíno. "Ten Simple Rules for Taking Advantage of Git and GitHub". *PLOS Computational Biology* 12(7), July 14, 2016. San Francisco, CA, USA: Public Library of Science (PLOS). ISSN: 1553-7358. doi:10.1371/JOURNAL.PCBI.1004947 (cit. on p. 12).

[58] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. 5th ed. Cham, Switzerland: Springer, 2016. ISBN: 978-3-319-26578-0. doi:10.1007/978-3-319-26580-3 (cit. on p. 4).

[59] Chris N. Potts and Vitaly A. Strusevich. "Fifty Years of Scheduling: A Survey of Milestones". *The Journal of the Operational Research Society (JORS)* 60(sup1):S41–S68: *Special Issue: Milestones in OR*, 2009. London, England, UK: Taylor and Francis Ltd. ISSN: 0160-5682. doi:10.1057/jors.2009.2. URL: http://eprints.soton.ac.uk/145495/1/Scheduling50YearsE-Print.pdf (visited on 2023-12-06) (cit. on p. 4).

[60] Sanatan Rai and George Vairaktarakis. "NP-Complete Problems and Proof Methodology". In: *Encyclopedia of Optimization*. Ed. by Christodoulos A. Floudas and Panos Miltiades Pardalos. 2nd ed. Boston, MA, USA: Springer, Sept. 2008, pp. 2675–2682. ISBN: 978-0-387-74758-3. doi:10.1007/978-0-387-74759-0_462 (cit. on p. 13).

[61] Sartaj Sahni and Teofilo Francisco Gonzalez Arce. "P-complete Approximation Problems". *Journal of the ACM (JACM)* 23(3):555–565, July 1976. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0004-5411. doi:10.1145/321958.321975. URL: https://sites.cs.ucsb.edu/~teo/papers/JACM-PCom.pdf (visited on 2026-02-21) (cit. on pp. 3, 13).

[62] Guntram Scheithauer. *Introduction to Cutting and Packing Optimization: Problems, Modeling Approaches, Solution Methods*. Vol. 263 of International Series in Operations Research & Management Science (ISOR). Cham, Switzerland: Springer, 2018. ISSN: 0884-8289. ISBN: 978-3-319-64402-8. doi:10.1007/978-3-319-64403-5 (cit. on pp. 3, 12).

[63] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge, England, UK: Cambridge University Press (CUP), July 2014. ISBN: 978-1-107-05713-5. URL: http://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning (visited on 2024-06-27) (cit. on p. 13).

[64] Anna Skoulikari. *Learning Git*. Sebastopol, CA, USA: O'Reilly Media, Inc., May 2023. ISBN: 978-1-0981-3391-7 (cit. on p. 12).

[65] Leon Steinberg. "The Backboard Wiring Problem: A Placement Algorithm". *SIAM Review* 3(1):37–50, Jan. 1961. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics Publications (SIAM). ISSN: 0036-1445. doi:10.1137/1003003 (cit. on p. 3).

[66] Éric D. Taillard. "Benchmarks for Basic Scheduling Problems". *European Journal of Operational Research* 64(2):278–285, Jan. 1993. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/0377-2217(93)90182-M. URL: https://arodes.hes-so.ch/record/8224/files/Preprint.pdf (visited on 2026-02-20) (cit. on pp. 4, 13).

[67] Sarah Louise Thomson, Gabriela Ochoa, Daan van den Berg, Tianyu Liang (梁天宇), and Thomas Weise (汤卫思). "Entropy, Search Trajectories, and Explainability for Frequency Fitness Assignment". In: *Parallel Problem Solving from Nature (PPSN XVIII)*. Vol. 1. Sept. 14–18, 2024, Hagenberg, Mühlkreis, Austria. Ed. by Michael Affenzeller, Stephan M. Winkler, Anna V. Kononova, Heike Trautmann, Tea Tušar, Penousal Machado, and Thomas Bäck. Vol. 15148 of Lecture Notes in Computer Science (LNCS). Cham, Switzerland: Springer. ISSN: 0302-9743. ISBN: 978-3-031-70054-5. doi:10.1007/978-3-031-70055-2_23 (cit. on p. 13).

