Master's thesis

HYBRID APPROACHES FOR COMBINATORIAL OPTIMIZATION PROBLEMS

Bc. Michal Lukeš

Faculty of Electrical Engineering
Department of Computer Science
Supervisor: Ing. Josef Grus
May 24, 2024

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Lukeš Michal** Personal ID number: **474572**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Data Science**

II. Master’s thesis details

Master’s thesis title in English:

Hybrid approaches for combinatorial optimization problems

Master’s thesis title in Czech:

Hybridní přístupy pro řešení kombinatorických optimalizačních úloh

Guidelines:

The student will study Vehicle Routing Problem with Time Windows (VRPTW) and the Nurse Rostering Problem (NRP) and formulate the problems using Constraint Programming. The student will choose, design, implement and verify appropriate metaheuristic algorithms for the combinatorial optimization problems (e.g., Simulated Annealing) such that the algorithm is easily extendable to other combinatorial problems. Then, the student will focus on hybrid solution methods, making use of Constraint Programming, metaheuristics, and local search. The student will compare its results with the state-of-the-art results in the literature, using relevant standardized benchmarks for each of the studied problems.
The student will integrate its benchmarks, algorithm, and evaluation/visualization components into the General optimization platform (https://github.com/Omastto1/General-Optimization-Solver), which includes several solvers and components to formulate, solve, and evaluate different combinatorial problems.

Bibliography / sources:

Ngoo, Chong Man, Say Leng Goh, San Nah Sze, Nasser R. Sabar, Salwani Abdullah, and Graham Kendall. “A Survey of the Nurse Rostering Solution Methodologies: The State-of-the-Art and Emerging Trends.” IEEE Access 10 (2022): 56504–24. https://doi.org/10.1109/access.2022.3177280.
Joshi, Kanchan, Karuna Jain, and Vijay Bilolikar. “A VNS-Ga-Based Hybrid Metaheuristics for Resource Constrained Project Scheduling Problem.” International Journal of Operational Research 27, no. 3 (2016): 437. https://doi.org/10.1504/ijor.2016.078938.
Subramanian, Anand, Eduardo Uchoa, and Luiz Satoru Ochi. “A Hybrid Algorithm for a Class of Vehicle Routing Problems.” Computers & Operations Research 40, no. 10 (2013): 2519–31. https://doi.org/10.1016/j.cor.2013.01.013.

Name and workplace of master's thesis supervisor:

Ing. Josef Grus Department of Control Engineering FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **06.09.2023** Deadline for master's thesis submission: **24.05.2024**

Assignment valid until: **16.02.2025**

Ing. Josef Grus Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Czech Technical University in Prague
Faculty of Electrical Engineering

Citation of this thesis: Lukeš Michal. *Hybrid approaches for combinatorial optimization problems.* Master's thesis. Czech Technical University in Prague, Faculty of Electrical Engineering, 2024.

Contents

List of Figures

List of Tables

Declaration

I declare that I elaborated this thesis on my own and that I mentioned all the information sources that have been used in accordance with the Guideline for adhering to ethical principles in the course of elaborating an academic final thesis.

In Praze on May 24, 2024

Abstract

The goal of this thesis is to research, implement, verify and compare solvers for the Vehicle Routing Problem with Time Windows and the Nurse Rostering Problem using tools for mathematical optimization like Mixed Integer Linear Programming and Constraint Programming, Metaheuristics or Localsearch and ultimately combine them in a hybrid solution method that benefits from their individual advantages. The tools and results developed are available online to enable the community to expand on this work easily by developing and comparing their solutions.

Keywords VRP, Vehicle Routing Problem, NRP, Nurse Rostering Problem, CVRPTW, Capacitated Vehicle Routing Problem With Time Windows, Solomon, Benchmark, Mixed Integer Linear Programming, Combinatorial Optimization, Genetic Algorithms, Constraint Programming, Metaheuristic

Abstrakt

Cílem této práce je prozkoumat, implementovat, ověřovit a porovnat řešiče pro problém vozového parku s časovými okny (Vehicle Routing Problem with Time Windows) a problém rozvrhu sester (Nurse Rostering Problem) pomocí nástrojů pro matematickou optimalizaci jako je Smíšené Celočíselné Lineární Programování (Mixed Integer Linear Programming) a Programování s Omezeními (Constraint Programming), Metaheuristiky nebo Lokální Vyhledávání a nakonec je zkombinovat v hybridní metody řešení která kombinuje jejich silné stránky. Vyvinuté nástroje a výsledky jsou dostupné online, aby komunita mohla snadno rozšířit tuto práci dalším vývojem a porovnáváním vlastních řešení.

Klíčová slova VRP, Vehicle Routing problém, NRP, Nurse Rostering problém, CVRPTW, Capacitated Vehicle Routing problém s Časovými Okny, Solomon Benchmark, Smíšené Celočíselné Lineární Programování, Kombinatorická Optimalizace, Genetické Algoritmy, Programování s Omezeními, Metaheuristiky

List of abbreviations

TSP	Traveling Salesman Problem
VRP	Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Windows
CVRPTW	Capacitated Vehicle Routing Problem with Time Windows
NRP	Nurse Rostering Problem
LP	Linear Programming
ILP	Integer Linear Programming
CP	Constraint Programming
MIP	Mixed Integer Programming
MILP	Mixed Integer Linear Programming
QP	Quadratic Programming
GA	Genetic Algorithm
SA	Simulated Annealing
PSO	Particle Swarm Optimization
ALNS	Adaptive Large Neighborhood Search
HCO	Hill-Climbing Optimization

Introduction

Combinatorial optimization lies at the intersection of mathematics, computer science, and operations research, focusing on finding the best solution from a finite set of possibilities. It deals with problems where the goal is to optimize one or more objective functions subject to a set of constraints. Unlike in the case of continuous optimization, the variables can be limited to discrete values in the case of combinatorial optimization. An exhaustive search of all valid options is usually not feasible, requiring specialized algorithms designed to explore the solution space efficiently.

Combinatorial optimization is present in logistics, transportation, and personnel scheduling. Since these problems usually originate from real-life problems, a better solver can help save time, money and other resources. Therefore, it is essential to implement, validate and compare new proposed algorithms quickly. For this reason, this work collaborates with Tomáš Omasta and extends [LO23].

One typical real-life example of the combinatorial optimization problem is the Traveling Salesman problem, which, given a list of cities and the distances between each pair of cities, aims to find the shortest path such that each city is visited exactly once and which starts and ends in the origin city. It is an NP-hard problem, meaning it can not be solved in polynomial time. The vehicle routing problem (VRP) generalises Traveling Salesman by adding multiple "salesmen" renamed to vehicles in this case. Also, the towns are now called customers instead. Vehicle routing can be further expanded by adding additional constraints such as maximum capacity, which each vehicle can carry, pickup and delivery, time windows, or vehicle battery capacity. Combining constraints allows for a better simulation of the real world for a particular application, which is also common. By finding optimal or near-optimal solutions, organizations can minimize costs, improve operational efficiency, enhance customer satisfaction, and help fight against climate change by using less fossil fuels.

Another such problem is the nurse rostering problem (NRP), sometimes called the nurse scheduling problem, which focuses on finding an optimal way to assign nurses to shifts, typically with a set of hard constraints which all valid solutions must follow, and a set of soft constraints which define the relative quality of valid solutions. This problem has many formalizations, each suited to its particular application.

Collaboration with Tomáš Omasta has been instrumental in implementing a Python interface, providing a convenient platform for comparing various optimization approaches. The interface, available at [LO23], enables easy experimentation and benchmarking of

different algorithms and problem instances. His work can be found at [Oma24].

Several tools and frameworks are utilized to accomplish the project's goals. These include Cplex[Cpl22], a widely-used commercial solver known for its capabilities in solving Constraint Programming (CP) problems, and OR-Tools[PF23], an open-source library that offers various optimization algorithms and solvers. These frameworks are used to implement mathematical optimization in several formulations for the mentioned problems. Pymoo [BD20] is used for its implementation of several metaheuristics and hyperparameter searches. We implement a chromosome representation using several local search decoders for these metaheuristics. The performance of these methods was compared and studied. The best mathematical optimization solver is combined with the best metaheuristic for the best hybrid solver our other implementations offer.

Chapter 1
Background

To properly understand our work, we must first introduce the topics on which we build. We introduce some exact approaches for solving combinatorial optimization problems and then compare them against heuristic and other methods.

1.1 Modern History of Operational Research

Operations research gained prominence during World War II to address military logistics and decision-making problems. Notable contributions include the simplex algorithm for linear programming [Dan90].

One significant area of research in combinatorial optimization is the application of Integer Linear Programming (ILP) formulations. Dantzig and Fulkerson's [DFJ54] seminal work in 1954 on the transportation problem laid the foundation for using linear programming techniques to solve optimization problems. Subsequently, many researchers have focused on formulating combinatorial optimization problems as ILP models and developing efficient algorithms to solve them.

Papers such as [Joh73] introduced approximation algorithms for Graph Coloring, TSP or bin packing, while [Glo86] pioneered the field of metaheuristics with Tabu Search.

1.2 Exact approaches

In this section, we explore exact approaches to optimization, focusing on Integer Linear Programming and Constraint Programming, which guarantee to find optimal solutions by exhaustively exploring the solution space within defined constraints, focusing on finding the best solution at the expense of computational time. These methods form the foundation of many robust optimization frameworks.

1.2.1 Integer Linear Programming

ILP is a mathematical modelling technique that formulates optimization problems as a system of linear equalities and inequalities with integer or binary decision variables. ILP problems can be relaxed into LP problems by allowing the integer constraints to

be continuous. This relaxation can be solved using the Simplex method. The simplex method works by iteratively moving from one vertex of the feasible region to another along edges, seeking to optimize the objective function. It operates by selecting a pivot element and performing row operations to pivot towards an improved solution until reaching an optimal vertex where no further improvement is possible. Using linear programming relaxation, we can transform an NP-hard optimization problem (integer programming) into a related problem that is solvable in polynomial time (linear programming). The solution to the relaxed linear program can be used to gain information about the solution to the original integer program. Automated solvers for ILP rely on mathematical optimization algorithms, such as the branch-and-bound method, to efficiently explore the solution space and find the best feasible solution. ILP is particularly effective when the problem can be accurately modelled using linear equations and the objective function, and constraints can be expressed as linear functions.

1.2.2 Constraint Programming

Constraint Programming is a declarative programming paradigm that focuses on modelling and reasoning about constraints. It leverages constraint propagation techniques to iteratively reduce the search space by enforcing local consistency and exploiting problem-specific properties. Unlike ILP, CP allows for a more flexible and intuitive representation of problems by explicitly defining the constraints that must be satisfied. For example, in a scheduling problem where tasks have dependencies, time windows, and resource constraints (e.g., no two tasks can use the same resource at the same time), CP can naturally express these constraints using global constraints like all_different, cumulative, and element. Among the popular domains are boolean, integer, interval and mixed domains, which combine two or more simple domains. CP solvers can importantly handle both single-objective and multi-objective optimization problems, offering a powerful tool for addressing real-world challenges across various industries.

1.2.3 Comparison

The strengths of ILP lie in its ability to handle large-scale linear optimization problems with well-defined objective functions and linear constraints. ILP solvers, such as Gurobi[Gur23], are highly optimized and can efficiently explore the solution space.

On the other hand, CP offers several advantages in specific scenarios. One key strength of CP is its ability to handle non-linear and non-convex constraints. It excels in modelling complex combinatorial problems with various types of constraints, such as scheduling, sequencing, and resource allocation problems. CP's declarative nature allows for a more natural representation of the problem, making capturing and expressing intricate constraints easier.

Moreover, CP's constraint propagation techniques enable the system to reason about the constraints and perform early pruning of infeasible regions of the search space. This pruning significantly reduces the search space and can lead to faster convergence to optimal or near-optimal solutions. CP's ability to handle discrete decision variables and combinatorial structures makes it well-suited for problems involving discrete choices and combinatorial optimization.

However, it is essential to note that the choice between ILP and CP depends on the specific characteristics of the problem at hand. Some problems may be more naturally formulated using ILP, while others may lend themselves better to CP modelling. Understanding the strengths and weaknesses of each approach allows for informed decision-making when selecting the appropriate technique for solving combinatorial optimization problems.

For example, [VHA11] found that their best implementation of CP outperforms their best ILP method "by a factor of 5 to 50" on a mix of randomly generated instances and benchmarks from other literature on the no-wait job shop problem. The job shop problem involves scheduling a set of jobs, each with a specific sequence of tasks that must be processed on different machines. The no-wait job shop problem adds the constraint that each job must move directly from one machine to the next without any waiting time between tasks, increasing the complexity of scheduling.

On the other hand, [FLS15] shows that the ILP approach significantly outperforms CP for the Lazy Bureaucrat Problem, evaluated on randomly generated instances using a standard instance generator. The Lazy Bureaucrat Problem involves scheduling tasks for a bureaucrat who prefers to minimize their work by taking the longest possible breaks between tasks. The challenge is to find an optimal schedule that maximizes these breaks while ensuring all tasks are completed within given constraints.

1.2.4 Mixed Integer Programming

Mixed Integer Programming (MIP) combines the capabilities of both ILP and CP by allowing for both integer and continuous variables within the same optimization model.

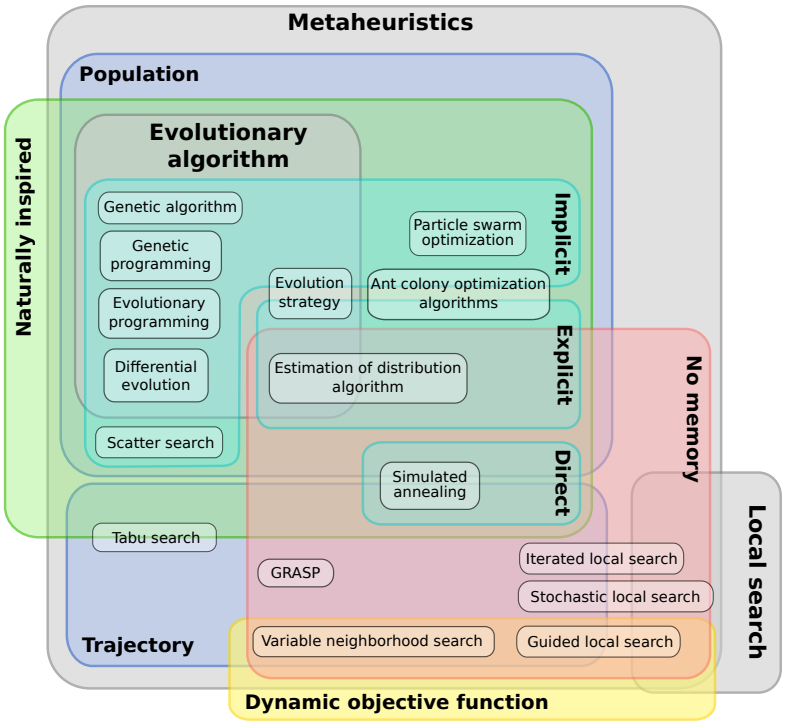
Only some advanced solvers allow the combination of constraints from both approaches. While this allows for more options during modeling of the problem it does not necesarily benefit the performance as shown by [KB16] again on a combination of own generated instances and some public benchmarks, this time for the classical job shop scheduling problem.

1.3 Heuristic Approaches

Heuristic algorithms are designed to find suitable solutions to optimization problems in a reasonable amount of time without guaranteeing optimality. These methods often involve iterative improvement techniques and search algorithms that explore the solution space by making local modifications to the current solution. Heuristics can be effective when dealing with large-scale problems where finding the optimal solution is computationally infeasible. Heuristics are often problem-dependent.

Metaheuristics are problem-independent techniques that can be applied to various problems. A metaheuristic knows nothing about the problem it will be applied to. It can treat functions as black boxes. A large number of metaheuristics exist, as well as a large number of their classifications. For example, based on the origin of their inception, they can be classified as evolutionary (Genetic Algorithm, Differential Evaluation, Harmony search...), trajectory-based (Simulated Annealing, Tabu Search, Variable Neighbourhood Search, ...) or nature-inspired (Particle Swarm Optimization, Grey Wolf Optimizer, Cultural Algorithm, ...), but there are also multiple views on this specific classification.

Figure 1.1 Euler diagram of the different classifications of metaheuristics from [DC]



Based on the type of search strategy, they can be classified as local search (Hill Climbing method), global search (GA and PSO) or both (SA and TS). Another classification is based on whether the algorithm improves a single solution (SA) or multiple (PSO).

1.3.1 Simulated Annealing

Simulated Annealing relies on a model [Met+53] for simulating the physical annealing process, where particles of a solid arrange themselves into a thermal equilibrium. It searches for a good solution by iteratively making minor changes to the current solution and accepting modifications based on a probability that decreases over time. This allows the algorithm to escape local optima early on and converge to a good solution as the ”temperature” lowers.

1.3.2 Particle Swarm Optimization

Particle Swarm Optimization is a population-based metaheuristic inspired by the social behaviour of birds and fish, initially proposed by [EK95]. It involves a swarm of particles (potential solutions) that move through the solution space, guided by their own best-known positions and the best-known positions of their neighbours. The particles iteratively adjust their positions to find the optimal solution.

1.3.3 Differential Evolution

Differential Evolution is a population-based optimization algorithm that uses differences between randomly selected pairs of solutions to evolve the population. By combining existing solutions and introducing mutations, DE explores the solution space efficiently. It's beneficial for continuous optimization problems and excels in maintaining diversity within the population.

1.3.4 Biased Random Key Genetic Algorithm

The Biased Random Key Genetic Algorithm is a variation of the genetic algorithm that represents solutions as vectors of random keys (real numbers). It biases the selection process towards better solutions, promoting faster convergence. BRKGA is effective for both combinatorial and continuous optimization problems, leveraging the strengths of genetic algorithms while incorporating a bias to enhance performance.

1.4 Other Approaches

1.4.1 Local Search

Local search algorithms focus on iteratively improving a given solution by exploring neighbouring solutions in the search space. These algorithms start with an initial solution and make small changes, evaluating the resulting solution's quality. If the modified solution improves the objective function, it is accepted as the new current solution. Local search algorithms continue this process until no further improvements can be made. Local search is particularly useful for problems with a large solution space and complex constraints where global optimization is challenging. Examples of local search algorithms include Hill Climbing, Genetic Local Search, and Variable Neighborhood Search.

1.4.2 Decomposition Techniques

Decomposition techniques [Con+06] divide complex optimization problems into smaller, more manageable subproblems. By decomposing the problem, it becomes possible to solve the subproblems independently or in a coordinated manner. This approach is beneficial for large-scale problems where the entire problem cannot be solved in a reasonable amount of time. Decomposition techniques can include methods such as Benders' decomposition, Lagrangian relaxation, and column generation.

1.4.3 Hybrid Approaches

Hybrid approaches combine two or more optimization techniques to leverage their respective strengths. These approaches aim to balance exploration and exploitation of the search space by integrating different optimization methods.

For example, a hybrid approach may combine ILP with local search or metaheuristics to benefit from the precision of ILP and its superior ability to find the global minimum

while using heuristics to improve search efficiency at the start of the search. Hybrid approaches can be tailored to specific problem characteristics and potentially provide superior performance compared to individual methods.

Another example is combining different metaheuristics. Since some of the techniques have excellent global search capabilities, they can be combined with other methods that perform better at local searches.

A different approach to hybridization are multi-stage approaches. These decompose a problem into smaller subproblems, solving each with the most suitable method. For example, a large-scale scheduling problem might be divided into smaller time periods or groups of tasks, solved individually using CP for handling constraints precisely, and then integrated into a coherent overall solution using heuristics.

1.5 Conclusion

It is worth noting that the choice of approach depends on various factors such as problem complexity, problem structure, available computational resources, and the desired trade-off between solution quality and computation time. Each approach has its own set of advantages and limitations, and the selection of the most appropriate technique should be based on a careful analysis of the problem requirements and constraints and verified on the results of experiments on the particular data set.

Chapter 2

Operational Research Solvers

Solvers for mathematical programming are sophisticated tools designed to help decision-makers find optimal or near-optimal solutions to complex problems involving large amounts of data and numerous constraints. These solvers leverage advanced mathematical techniques like LP, CP, or MIP, which were introduced in the previous chapter.

The performance of these solvers is always improving, either by increasing raw computational power, implementing more efficient software, or using new approaches.

While solvers can easily make use of all single-threaded performance increases, not all can efficiently utilize the increases in the number of logical cores. For example, Cplex notes that "parallelism efficiency decreases for more than 4-8 threads. Increasing the thread count past this number will not significantly reduce the execution time." while memory requirements scale linearly as each thread requires its copy of the model data.

Implementation speed-ups are solver-specific and much less linear. Continuing the Cplex example - given the same set of MIP problems and hardware, this equates to an average of about 1.8 times faster per annum. However, with especially large improvements occurring in CPLEX versions 2.1-3.0 (inclusion of the dual simplex method), over 5 times the performance of the previous version, and in CPLEX versions 6.0-6.5 (many improvements gleaned from "mining" the academic literature), almost 10 times the performance. [Bix20]

2.1 Cplex

Cplex [Cpl22] is a powerful and widely-used commercial solver for solving integer, linear, convex and non-convex quadratic programming problems and convex quadratically constrained problems. Developed by IBM, Cplex offers a comprehensive suite of tools and algorithms designed to handle complex constraints and deliver high-quality solutions. This section will provide an overview of Cplex as a general CP solver, highlighting its key features and capabilities and comparing it to OR-Tools, an open-source CP solver.

Cplex supports many modelling languages, like Constraint Programming Modeling Language (CPLEX-M) and the Object-Oriented C++ API. It also provides DOcplex, a lightweight wrapper around the C API compatible with the NumPy and pandas, prominent Python libraries.

2.1.1 Key Features and Capabilities of Cplex

- Optimization Algorithms: Cplex incorporates a variety of optimization algorithms tailored for combinatorial problems. These include primal and dual variants of the simplex method, barrier interior point, or second-order cone programming.
- Parallel and Distributed Computing: Cplex leverages the power of parallel and distributed computing to speed up the optimization process. It can distribute the search across multiple cores and machines, allowing for faster solution space exploration and the solution of larger and more complex problems.
- Community: Cplex boasts a large community of users, researchers, and developers. Through forums and online resources, individuals can access solved examples and valuable insights, share best practices, and receive assistance from experts, thereby enriching the user experience and promoting continuous improvement and innovation within the optimization community.

2.2 OR-Tools

OR-Tools [PF23] is an open-source library developed by Google that also offers a suite of tools for solving combinatorial optimization problems. It offers general solvers for CP, LP, and MIP and specialized solvers for vehicle routing and graph algorithms.

While it is difficult to say that one solver is generally better, one clear advantage is that OR-Tools is licensed under the Apache License 2.0, a permissive license that allows commercial use.

2.3 Gurobi

Gurobi is a commercial solver known for its high performance in solving large-scale linear programming (LP), mixed-integer programming, and quadratic programming (QP) problems. It is widely regarded for its speed, reliability, and advanced features such as parallel processing, cutting-edge heuristics, and automatic model tuning. Gurobi is one of the fastest solvers available, often outperforming other solvers on large and complex MIP problems. It offers intuitive interfaces for Python, MATLAB, and more.

Chapter 3
Vehicle Routing Problem

The Vehicle Routing Problem is a classic combinatorial optimization problem that deals with the efficient distribution of goods or services using a fleet of vehicles. It first appeared in a paper by George Dantzig and John Ramser in 1959 [CW64]. It generalises the travelling salesman problem (TSP). It involves determining the optimal set of routes for a given set of vehicles to serve a set of customers or locations while minimizing the total cost or distance travelled.

In real-world scenarios, the VRP arises in various applications, including transportation and logistics, delivery services, waste collection, and public transportation. The objective of the VRP is to find an optimal or near-optimal solution that minimizes transportation costs, maximizes resource utilization, and satisfies various constraints such as vehicle capacity, time windows, and customer preferences.

The VRP is a highly complex problem due to its combinatorial nature and the interplay of multiple constraints. As a generalization of TSP, it also belongs to the class of NP-hard problems, meaning that finding the exact optimal solution for large-scale instances is computationally infeasible within a reasonable amount of time. As a result, researchers have developed various approximation algorithms [BHM05], heuristics, and metaheuristics to find near-optimal solutions within a reasonable time frame.

The VRP can be formulated in different variants, each introducing additional constraints and requirements. Some common variants include:

1. Capacitated VRP (CVRP): This variant adds a weight variable to each customer, assumes that each vehicle has a limited carrying capacity and aims to find a set of routes such that the total demand of customers served by each vehicle does not exceed its capacity.

2. VRP with Time Windows (VRPTW): Customers have specified time windows within which they must be served in this variant. The goal is to find a set of routes that respect the time constraints while minimizing the total travel time or distance.

3. VRP with Pickup and Delivery (VRPPD): This variant involves transporting goods or services from pickup to delivery locations. Each vehicle must pick up items from certain locations and deliver them to other locations while satisfying capacity constraints and other requirements.