[68] Mariot Tsitoara. *Beginning Git and GitHub: Version Control, Project Management and Teamwork for the New Developer*. New York, NY, USA: Apress Media, LLC, Mar. 2024. ISBN: 979-8-8688-0215-7 (cit. on pp. 12, 14).

[69] Kristian Verduin, Sarah Louise Thomson, and Daan van den Berg. "Too Constrained for Genetic Algorithms. Too Hard for Evolutionary Computing. The Traveling Tournament Problem". In: *15th International Joint Conference on Computational Intelligence (IJCCI'23)*. Nov. 13–15, 2023, Rome, Italy. Ed. by Niki van Stein, Francesco Marcelloni, H. K. Lam, Marie Cottrell, and Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2023, pp. 246–257. ISSN: 2184-3236. ISBN: 978-989-758-674-3. doi:10.5220/0012192100003595 (cit. on p. 14).

[70] Pauli "pv" Virtanen, Ralf Gommers, Travis E. Oliphant, Matt "mdhaber" Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, Ilhan "ilayn" Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". *Nature Methods* 17:261–272, Mar. 2, 2020. London, England, UK: Springer Nature Limited. ISSN: 1548-7091. doi:10.1038/s41592-019-0686-2. URL: http://arxiv.org/abs/1907.10121 (visited on 2024-06-26). See also arXiv:1907.10121v1 [cs.MS] 23 Jul 2019. (Cit. on p. 14).

[71] Thomas Weise (汤卫思). *Global Optimization Algorithms – Theory and Application*. self-published, 2009. URL: https://www.researchgate.net/publication/200622167 (visited on 2025-07-25) (cit. on pp. ii, 7, 12, 14).

[72] Thomas Weise (汤卫思). *Programming with Python*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), 2024–2025. URL: https://thomasweise.github.io/programmingWithPython (visited on 2025-01-05) (cit. on pp. ii, 13).

[73] Thomas Weise (汤卫思), Raymond Chiong, Jörg Lässig, Ke Tang (唐珂), Shigeyoshi Tsutsui, Wenxiang Chen (陈文祥), Zbigniew "Zbyszek" Michalewicz, and Xin Yao (姚新). "Benchmarking Optimization Algorithms: An Open Source Framework for the Traveling Salesman Problem". *IEEE Computational Intelligence Magazine (CIM)* 9(3):40–52, Aug. 2014. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 1556-603X. doi:10.1109/MCI.2014.2326101 (cit. on p. 14).

[74] Thomas Weise (汤卫思), Alexander Podlich, and Christian Gorldt. "Solving Real-World Vehicle Routing Problems with Evolutionary Algorithms". In: *Natural Intelligence for Scheduling, Planning and Packing Problems*. Ed. by Raymond Chiong and Sandeep Dhakal. Vol. 250 of Studies in Computational Intelligence (SCI). Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, Sept. 2009. Chap. 2, pp. 29–53. ISSN: 1860-949X. ISBN: 978-3-642-04038-2. doi:10.1007/978-3-642-04039-9_2 (cit. on p. 3).

[75] Thomas Weise (汤卫思), Alexander Podlich, Kai Reinhard, Christian Gorldt, and Kurt Geihs. "Evolutionary Freight Transportation Planning". In: *Applications of Evolutionary Computing (EvoWorkshops 2009): Proceedings of EvoCOMNET, EvoENVIRONMENT, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, EvoNUM, EvoSTOC, EvoTRANSLOG*. Apr. 15–17, 2009, Tübingen, Baden-Württemberg, Germany. Ed. by Mario Giacobini, Anthony Brabazon, Stefano Cagnoni, Gianni A. Caro, Anikó Ekárt, Anna Isabel Esparcia-Alcázar, Muddassar Farooq, Andreas Fink, and Penousal Machado. Vol. 5484 of Theoretical Computer Science and General Issues (LNTCS), sub-series of Lecture Notes in Computer Science (LNCS). Berlin/Heidelberg, Germany: Springer-Verlag GmbH

Germany, Apr. 2009, pp. 768–777. ISSN: 2512-2010. doi:10.1007/978-3-642-01129-0_87 (cit. on p. 3).