4. Vehicle Routing Problem with LIFO: Similar to the VRPPD, an additional restriction is placed on the loading of the vehicles: at any delivery location, the item being delivered must be the item most recently picked up.

5. Open Vehicle Routing Problem (OVRP): The term "Open" is applied to problems where each vehicle is not required to return to the central depot after visiting the final customer, i.e. the vehicle routes are open paths instead of closed circuits.

The development of effective solution approaches for the VRP has been the subject of extensive research, leading to the formulation of various algorithms and techniques.

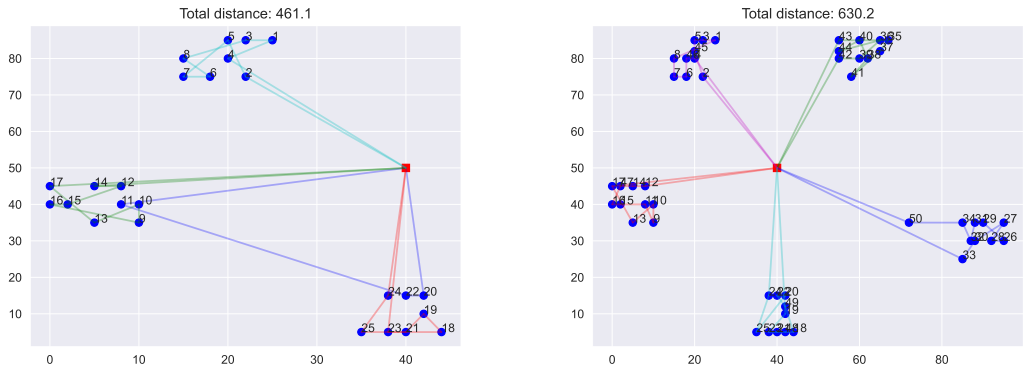
3.1 Solomon Benchmark

The Solomon benchmark [Sola] [Solb], developed in the 1980s, is a widely-used set of benchmark instances specifically designed for the Capacitated Vehicle Routing Problem with Time Windows (CVRPTW). This benchmark provides a standardized set of Euclidean problem instances that have been extensively utilized in Vehicle Routing Problem research.

The benchmark consists of a collection of realistic routing problems, each representing a different instance of the CVRPTW. These instances are based on real-world scenarios and capture various characteristics and complexities of routing problems businesses and organisations face.

Each instance in the Solomon benchmark is defined by a set of parameters, including the number of customers, the coordinates of the depot and customers, the demands of customers, and the time windows within which customers must be serviced. Since we are given coordinates instead of a distance matrix, these instances are Euclidian, and the triangle inequality holds. There are as many available vehicles as there are customers to ensure a trivial solution always exists. These parameters are carefully selected to create diverse problem instances that reflect different aspects of the CVRPTW, such as vehicle capacity constraints, time constraints, and geographical considerations. There are six sets of problems: The geographical data are randomly generated in problem set R, clustered in problem set C, and a mix of random and clustered structures in problem

Figure 3.1 Example of a solution to instances RC101.25 and RC205.50



set RC. Problem set '1' has a short scheduling horizon and allows only a few customers per route (approximately 5 to 10). In contrast, set '2' has a long scheduling horizon, permitting many customers (more than 30) to be serviced by the same vehicle. This results in 56 instances of each size, 25, 50 and 100, or 168 instances total.

The original formulation declares that the single objective is to minimize the total distance travelled, however [Solb] uses the same instaces except they use multiple objectives: first to minimize the number of vehicles and then the total distance.

The Solomon benchmark has been widely adopted in the research community, and as a result, it is often used in other, even more specialized variants of VRP by adding additional constraints. Similarly, there have been attempts to create larger instances to test new, more powerful approaches, most notably the Gehring & Homberger benchmark [GH99] with new instances of sizes ranging from 200 to 1000.

Benchmark instances assess algorithms' generalization capability. Derived from real-world scenarios, they indicate how well algorithms might perform in practical settings.

3.1.1 ILP CVRPTW formulation

Eqn. (3.1.1)

Objective:

$$\min \sum_{u=1}^{U} \sum_{(v_i,v_j)\in A} t_{ij} x_{ij}^{u} \tag{3.1}$$

Subject to:

$$\sum_{u=1}^{U} \sum_{(v_i,v_j)\in A} x_{ij}^{u} \geq 1 \quad (\forall v_i \in V \setminus \{v_0\}) \tag{3.2}$$

$$\sum_{(v_i,v_j)\in A} x_{ij}^{u} - \sum_{(v_j,v_i)\in A} x_{ji}^{u} = 0 \quad (\forall v_i \in V, 1 \leq u \leq U) \tag{3.3}$$

$$\sum_{(v_0,v_i)\in A} x_{0i}^{u} \leq 1 \quad (1 \leq u \leq U) \tag{3.4}$$

$$\sum_{(v_i,v_j)\in A} d_i x_{ij}^{u} \leq Q \quad (1 \leq u \leq U) \tag{3.5}$$

$$s_i^u + s_{ij} + t_{ij} - s_j^u + M x_{ij}^u \leq M \quad (\forall (v_i, v_j) \in A, v_i \neq v_0, 1 \leq u \leq U) \tag{3.6}$$

$$s_i^u + s_{ij} + t_{i0} - b_0 + M x_{i0}^u \leq M \quad (\forall (v_i, v_0) \in A, 1 \leq u \leq U) \tag{3.7}$$

$$a_i \leq s_i^u \leq b_i \quad (\forall v_i \in V, 1 \leq u \leq U) \tag{3.8}$$

$$x_{ij}^u \in \{0, 1\} \quad (\forall (v_i, v_j) \in A, 1 \leq u \leq U) \tag{3.9}$$

Where: $G = (V, A)$ is a directed graph, where $V = \{v_0, v_1, \ldots, v_n\}$ is the set of nodes and A is the set of arcs. Node v_0 is the depot, and the other nodes represent the

customers. A travel time t_{ij} is defined for each pair of nodes. Each customer v_i has a demand d_i, a time window $[a_i, b_i]$ within which service must start, and a service time st_i. A fleet of U vehicles with capacity Q is available. Variables x_{ij}^u indicate whether arc (v_i, v_j) is used by vehicle u, and s_i^u represents the service starting time at node v_i visited by vehicle u. b_0 denotes the end of the time horizon, and M is a large number used in the big-M method.

Constraints:

- Constraints (3.2) ensure every customer is visited.
- Constraints (3.3) and (3.4) define the sequence followed by vehicles.
- Constraints (3.5) concern vehicle capacity.
- Constraints (3.6)-(3.8) are related to the time windows.

3.2 Related Work

In the field of VRP, numerous papers have proposed innovative approaches. The paper by Clarke and Wright [CW64] in 1964 introduced a savings algorithm for the VRP, which provided an initial solution by merging feasible routes. Later research efforts have focused on improving route construction methods, optimizing various VRP variants such as the Capacitated VRP with Time Windows (CVRPTW) and the Vehicle Routing Problem with Pickup and Delivery (VRPPD), and incorporating additional constraints and objectives.

Booth and Beck's work [BB19] focuses on the Electric Vehicle Routing Problem with Time Windows, which extends traditional vehicle routing problems to incorporate the unique characteristics of electric vehicles. They propose the first CP approaches for modelling and solving the EVRPTW and compare them to a Mixed-Integer Linear Programming (MILP) model. The authors introduce two CP models, including a single resource transformation that significantly improves problem-solving capabilities. Experimental results demonstrate the superiority of the single resource CP model over the alternative resource model and the MILP model for medium-to-large problem classes. They also explore a hybrid MILP-CP approach that outperforms individual techniques for distance minimization problems with long scheduling horizons. The authors conclude by highlighting the importance of studying EV routing in the context of CP and identifying opportunities for applying their techniques to related areas like multi-robot task allocation.

Adam Zvada's work [Zva21] addresses the Vehicle Routing Problem with soft-constrained time windows (VRPTW) using machine learning and optimization heuristics. The thesis proposes a deep reinforcement learning model based on the Transformer architecture and Graph Attention Network for embedding the input instance. The model incorporates a reward function that considers the time window constraint. The thesis also explores other metaheuristic methods for solving VRPTW to benchmark the model's performance. The results show that while the deep learning model can solve VRPTW, it is outperformed by metaheuristic solvers. The conclusion emphasizes the lack of attention given to VRPTW and highlights the need for further research in this area. Future

work includes adopting a different model architecture to support soft-constrained time windows and pick and deliver scenarios.

In their work, [Hà+20] address a variant of the vehicle routing problem with two kinds of vehicles. They propose a Constraint Programming model and an Adaptive Large Neighborhood Search (ALNS) algorithm, incorporating linear programming models to check insertion feasibility. The experiments show that their CP model outperforms an existing CP-based ALNS on small instances, while the LP-based ALNS achieves superior solution quality across all instance classes. The authors also apply their method to a well-studied variant of the problem and improve upon four best-known solutions. Future research directions include developing efficient exact methods and applying the LP-based ALNS to other vehicle routing problem variants.

3.2.1 State of the Art

3.2.1.1 Exact Solvers

For recent state-of-the-art approaches using exact solvers, we can refer to [BA21], where the authors used meta-goal programming. In goal programming, goals are specified with target values, and the objective is to minimize the deviations from these targets. Meta-goal programming, an extension of traditional goal programming, is a type of multi-objective optimization technique used to find solutions that satisfy multiple goals by transforming them into a single-objective one by considering the sum of deviations (total deviation) and maximum deviation at the same time. A preference weight is defined by decision-maker(s) as well as a goal for each objective function. Another example is [Gov+21] using a bi-objective MILP model solved by fuzzy goal programming. Fuzzy goal programming incorporates fuzzy set theory to handle uncertainty and imprecision in goal formulation and constraints. This approach is particularly useful when goals and constraints are not sharply defined and are subject to vagueness or ambiguity.

3.2.1.2 Metaheuristics Solution Techniques

Here we would like to highlight [Gon+12] showcasing particle swarm optimization, [SSM21] with nondominated sorting genetic algorithm II, a multi-objective metaheuristic. [Rui+19] and [PPF18]which used biased random-key genetic algorithm, or [SBM23] with differential evolution.

3.2.1.3 Hybrid approaches

Among relevant approaches that combine two or more solvers belongs the combination of nondominated sorting genetic algorithm II with multiple objective particle swarm optimization, both viable for multi-objective optimization VRP variants, as showed in [GJ20] or [FRM20]. Nearest-neighbour procedure and petal algorithms are shown in [CLH21] and [AT15]. A petal algorithm [RBL96] is a heuristic approach used to solve VRP. The basic idea behind the petal algorithm is to generate a set of ”petals” or ”routes”, each originating from the depot and covering a subset of customers. These routes are then combined or optimized to create a feasible solution that covers all customers while respecting vehicle capacity constraints.

3.3 Conclusion

In conclusion, the VRP remains a vibrant area of research, continuously evolving with advances in optimization techniques and computational capabilities. The ongoing development of innovative algorithms and the formulation of new problem variants ensure that the VRP will continue to be a focal point for both theoretical exploration and practical application in optimizing logistics and distribution systems.

Chapter 4
Nurse Rostering Problem

The Nurse Rostering Problem, or Nurse Scheduling Problem, is a well-studied combinatorial optimization problem with significant practical importance in healthcare management. The objective is to create optimal work schedules for nurses, ensuring that various operational constraints and nurses' preferences are satisfied. Effective nurse rostering is crucial for maintaining high standards of patient care, optimizing hospital resources, and ensuring the well-being and job satisfaction of nursing staff. The complexity and critical nature of the problem have led to extensive research and the development of numerous datasets and benchmark instances for testing and comparing different solution approaches.

4.1 Datasets and Variability

The datasets used in nurse rostering research often come from real-world scenarios, and some have been compiled into benchmark repositories for academic and practical use [Uni] [Ces+15]. These datasets typically vary based on factors such as the nursing staff's size, the scheduling period's length, shift patterns, and specific institutional policies. Some datasets feature a couple of straightforward scheduling needs, while others have complex and stringent requirements with large horizons.

4.1.1 Problem Variants

- Single-Stage NRP: In academic contexts, solutions often address a single, fixed planning horizon where complete information is available. The single-stage NRP involves scheduling nurses for shifts within a set planning period (typically one week or longer) without considering past or future data. Examples of this variant can be found in the problem instances from the First International Nurse Rostering Competition (INRC-I) [Has+14].
- Multi-Stage NRP: In real-world scenarios, the quality of a roster within the current planning horizon is heavily influenced by past outcomes and future considerations, such as requested days off. Thus, optimizing each planning horizon independently may lead to suboptimal overall rosters. The problem instances from the Second

International Nurse Rostering Competition (INRC-II) [Ces+15] adopt a multi-stage approach. Here, solving a single stage of the problem corresponds to one week while considering historical data (e.g., each nurse's last worked shift and total night shifts). Moreover, rosters are evaluated across multiple planning horizons.