[76] Thomas Weise (汤卫思) and Zhize Wu (吴志泽). "Replicable Self-Documenting Experiments with Arbitrary Search Spaces and Algorithms". In: *Conference on Genetic and Evolutionary Computation (GECCO'2023), Companion Volume*. July 15–19, 2023, Lisbon, Portugal. Ed. by Sara Silva and Luís Paquete. New York, NY, USA: Association for Computing Machinery (ACM), 2023, pp. 1891–1899. ISBN: 979-8-4007-0120-7. doi:10.1145/3583133.3596306 (cit. on p. 13).

[77] Thomas Weise (汤卫思), Yuezhong Wu (吴越钟), Raymond Chiong, Ke Tang (唐珂), and Jörg Lässig. "Global versus Local Search: The Impact of Population Sizes on Evolutionary Algorithm Performance". *Journal of Global Optimization* 66(3):511–534, Feb. 2016. London, England, UK: Springer Nature Limited. ISSN: 0925-5001. doi:10.1007/s10898-016-0417-5 (cit. on p. 14).

[78] Thomas Weise (汤卫思), Yuezhong Wu (吴越钟), Weichen Liu (刘伟臣), and Raymond Chiong. "Implementation Issues in Optimization Algorithms: Do They Matter?" *Journal of Experimental & Theoretical Artificial Intelligence (JETAI)* 31(4):533–554, 2019. London, England, UK: Taylor and Francis Ltd. ISSN: 0952-813X. doi:10.1080/0952813X.2019.1574908 (cit. on p. 7).

[79] L. Darrell Whitley. "Blind No More: Deterministic Partition Crossover and Deterministic Improving Moves". In: *Genetic and Evolutionary Computation Conference Companion (GECCO'2018)*. July 15–19, 2018, Kyoto, Japan. Ed. by Hernán E. Aguirre and Keiki Takadama. New York, NY, USA: Association for Computing Machinery (ACM), 2018, pp. 515–532. ISBN: 978-1-4503-5764-7. doi:10.1145/2908961.2926987 (cit. on p. 7).

[80] L. Darrell Whitley. "The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best". In: *3rd International Conference on Genetic Algorithms (ICGA'1989)*. June 1989, Fairfax, VA, USA: George Mason University. Ed. by J. David Schaffer. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, 1989, pp. 116–123. ISBN: 978-1-55860-066-9. URL: https://www.researchgate.net/publication/2527551 (visited on 2025-08-08) (cit. on p. 14).

[81] CAO Xiang (曹翔), Zhize Wu (吴志泽), Daan van den Berg, and Thomas Weise (汤卫思). "Randomized Local Search vs. NSGA-II vs. Frequency Fitness Assignment on The Traveling Tournament Problem". In: *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. Nov. 20–22, 2024, Porto, Portugal. Ed. by Francesco Marcelloni, Kurosh Madani, Niki van Stein, and Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024, pp. 38–49. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0012891500003837 (cit. on pp. 13, 14).

[82] Rui Zhao (赵睿), Tianyu Liang (梁天宇), Zhize Wu (吴志泽), Daan van den Berg, Matthias Thürer, and Thomas Weise (汤卫思). "Randomized Local Search on the 2D Rectangular Bin Packing Problem with Item Rotation". In: *Genetic and Evolutionary Computation Conference (GECCO'2024)*. July 14–18, 2024, Melbourne, VIC, Australia. Ed. by Xiaodong Li and Julia Handl. New York, NY, USA: Association for Computing Machinery (ACM), 2024, pp. 235–238. ISBN: 979-8-4007-0494-9. doi:10.1145/3638530.3654139 (cit. on p. 13).

[83] Rui Zhao (赵睿), Zhize Wu (吴志泽), Daan van den Berg, Matthias Thürer, Tianyu Liang (梁天宇), Ming Tan (檀明), and Thomas Weise (汤卫思). "Randomized Local Search for Two-Dimensional Bin Packing and a Negative Result for Frequency Fitness Assignment". In: *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. Nov. 20–22, 2024, Porto, Portugal. Ed. by Francesco Marcelloni, Kurosh Madani, Niki van Stein, and Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024, pp. 15–26. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0012888500003837 (cit. on pp. 4, 12, 13).