4.2 Common Constraints in Nurse Rostering

Nurse rostering involves balancing numerous constraints that can generally be categorized into hard constraints and soft constraints. Hard constraints are mandatory requirements that must be met for a roster to be feasible, while soft constraints are preferences or desirable conditions that should be satisfied as much as possible but can be violated if necessary, typically with an associated penalty.

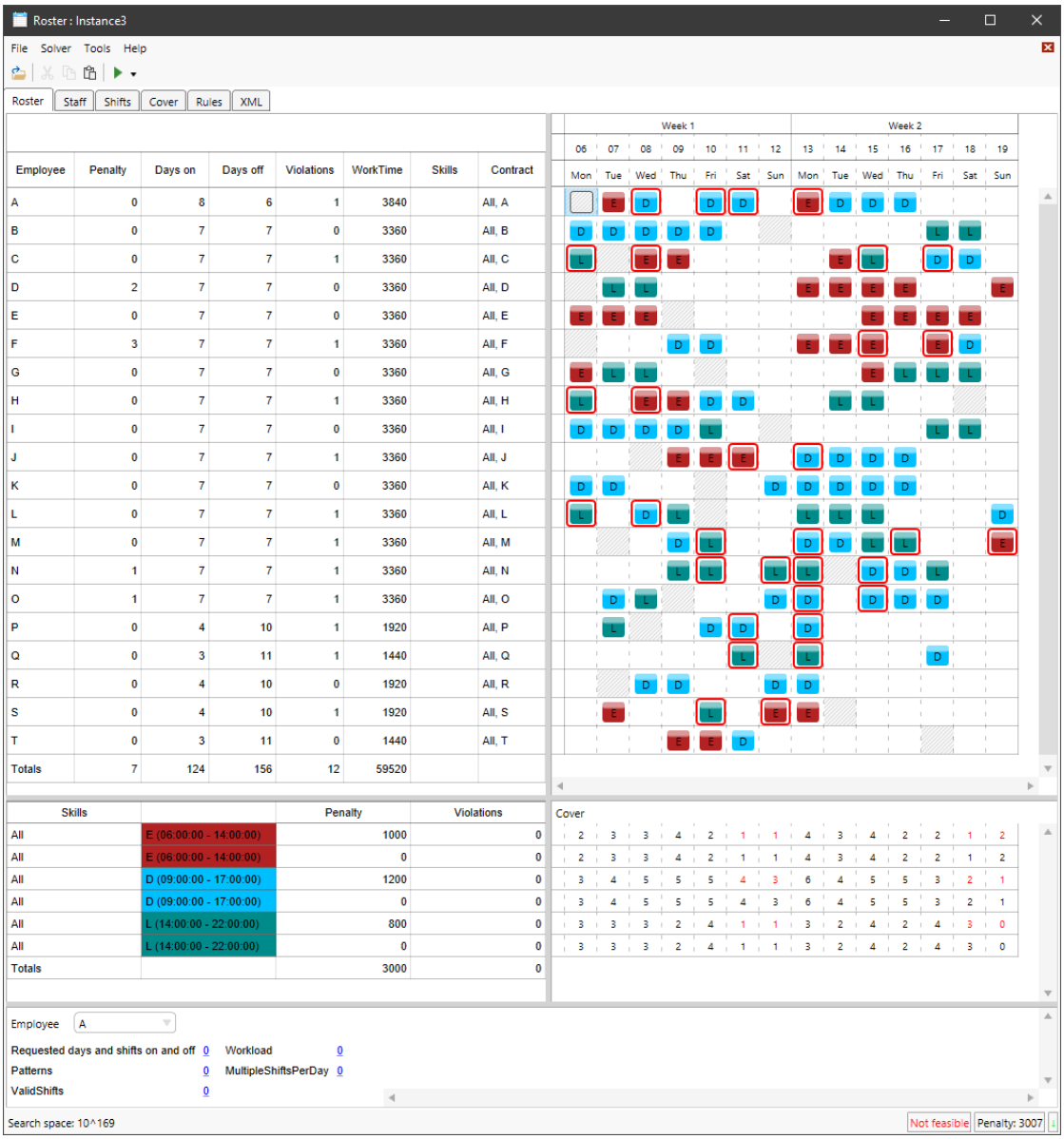
4.2.1 Hard Constraints

- Coverage Constraints: Ensure that each shift has sufficient nurses with the required skills to meet patient care needs. For example, certain shifts might require a minimum number of nurses or specific qualifications.
- Shift Type Constraints: Manage the assignment of different types of shifts, such as morning, afternoon, night, and day-off shifts, ensuring that nurses receive a variety of shift types as specified by the scheduling rules.
- Maximum Consecutive Workdays: Limit the number of consecutive days a nurse can work to prevent fatigue. For example, a nurse might not be allowed to work more than six consecutive days.
- Minimum Rest Period: Ensure adequate rest between shifts, such as guaranteeing at least 11 hours of rest between the end of one shift and the start of the next.
- Legal and Contractual Obligations: Adhere to labour laws and contractual agreements, such as maximum weekly working hours, mandatory breaks, and specific time-off requirements.

4.2.2 Soft Constraints

- Shift Preferences: Accommodate nurses' preferences for certain shifts or days off as much as possible. For example, some nurses might prefer not to work night shifts or may have requested specific days off.
- Fairness and Equity: Ensure a fair distribution of shifts among nurses, preventing scenarios where some nurses receive significantly more or fewer preferred shifts than others.
- Patterns and Continuity: Maintain preferred patterns or sequences of shifts, such as avoiding isolated single-day shifts or clustering similar shift types together.

Figure 4.1 Example of a solution to Instance3 from the Nurse Rostering Benchmark in the RosterViewer available at [Lim]



- Workload Balance: Distribute the workload evenly across the nursing staff, taking into account factors like total hours worked and the distribution of shift types.

For our work, we selected [Tim14] as it is one of the major benchmarks in this field and offers reasonably difficult instances. The instances are single-stage, and the constraints are similar to INRC-I. It is also well maintained with the last change in July 2022 and offers an ILP formulation. It also refers to a free application to verify and visualize the results.

4.3 Related Work

4.3.1 Exact Solvers

[ZZS18] applied ILP for a long-term nurse rostering problem in a hospital, addressing variability in nurse availability and schedule updates with CPLEX. [MM19] added soft constraints to connect weekly rosters, enhancing performance but still lacking optimal solutions for INRC-II. [RM16] reformulates problems as network-flow-based MILPs. Authors used this for INRC-II, with a Coin CBC solver implementing a look-ahead method that extends planning periods and relaxes constraints, achieving top results in the competition. [BC14] applied column generation, splitting the problem into a master problem and a pricing problem, performing well on INRC-I datasets. Lexicographic Goal Programming was used by [Böo+21] for multi-objective optimization in Danish hospitals. Lexicographic Goal Programming solves multi-objective problems by prioritizing goals sequentially, ensuring high-priority goals are met before addressing lower ones, and using Gurobi for optimization. Fuzzy Mathematical Programming was applied by [Jaf+16] for a hospital in Iran.

4.3.2 Metaheuristics Solution Techniques

[LH12] applied Adaptive Neighborhood Search to the INRC-I dataset, improving best-known results for 12 instances and [TSB15] used Variable Neighborhood Search for INRC-I, achieving best results for 50 instances. [MK19] implemented Iterative Local Search for INRC-I, outperforming other methods on smaller instances. [CGS20] used SA for the INRC-II dataset and improved many best-known results for the 4-week horizon instances. [Alt+12] applied Particle Swarm Optimization to a French hospital's problem, outperforming IP and CP methods.

4.3.3 Hybrid approaches

[RAL17] integrated IP and CP, using Gurobi for pre-solving and IBM ILOG CP for solving CSP models, achieving strong results on the INRC-I dataset. [Awa+17] combined Hybrid Harmony Search Algorithm with Hill-Climbing Optimization (HCO) and Particle Swarm Optimization (PSO) to address the INRC-I dataset, achieving improved results in a significant number of instances. [CZ20] combined decision trees with the Bat Algorithm, Particle Swarm Optimization, and a greedy search to address a real-world

diagnostic unit problem. This hybrid method generated higher-quality solutions in less computational time compared to individual methods.

4.4 Conclusion

The Nurse Rostering Problem is a complex and critical optimization challenge in healthcare management. With various available datasets reflecting different constraints and requirements, researchers can develop and test robust and adaptable algorithms to real-world scenarios. By addressing the NRP through advanced optimization techniques and leveraging the latest benchmark datasets, significant improvements in nurse scheduling can be achieved, ultimately benefiting both healthcare providers and patients.

Chapter 5
Implementation

As mentioned, this work was implemented in collaboration with Tomáš Omasta and is publicly available at [LO23]. While we collaborated on the design and implementation of many of the internal parts, the main focus of this work was on VRP, NRP, performance logging and statistics used to present the results in the next chapter.

5.1 Performance Logging

We implemented the performance logging for all three solvers used in our work, namely Docplex, OR-Tools and Pymoo; only the hybrid combination of multiple solvers does not currently support it. While, in theory, it is as simple as appending the results to the previous, doing just that proved challenging in the current implementation while not being essential for designing new approaches. This performance is represented by an entry under the key "solution_progress" in each of the logged solves in the run history of each instance. The entry contains a list of all new minimums of the criteria function, the time when they were found, and other optional solver-specific variables in a list as logged by the solver.

5.1.1 Benefits

We have not seen this kind of data widely utilized in the studied literature and believe it will unlock a new dimension through which approaches can be studied and compared. It allows a new look into how each implementation works without solving each instance for each length of time with the added benefit of precise timing. For example, a hybrid approach is designed to utilize the one strongest approach at each phase of the optimization process. Having the data of how each approach performs during the entire studied interval will simplify choosing how much time is allowed for each.

5.1.2 Disadvantages

As we mentioned, while this approach might benefit the research where the time allowed for the solver is treated as a parameter, it does come with a performance penalty, which should not be ignored in the final deployment. We have not studied this performance

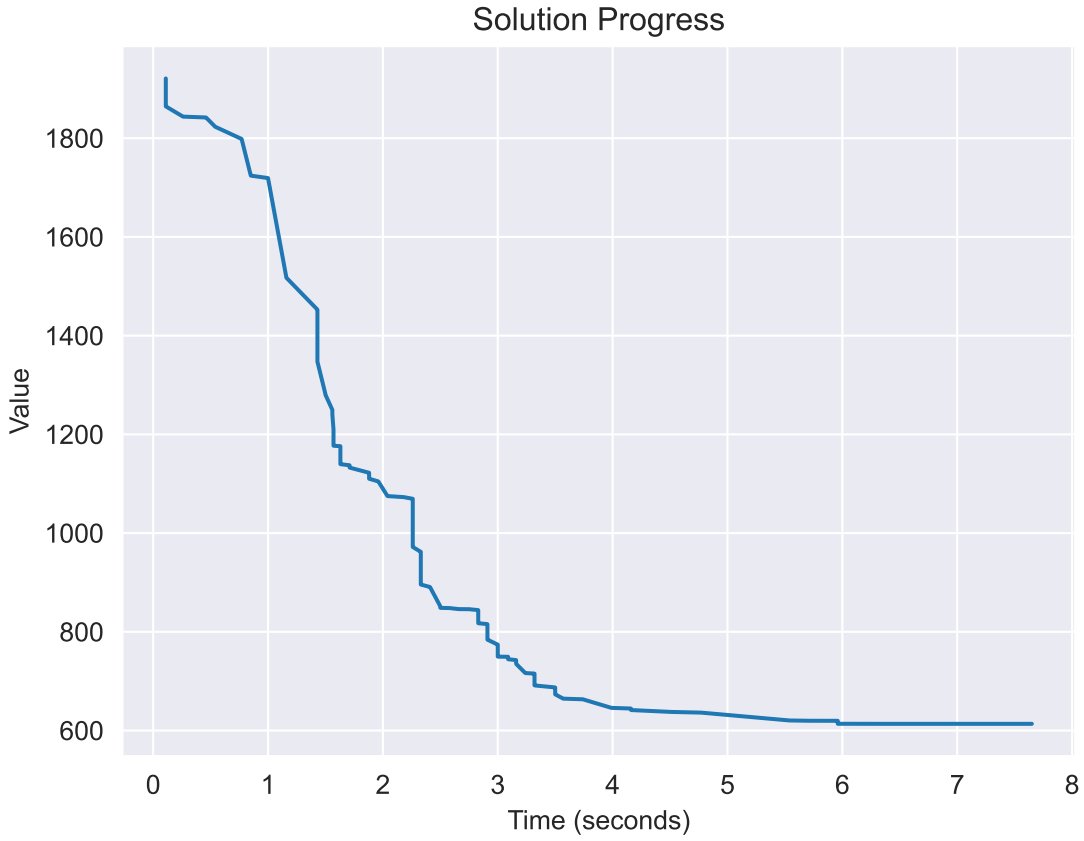


Figure 5.1 Example of the solution progress

penalty as we believe it not to be influential because the difference will be affected by more factors like the overhead of our framework and the usage of Python, which is widely known to be one of the languages with the worst performance, as the implementation programming language of our choice. These differences to a deployment where time is of the essence make this penalty non-significant. Due to the nature of implementation into our chosen solvers, we expect the least performance hit will be to Docplex, where we implemented the search history by choosing non-standard logging options, which allowed us to search the logs using a regex after the solve. In both OR-Tools and Pymoo, we had to modify a callback used after each evaluation, which caused a much greater impact on the performance.

5.1.3 Testing

During the testing, we experienced and subsequently fixed or improved various bugs. In the implementation for Pymoo, we modified the function used to log time as the standard implementation with time.time() is not recommended to be used, instead suggesting time.perf_counter(), also mentioned in [Kuž23]. The other issue we encountered was caused by an error in the batch evaluation of benchmarks, which caused the search

history of each solved instance to be appended to the previous, invalidating this data.

5.2 Statistics and Vizualization

To analyze and present the results of our experiments, we implemented a suite of examples to process, analyze and visualize the data. Note these are not usually in the form of ready-to-use general functions but rather a list of specialized code snippets and their evaluation results that might need to be modified to work on a different application. For processing the results, we transformed the relevant fields of all instances to a Pandas Dataframe - these include information about the instance and the relevant information for each solve in a list.

We provide a utility function for VRP plots that modifies the X-axis to show the type of instance for the Solomon benchmark. We utilize this function when comparing the distance or time required by different solvers, for example, in figure 6.2.

5.3 Vehicle Routing

Our main focus was on VRP; thus, we implemented and compared multiple solving approaches to the instances in the Solomon benchmark as well as a loader for these instances and a visualiser and validator for the results into the General Optimization Solver repository [LO23].

We represent the solved solutions in a unified fashion across all solvers in order to have directly comparable results. In the case of VRP, by a list of lists, with one list for each vehicle used, the list contains the path taken. 0 represents the depo, and each vehicle has to start and end at the depo, while customers are represented by their respective number [*1, 100*].

5.3.1 Cplex

5.3.1.1 Mixed Integer Programming

The MIP implementation was inspired by examples of implementations for Cplex using the available APIs for Python and others from Alex Fleisher, an IBM employee, available at [Fle].

We are going to refer to a variable n, which equals the number of customers.

This implementation uses a list of integer variables as the main decision variable that stores the paths. This list is non-inferred, and every other variable used in the model is marked as inferred. This means that if all variables in this list have their values set, all other variables will already have their values determined. We define one variable for each customer and two variables for each vehicle to serve as a starting location and ending location, respectively, or since the number of vehicles is defined to be the same as the number of customers, this also equals *3n*. This list defines how each vehicle travels backwards, and this reversal is necessary for the other constraints, such that each index in the list denotes the number of the represented nodes, and its value is the number of nodes visited by the same vehicle beforehand. This limits the values to [*0, 3n-1*]. A

node can not link back to itself. The first visited node links back to the ending node, which finishes the circuit.

We are going to need an extra list of variables that represent which vehicle serves which node called *veh*. The nodes which serve as starting/ending nodes are hard set to the value of the respective vehicle, while customer nodes can have any values in $[0, n]$. However, each customer in a circuit needs to have the same value.

The capacity constraint is resolved using function *pack*, which is "used to represent sub-problems where the requirement is to assign objects to containers such that the capacities or minimum fill levels of the containers are respected" [Cor]. For this function, we define 3 other lists of variables: *load* to represent how capacity is used in each vehicle with the maximum value set to the maximum capacity, *where*, which contains the first n values of *veh*, just the customers. *Demand* with demands for each customer and *used*, a single integer variable that equals the number of vehicles used.

The time constraints are represented by a list of integer variables with a length of $3n$. Each of the customer nodes is limited in value to its time window. Starting nodes are hard to set to zero. The value of each node is computed with respect to the start time of the previous node, its service time, distance to the next node and the next node's earliest start.

Finally, a list of the travelled distance of the vehicles is computed, and the sum of this list is minimized.

This is the only implementation that currently works with warm start, meaning a solution is given to the solver during the inicialization that guides the solver, it does not restrict the search space to the values of variables it sets. This is done by reconstructing the model variables based on the given solution. Since we only have one list of inferred variables, we only need to reconstruct *prev*, and Cplex is able to complete the solution.

5.3.1.2 Constraint Programming

We modified the formulation from [BB19] for the CP implementation. These modifications were made to adapt their solution from the vehicle routing problem with synchronization constraints and two types of vehicles for our CVRPTW. We will again use n for the number of customers.

Variables:

- *Visit_veh*: Here we represent the solution using two dimensional array *visit_veh* of interval variables for the visit to a customer i by vehicle v. This variable is optional for customer nodes, meaning it might or might not be used in the solution.
- *Visit*: An interval variable representing the visit to a customer. Each node has an associated visit interval.
- *Route*: A sequence variable representing the order of visits for each vehicle.
- *Total_distance*: The objective variable representing the total distance travelled by all vehicles.
- *Load*: The load carried by each vehicle, which should not exceed the vehicle's capacity.

Constraints:

- *Alternative*: The *alternative* function is used to specify that each node must be visited by exactly one vehicle
- *No Overlap*: Ensures that visits do not overlap in time for each vehicle
- *Start of Route*: The first visit of each vehicle at the starting node starts at time 0
- *First and Last Visit*: Specifies the first and last visit of each route to be the starting and ending node of each vehicle
- *Capacity*: The load for each vehicle is calculated as the sum of the demands of all customers visited by that vehicle. The load must not exceed the vehicle's capacity

5.3.2 OR-Tools

We adapted the code from [Kre] for this implementation. The model did not have to be modified for our application; we only had to adapt the pre-processing data. This model utilizes the advantages of OR-Tools specialized VRP model with functions like *AddDimensionWithVehicleCapacity* for vehicle capacity, *RegisterUnaryTransitCallback* for customer demands or *RegisterTransitCallback* for distance between nodes.

5.3.3 Pymoo

Since this is not an exact algorithm, it follows a different design principle, unlike the previous implementations. For metaheuristic approaches, we need to implement a way to transform the solution to a chromosome and back. A chromosome is represented by a list of floats. We chose the representations present, for example, in [MJP21]. This representation works by taking the list of paths without the depo indexes and flattening them into a single list. Transforming the list of floats back into paths starts by calculating the order from lowest to largest. We can understand this as an order of visits. An important advantage of this representation is that every chromosome can be mapped to at least one valid path.

5.3.3.1 Local search

We need to implement a local search algorithm to transform the order of visits into paths, as one order of visits may represent many valid paths. For this reason, we implement numerous algorithms. However, different chromosome representations might have a straightforward way of converting a chromosome into paths.

- We start with a simple algorithm that starts at the depo and appends customers in the visit order until it fails a time window or capacity constraint. Only then does it return to the depo and continue with a new vehicle.
- The second algorithm is designed to search the entire search space of valid solutions. It is guaranteed to find the best solution for a given visit order, but the time complexity has an upper bound of 2^n where n is the number of customers since it has up to two options at every customer - use a new vehicle or use the old vehicle.

- A variant of the previous algorithm uses a heuristic based on the distances of the current node, next node and depo to prune many of the branches. This is supposed to reduce the expected time complexity extremely while keeping good solutions.

- Other algorithms utilize heuristics for the decision on when to use a new vehicle even when they are not yet forced to by the constraints. These heuristics are based on comparing the distances of the current node, the next node and the depo where a new vehicle can start. Some of the heuristics also took into consideration how much capacity of the current vehicle was used.

- We also leave a couple of our ideas for smarter local search algorithms in the code ready for future improvements. For example, one idea is to compute the initial paths using the first algorithm we mentioned but then try swapping vehicles for some of the customers so that the resulting paths are still valid. A second idea is based on the iterative deepening search. The algorithm would search depth-first until it needs to use the new vehicle and then expand the search to width until these alternative paths reach the customer for which the new vehicle was necessary. It would then choose the best solution and repeat.

The advantage of this approach is that with implemented functions to transform paths to and from a chromosome and a function to sum the cost of paths, we can plug and play any of the metaheuristics offered by Pymoo. These functions can also be utilised for any other metaheuristic not included in Pymoo except with more effort. Since our repository is built around general solvers it is recommended to copy our functions from our code into a smaller project. The effort of implementing a new solver into our project is expected to be much greater than that of implementing a metaheuristic. This is because many metaheuristics are, in their essence, very simple algorithms that only result in short functions of a couple of lines of code. The results can then be copied back and compared.

5.3.3.2 Hyperparameters

Many metaheuristics feature a list of hyperparameters that can fit a particular use case better. Hyperparameter optimization is an extremely computationally expensive topic, usually utilizing grid search, which is simply an exhaustive search through a manually specified subset of the hyperparameter space by evaluating each combination of hyperparameters. For instance, Differential Evolution has 6 binary or continuous hyperparameters, which would result in a 6D search space for every instance.

Fortunately, we can use more efficient methods as a lot of research has been conducted on this topic because of its application in training neural networks.

We adopt a new feature of Pymoo, which offers code for hyperparameter optimization on [Pym]. The last example from this page allows us to find the best set of hyperparameters for one instance while reducing randomness by trying multiple seeds for the random function. It runs the method for each random seed until the set termination criteria are met. The performance metric used is the average number of evaluations.

We utilize this example to implement our expanded hyperparameter selection by training metaheuristics on a learning set of instances and then evaluating a different set

of testing instances. The best hyperparameters are then saved to be used for further evaluation.

5.3.4 State of the Art

We were not able to find a reliable source for current state-of-the-art results. Many repositories were abandoned, for example the original repository for Solomon benchmark [Sola] was last updated on March 24, 2005. For this reason, we developed a workflow to find and potentially update the known best results, which we refer to in our repository. First, we need to paste the values into *data/VRPTW/best.csv* in the same format as the values already present. Then, the code in *stats.ipynb* will process and analyse the data and finally update the known best values for our instances. For the values currently present in the repository, we searched through multiple repositories and publications in order to get a representative snapshot of the current state of the art. Since it is unfortunately not common practice to publish the paths for these values, we opted not to verify found solutions. Because our solution to this problem can not recognize whether the best solution comes originally from the resource we linked, we do not mention the author of the solution.

We recognize that our solution to this problem is not ideal and is more of a bandaid. The field would benefit from a unified, recognized, and updated repository of solutions for these still popular instances, ideally done by the resurrection of the original Solomon benchmark repository, which still holds great significance.

5.4 Nurse Rostering

Our NRP solver is an implementation of the IP formalization given in [Tim14]. As the decision variable, it uses a three-dimensional array of binary variables representing all the nurses, days and shifts implemented as a *binary_var_dict*. The most challenging aspects of this implementation are constraints on minimum consecutive shifts and minimum consecutive days off, which have to be implemented by preventing every solution that would not follow these constraints. As a result, the number of total constraints and the size of the model scale very quickly with the amount and values of these constraints, and the performance decreases.

Our state-of-the-art solutions for this problem originate from the source of the instances [Tim14].

5.5 Conclusion

Our work focuses on two very different problems. On the one hand, we have VRP with an incredibly large search space and relatively few constraints limiting it. This leads to the behaviour of the solvers to run until they reach the timeout. On the other hand we have NRP with also a very large search space but rather limiting hard constraints wher the solvers. Unfortunately, we were not able to dedicate enough time to NRP to explore it as we did with VRP.

The general parts of the implementation, like the performance logging, can be found in the folder *src*. Problem-specific implementation is in *src/vrp* and *src/nrp* folders. For example, the statistics and visualization are in the file *stats.ipynb* and a selection of other useful tools like the hyperparameter optimization are in *tools.ipynb* and solvers are in the *solvers* subfolder. In the subfolder *starters* are files used to evaluate the implementations.

Chapter 6
Experiments

The experimental chapter of this thesis is designed to verify, compare and study our implemented approaches and show the powerful tools designed for these reasons. To validate our results, we implemented a solution checker for both of our problems that validates whether all hard constraints are followed and compares the value of the solution with the real value. We used this function on all of our reported results when evaluating and are confident in the validity of our solutions.

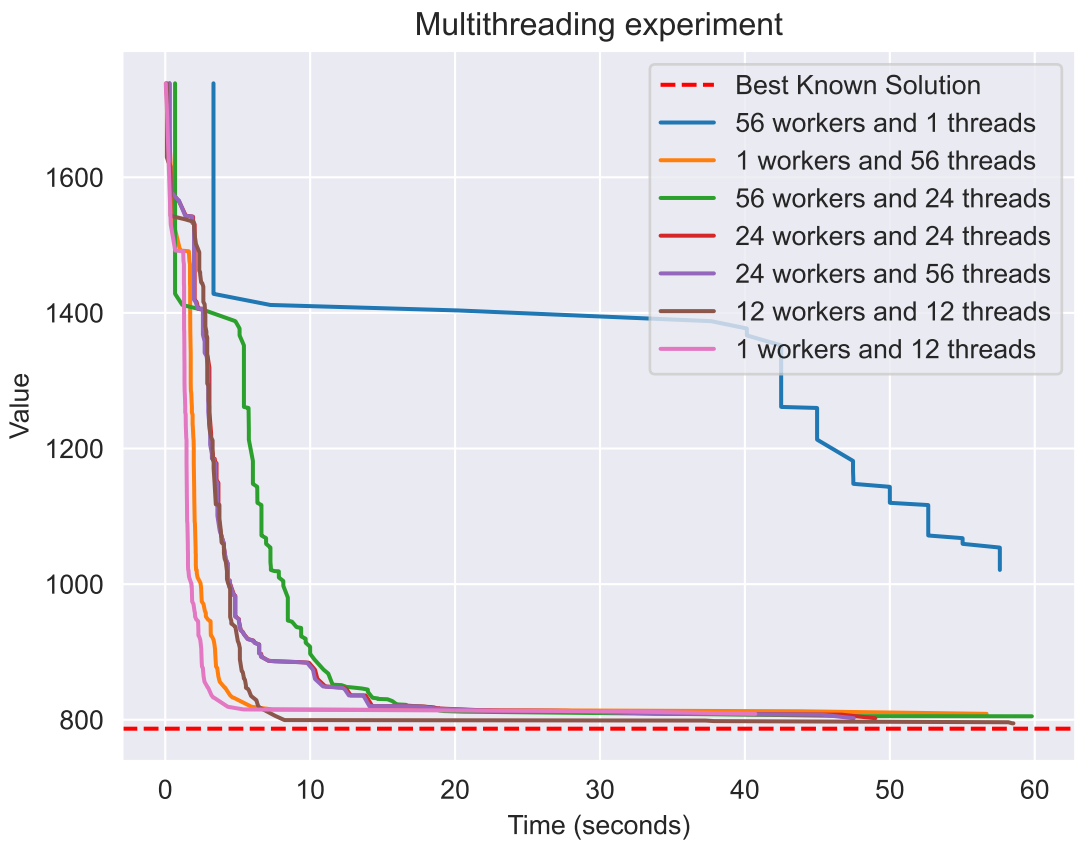
We decided to compare our results on 15 minutes of evaluation. This amount of time for each solve of each instance works out to almost 2 days of computational time for the entire Solomon benchmark. Longer solving times would be impractical for our implementation, and thanks to our performance logging, we also get the values for all shorter solves with perfect granularity.

In this section, we start by selecting the parameters for our evaluation, such as the number of threads and workers. Then, we find the best two implementations of exact solvers and metaheuristics to combine in the final hybrid approach. In doing so, we showcase the features we developed to help users faster develop and compare their future approaches to these problems.

The experiments were run on a CentOS Linux release 7.5 cluster node, which uses Slurm as a workload manager and job scheduler. The cluster node has two Intel Xeon E5-2690 v4 CPUs, 14 Cores/CPU, 2.6GHz with 35 MB SmartCache. The RAM was limited to 32 GB. Later, due to issues with the cluster, we had to use a local computer with AMD Ryzen 5 5600X. This processor has 6 physical cores and 12 threads and operates at 3.7 GHz with a Boost frequency of up to 4.6 GHz with 32 MB of L3 cache paired with 32 GB of DDR4 memory at a speed of 3600 MHz. The software used was CPLEX/22.11, GCCcore-10.2.0, and Python-3.8.6. The experiments were run on a single CPU node.

6.1 Multithreading experiment

This experiment was done in order to obtain the ideal settings for all the other tests. We use the performance logging on multiple solves of the same instance to get these values. We used our MIP model implementation and Cplex. For consistency, we fixed the random seed.



■ **Figure 6.1** Example of the solution progress for multiple solves on the instance R109.50. All runs were performed on cluster nodes except for those with 12 threads, which were run on the local computer

As shown in figure 6.1, the solver performs similarly well unless we significantly mismatch the number of workers and threads. This is an important observation since the Cplex's default setting is to use as many workers as there are threads available on the machine. This results in a workers-to-threads mismatch on the cluster and subpar performance since using an entire node's worth of resources, all 56 threads, is not optimal. For this reason, we added a parameter passed to Cplex, allowing us to control the number of workers used.

Another observation is that fewer workers tend to converge faster at the beginning, but it takes longer to find the global optimum.

We can also note that using fewer, much faster cores is better than using many slow cores. However, computing all solutions on the cluster was much more feasible for us. This confirms the claim that Cplex performance does not scale well above 8 threads 2.

In Cplex, memory requirements scale linearly, as mentioned previously, but our VRP implementation uses very little memory. This is confirmed by the fact that using many workers does not significantly negatively affect performance.

As a result of this experiment, we chose to continue with the value of 28 threads and 28 workers. We believe that this combination offers the best conditions for Cplex. In our use case, we are not heavily limited by our resources, and this number of threads only populate half a compute node. We also see no benefit in using more threads. An important consideration was also that these systems use two CPUs, and using only one of them avoids the possibility of instability caused by the communication between CPUs.

6.2 Experiment with exact solvers

Name of the Solver	Count of Best Solutions	Count of Bad Solutions
Docplex, CP model	40	21
OR-Tools	60	10
Docplex, MIP model	126	0
Name of the Solver	**Total Sum of Distances**	**Average Time (s)**
Docplex, CP model	117788.3	403
OR-Tools	111532.0	227
Docplex, MIP model	106721.2	222

Table 6.1 For this table, we define the best solution as a solution within 0.5% of the best-found solution and a bad solution as one with a length of paths over 25% of the best-found solution. The total sum of distances equals the sum of all paths over all instances. The average time is the average of when the solvers inserted their last solution into the search progress

In figure 6.2, we can see that our MIP model managed to find almost all known best paths for instances of size 25 and 50 and the best results of all our solvers on the instances for 100 customers. The CP model implementation outperforms the OR-Tools implementation on instances of size 25 but loses on the larger instances. Table 6.1 confirms that the MIP model performs the best with a significant lead, and OR-Tools is overall second best in these metrics.

Meanwhile, the MIP implementation with Cplex crushes both other implementations in terms of value, speed, and consistency. However, it is important to note that all

implementations reach overall good results, with even the CP model finishing with only 21 out of the 168 instances over 25% the known best values and with only a single result in the entire experiment going over 50% - the result of OR-Tools model reached 62% on the instance RC202.25.

6.3 Experiments on metaheuristic methods

Unlike exact solvers, metaheuristics benefit from tuning many parameters. We will focus on finding the best general local search implementation, choosing a good representative selection of metaheuristics, choosing the best set of hyperparameters for each of the metaheuristics, and finally, choosing the best metaheuristics for our goal. We choose to use the same number of threads and time as in the case of exact approaches to achieve the best comparison between the approaches.

6.3.1 Best local search

We conduct two sets of experiments to find the benefits of each of our algorithms mentioned in section 5.3.3.1. The first experiment is aimed at the general performance of each implementation, while the second one is specifically aimed at their usage for the metaheuristic algorithm. The application of each of the experiments is slightly different as in the optimization part of the metaheuristic, we value time much more than at the end, where we convert the final chromosome into a solution. This is because the optimization part is limited by the seed of the decoder, whereas the final decoding is run only once, and its performance will affect the resulting value more.

6.3.1.1 First Experiment

The first experiment aims to test how the local search decodes a given chromosome. For examples of chromosomes, we used the best paths for each instance found by the exact approaches, without deposit and flattening into a list. We look for the final length of the paths reconstructed and the decoder's length.

Name of the decoder	Count of fails
decode_chromosome_fast	59
decode_chromosome_rec	0
decode_chromosome_rec_pruned	8
decode_chromosome_second	11

Table 6.2 Table of decoders and their overall fails. We mark a result as a failure if it is over 5% longer than the original paths

The table 6.2 shows that the full recursive function has a perfect score as expected. The pruned version is only slightly worse, and the version with a heuristic is slightly worse yet. For a full picture, we have to look at table 6.3. Our perfect implementation takes an incredible amount of time to complete even the smallest instances, with a couple of minutes on average for each. This is that time complexity of 2^n we were talking about. We do not have the results for the larger instances as they would still be running even if

N	Decoder	Time (s)
25	decode_chromosome_fast	0.009
25	decode_chromosome_rec	11289.048
25	decode_chromosome_rec_pruned	0.039
25	decode_chromosome_second	0.011
50	decode_chromosome_fast	0.017
50	decode_chromosome_rec	Did not finish
50	decode_chromosome_rec_pruned	0.089
50	decode_chromosome_second	0.020
100	decode_chromosome_fast	0.036
100	decode_chromosome_rec	Did not finish
100	decode_chromosome_rec_pruned	0.215
100	decode_chromosome_second	0.045

Table 6.3 Table of decoders and the sum of time they needed to finish each size of instances

we did not cancel them (if we take that one instance of size 25, which took 150 seconds, the time needed, for instance, with 50 customers, scales to about 160 years). This result means that this decoder is no longer of use to us. Next, we see that both decoders with linear time complexity take roughly the same time on each side of the instance, as expected. Both of these and the pruned recursive decoder roughly double the time needed as the number of instances doubles. This verifies that our pruned algorithm cuts down the expected time complexity from 2^n to n.

Combined results in these tables mean that the best decoder should be either the recursive pruned one or the heuristic one, depending on whether we prefer slightly better results or faster results.

6.3.1.2 Second Experiment

The second experiment is evaluated by using the decoders for Particle Swarm Optimization. We chose PSO because it does not have tunable hyperparameters, thus making it slightly more general for this application. The assumption for all the other metaheuristics is that the difference between performance with different decoders will be the same or similar.

Important results were seen in 6.3 is that the recursive pruned decoder is not evaluated for the larger instances. This is because, in at least one instance, it encountered a chromosome on which the pruning does not work as well. This means that low expected time complexity can not be solely relied on when using metaheuristics as the huge amount of evaluations also results in needing its upper limit to be conservative.

We instead include a different decoder called *min of 2*. This simple modification evaluates both variants *fast* and *second* and returns the better result. Results for the first experiment can be extrapolated from the original results - the values and count of fails will be the minimum of these 2 or lower, while the time will be negligibly larger than the sum of both.

Disregarding the result of the recursive decoder in table 6.4, we see a surprising result: the worst-performing function from the previous experiment finished as the best.

Name of the Solver	Total Sum of Distances
decode_chromosome_rec_pruned	27637.9
decode_chromosome_fast	179380.1
decode_chromosome_second	179992.2
min of 2	185388.9

Table 6.4 Table of decoders and the sum distances of all their solutions. The value for decode_chromosome_rec_pruned is misleading as it only finished on the smallest set of instances

A large difference in decoding between chromosomes created from great solutions and what are essentially random solutions would explain this unexpected result.

Another takeaway from these results is that even with similar time complexity, the decoder *min of 2* always returns the same or a better solution than either one of the decoders it uses, finished with worse overall results. This can be explained by the difference in speed. Even with the same time complexity, the actual time can be different, and in the case of this decoder, it absolutely will be. According to this result, we estimate that the speed of the decoder plays a very significant role in the performance of metaheuristics. This is not a great result for our repository overall. It means that a metaheuristic with a decoder implemented in Python will perform much worse than the same metaheuristic with the same type of decoder implemented in a faster programming language like C++.

6.3.2 Best metaheuristic

For the selection of tested metaheuristics, we choose Particle Swarm Optimization, as mentioned earlier, as well as Differential Evolution and Biased Random Key Genetic Algorithm. This selection was based on the review of literature done in 3.2.1.2 with a focus on metaheuristics implemented in Pymoo.

To find the best metaheuristic, we first need to apply the hyperparameter optimization mentioned in 5.3.3.2. Using the best set of hyperparameters for each metaheuristic, we evaluate them using the benchmark.

We use figure 6.4 to analyse these experiments' results. From this figure, we can deduce that, on average, the MIP model beats all metaheuristics at every point in time, starting at the point at which it finds its first valid solution. PSO and DE are able to construct the first solution in just a hundredth of a second, while BRKGA operates in the tenths of a second but converges much faster, beating them at around the one-second mark.

We did not expect the crushing result of Cplex. Furthermore, we observed that metaheuristics performed slightly better on the smaller instances, some even beating Cplex until the one-second mark, but an even bigger lead in performance for Cplex on the larger instances. This goes against our expectations, as metaheuristics usually perform better at the beginning of the optimization process and scale better for larger problems. We expect that this is caused by our metaheuristics implementations underperforming. When compared to similar approaches in literature [Gon+12], we found that our solutions have roughly twice the value of the reference at the same time. Our implementation decisions certainly play a role in this result. A faster implementation of

the chromosome decoders, ideally in C++, would play a significant role. The next step would be to measure the impact of our logging implementation on this faster decoder; as we noted in the decoder experiment, the speed of the decoder plays a crucial role. We believe that these changes would get us closer to the expected performance. The scaling issue needs to be further studied, we recommend integration of larger datasets like [GH99]. The Cplex results could be moved slightly further to the left on the time axis if we included the time necessary to build the model.

6.4 Hybridization experiment

For the hybridization we selected two parts, first use BRKGA and follow it by giving its result to the MIP model. The selection of metaheuristics is influenced by the smallest time period, which Pymoo accepts as a parameter, and that is one second. If we look at the figure at 6.4, we see that the one-second mark values of our metaheuristics are rather ambiguous, so we choose BRKGA for its superior convergence performance. Because of the results of the previous section the experiment is rather theoretical as we do not expect an improvement, we only want to verify the implementation. Essentially, we are giving Cplex a solution to start with, but with the knowledge that if we had used Cplex from the start, it would have had a better solution by that time.

Name of the Solver	**Count of Best Solutions**	**Count of Bad Solutions**
GA BRKGA	17	6
Docplex, MIP model	126	0
Hybrid approach	124	0
Name of the Solver	**Total Sum of Distances**	**Average Time (s)**
GA BRKGA	127392.5	403
Docplex, MIP model	106721.2	222
Docplex, MIP model	107934.8	900

Table 6.5 For this table, we define the best solution as a solution within 0.5% of the best-found solution and a bad solution as one with a length of paths over 25% of the best-found solution. The total sum of distances equals the sum of all paths over all instances. The average time is the average of when the solvers inserted their last solution into the search progress. Since the hybrid approach does not support performance logging, we used the sum of times given to both solvers.

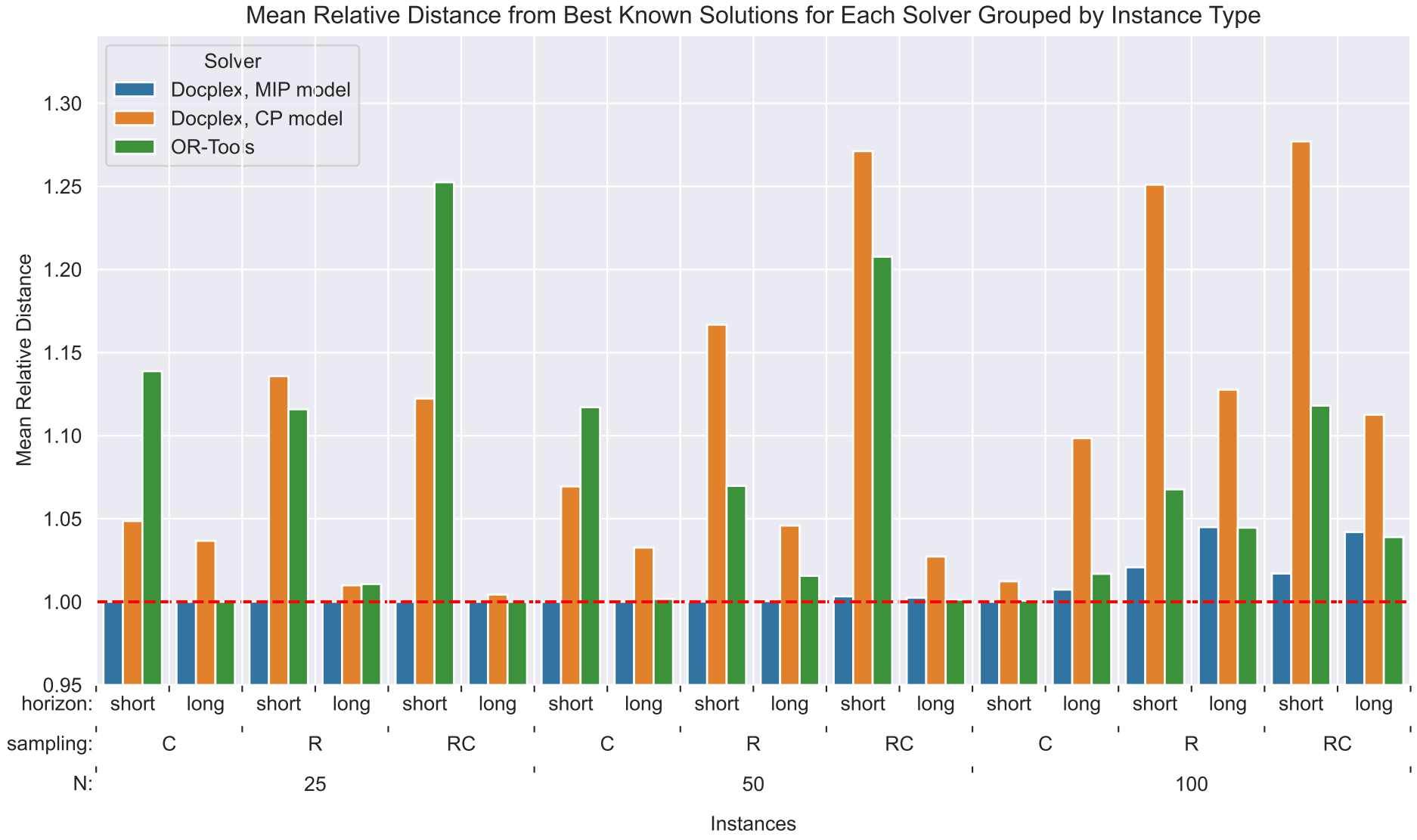
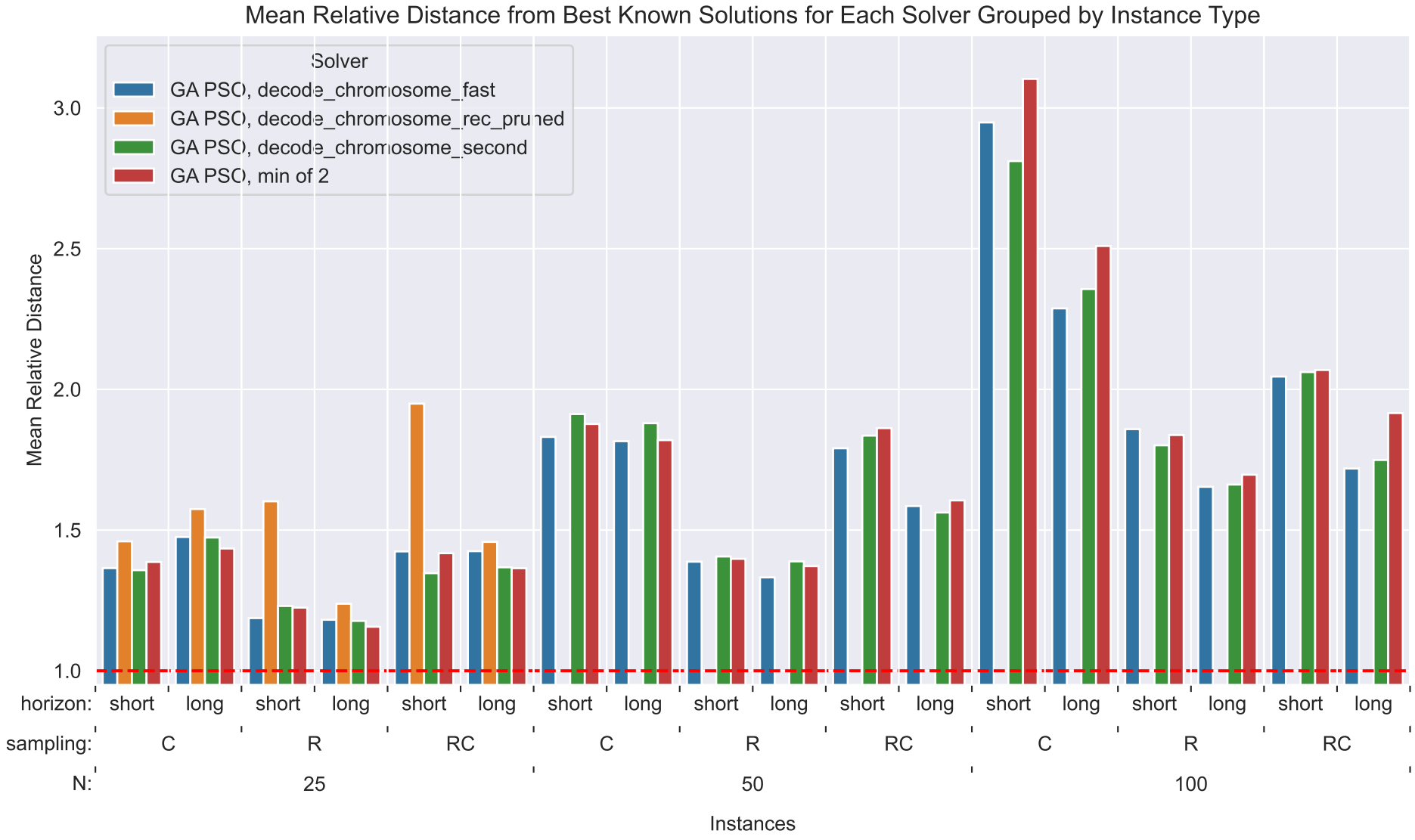


Figure 6.2 Results of all three of our exact solver implementations grouped by the kind of instance. Values are computed relative to the best state-of-the-art approach we have found.



■ **Figure 6.3** Results of our decoders when solving the instances similar to 6.2

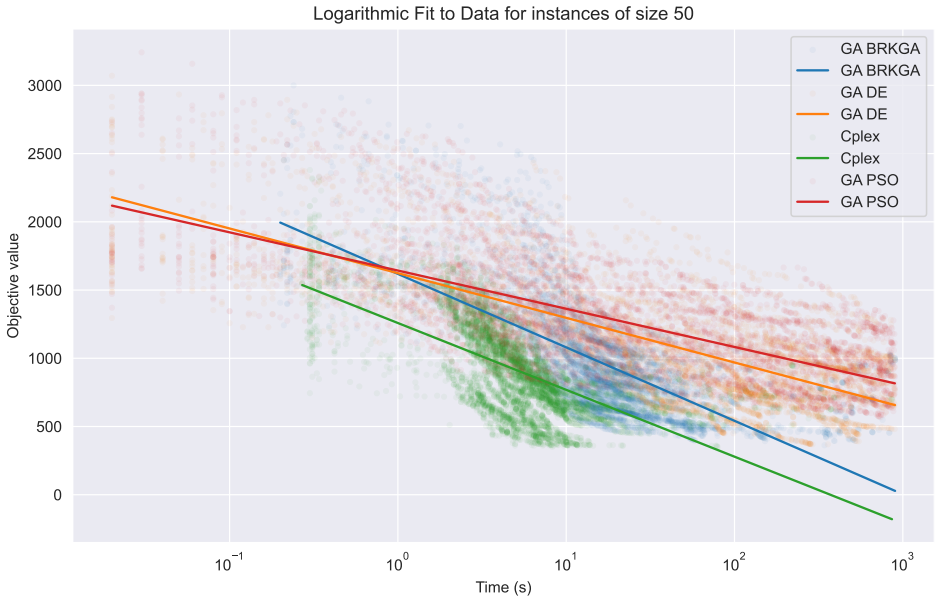


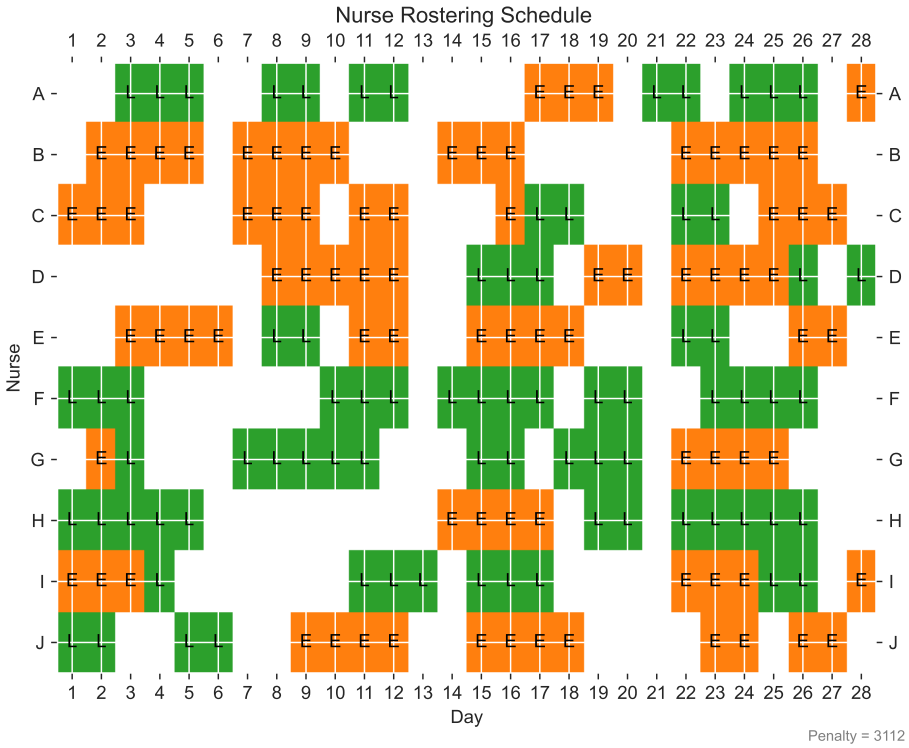
Figure 6.4 Results of the selected metaheuristics and the MIP model on instances for 50 customers. The points represent the values of all found solutions and their time. We estimate the search progress over time using a logarithmic function to simplify the analysis. The x axis is set to logarithmic scaling to improve visibility in the most clustered area.

Chapter 7

Nurse Rostering Experiment

This experiment serves to verify the validity of our implementation. In table 7.1, we can see that our implementation returned valid solutions for most of the instances. For larger and more constrained instances, the solver ran out of memory. The main cause of this issue is mentioned in 5.4 as the constraints on minimum consecutive shifts and minimum consecutive days off that scale are extremely large with those larger instances.

Figure 7.1 Example of a solution to Instance4 using our own function to visualise the solution. This instance has a time span of four weeks, 10 different employees and two kinds of shifts



Instance	Refence solution	Found value	Time
1	607	1204	5.468
2	828	2121	900.015
3	1001	3007	900.057
4	1716	3112	900.021
5	1143	4529	900.028
6	1950	5227	900.037
7	1056	4634	900.029
8	1300	6711	900.057
9	439	2657	900.061
10	4631	12745	900.074
11	3443	None	None
12	4040	None	None
14	1278	9945	900.094
15	3829	None	None
16	3225	None	None
17	5746	None	None
18	4459	None	None

Table 7.1 Table with results from NRP implementation verification run. These solutions were computed on the local computer instead of the cluster with a timeout set to 15 minutes. During the computation, the system ran out of available memory.

Chapter 8
Conclusion

In collaboration with Tomáš Omasta, this thesis implements new approaches into the [LO23]. This allows for similar new approaches to better and faster analyze their benefits. The ultimate objective of our work is to implement a hybrid solver. We explore many details that are required for this goal.

We split the text into three parts. In the first, we briefly introduced all the theory necessary for understanding our work and explored similar approaches from the literature. In the second part, we explained how our own implementation of each part of the solver works. In the final part, we presented the results from our experiments with visualizations that provide a detailed look into how each part performs.

We have implemented and verified processes necessary for implementing hybrid approaches. We compared several approaches to exact solvers. We have shown the complexity of choosing the best parameters for heuristic approaches by creating several chromosome decoders, selecting multiple metaheuristics and comparing them. For this comparison, we also developed important tools, which we have shown. Finally, we used everything together to implement the best hybrid approach to CVRPTW that our current implementation allows. At the very end, we presented the results of the ILP implementation of NRP. In the points where we noticed irregularities, we recommended options for exploration of this phenomenon and improvements or extensions to our designs.

Appendix A
Used Software

The following software was used in the development of this thesis:

- GitHub Copilot [1]
- ChatGPT (OpenAI) [2] for text style feedback suggestions
- Grammarly [3] for grammar and spelling checking
- Pycharm [4] for the programming
- Overleaf [5] for writing the thesis

[1] https://github.com/features/copilot
[2] https://chatgpt.com
[3] https://app.grammarly.com/
[4] https://www.jetbrains.com/pycharm/
[5] https://www.overleaf.com

Bibliography

[Alt+12] Leopoldo Altamirano et al. "Anesthesiology nurse scheduling using particle swarm optimization". In: *International Journal of Computational Intelligence Systems* 5.1 (2012), pp. 111–125.

[AT15] Mustafa Avci and Seyda Topaloglu. "An adaptive local search algorithm for vehicle routing problem with simultaneous and mixed pickups and deliveries". In: *Computers & Industrial Engineering* 83 (2015), pp. 15–29.

[Awa+17] Mohammed A Awadallah et al. "Hybridization of harmony search with hill climbing for highly constrained nurse rostering problem". In: *Neural Computing and Applications* 28 (2017), pp. 463–482.

[BA21] Erfan Babaee Tirkolaee and Nadi Serhan Aydın. "A sustainable medical waste collection and transportation model for pandemics". In: *Waste Management & Research* 39.1_suppl (2021), pp. 34–44.

[BB19] Kyle Booth and J. Beck. "A Constraint Programming Approach to Electric Vehicle Routing with Time Windows". In: Jan. 2019, pp. 129–145. ISBN: 978-3-030-19211-2. DOI: 10.1007/978-3-030-19212-9_9.

[BC14] Edmund K Burke and Tim Curtois. "New approaches to nurse rostering benchmark instances". In: *European Journal of Operational Research* 237.1 (2014), pp. 71–81.

[BD20] J. Blank and K. Deb. "pymoo: Multi-Objective Optimization in Python". In: *IEEE Access* 8 (2020), pp. 89497–89509.

[BHM05] Cristina Bazgan, Refael Hassin, and Jérôme Monnot. "Approximation algorithms for some vehicle routing problems". In: *Discrete Applied Mathematics* 146 (2005), pp. 27–42. URL: https://hal.science/hal-00004033.

[Bix20] Robert Bixby. *Mathematical Optimization: Past, Present, and Future (Part 2) - Gurobi Optimization — gurobi.com.* https://www.gurobi.com/resources/mathematical-optimization-past-present-and-future-part-2/. [Accessed 21-05-2024]. 2020.

[Böo+21] Elín Björk Böovarsdóttir et al. "Achieving compromise solutions in nurse rostering by using automatically estimated acceptance thresholds". In: *European Journal of Operational Research* 292.3 (2021), pp. 980–995.

[Ces+15] Sara Ceschia et al. *Second International Nurse Rostering Competition (INRC-II) — Problem Description and Rules —*. 2015. arXiv: 1501.04177 [cs.AI].

[CGS20] Sara Ceschia, Rosita Guido, and Andrea Schaerf. "Solving the static INRC-II nurse rostering problem by simulated annealing based on large neighborhoods". In: *Annals of Operations Research* 288.1 (2020), pp. 95–113.

[CLH21] Siqi Cao, Wenzhu Liao, and Yuqi Huang. "Heterogeneous fleet recyclables collection routing optimization in a two-echelon collaborative reverse logistics network from circular economic and environmental perspective". In: *Science of the Total Environment* 758 (2021), p. 144062.

[Con+06] Antonio J Conejo et al. *Decomposition techniques in mathematical programming: engineering and science applications*. Springer Science & Business Media, 2006.

[Cor] International Business Machines Corporation. *Module docplex.cp.modeler 2014; DOcplex.CP: Constraint Programming Modeling for Python V2.25 documentation — ibmdecisionoptimization.github.io*. https://ibmdecisionoptimization.github.io/docplex-doc/cp/docplex.cp.modeler.py.html. [Accessed 23-05-2024].

[Cpl22] IBM ILOG Cplex. "V22.1.1: User's Manual for CPLEX". Version V22.1.1. In: (2022).

[CW64] G. Clarke and J. W. Wright. "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points". In: *Operations Research* 12.4 (1964), pp. 568–581. ISSN: 0030364X, 15265463. URL: http://www.jstor.org/stable/167703 (visited on 06/04/2023).

[CZ20] Ping-Shun Chen and Zhi-Yang Zeng. "Developing two heuristic algorithms with metaheuristic algorithms to improve solutions of optimization problems with soft and hard constraints: An application to nurse rostering problems". In: *Applied Soft Computing* 93 (2020), p. 106336.

[Dan90] George B Dantzig. "Origins of the simplex method". In: *A history of scientific computing*. 1990, pp. 141–151.

[DC] Johann "nojhan" Dréo and Caner Candan. *Metaheuristic - Wikipedia — en.wikipedia.org*. https://en.wikipedia.org/wiki/Metaheuristic. [Accessed 24-05-2024].

[DFJ54] G. Dantzig, R. Fulkerson, and S. Johnson. "Solution of a Large-Scale Traveling-Salesman Problem". In: *Journal of the Operations Research Society of America* 2.4 (1954), pp. 393–410. ISSN: 00963984. URL: http://www.jstor.org/stable/166695 (visited on 06/04/2023).

[EK95] Russell Eberhart and James Kennedy. "A new optimizer using particle swarm theory". In: *MHS'95. Proceedings of the sixth international symposium on micro machine and human science*. Ieee. 1995, pp. 39–43.

[Fle] Alex Fleischer. *AlexFleischerParis - Overview — github.com*. https://github.com/AlexFleischerParis. [Accessed 23-05-2024].

[FLS15] Fabio Furini, Ivana Ljubić, and Markus Sinnl. "ILP and CP formulations for the lazy bureaucrat problem". In: *Integration of AI and OR Techniques in Constraint Programming: 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings 12*. Springer. 2015, pp. 255–270.

[FRM20] Reza Alizadeh Foroutan, Javad Rezaeian, and Iraj Mahdavi. "Green vehicle routing and scheduling problem with heterogeneous fleet including reverse logistics in the form of collecting returned goods". In: *Applied Soft Computing* 94 (2020), p. 106462.

[GH99] Hermann Gehring and Jörg Homberger. "A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows". In: *Proceedings of EUROGEN99*. Vol. 2. Citeseer. 1999, pp. 57–64.

[GJ20] Atefeh Hemmati Golsefidi and Mohammad Reza Akbari Jokar. "A robust optimization approach for the production-inventory-routing problem with simultaneous pickup and delivery". In: *Computers & industrial engineering* 143 (2020), p. 106388.

[Glo86] Fred Glover. "Future paths for integer programming and links to artificial intelligence". In: *Computers & operations research* 13.5 (1986), pp. 533–549.

[Gon+12] Yue-Jiao Gong et al. "Optimizing the Vehicle Routing Problem With Time Windows: A Discrete Particle Swarm Optimization Approach". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.2 (2012), pp. 254–267. DOI: 10.1109/TSMCC.2011.2148712.

[Gov+21] Kannan Govindan et al. "Medical waste management during coronavirus disease 2019 (COVID-19) outbreak: A mathematical programming model". In: *Computers & Industrial Engineering* 162 (2021), p. 107668.

[Gur23] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2023. URL: https://www.gurobi.com.

[Hà+20] Minh Hoàng Hà et al. "A new constraint programming model and a linear programming-based adaptive large neighborhood search for the vehicle routing problem with synchronization constraints". In: *Computers & Operations Research* 124 (2020), p. 105085. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2020.105085. URL: https://www.sciencedirect.com/science/article/pii/S0305054820302021.

[Has+14] Stefaan Haspeslagh et al. "The first international nurse rostering competition 2010". In: *Annals of Operations Research* 218 (2014), pp. 221–236.

[Jaf+16] Hamed Jafari et al. "Fuzzy mathematical modeling approach for the nurse scheduling problem: a case study". In: *International journal of fuzzy systems* 18 (2016), pp. 320–332.

[Joh73] David S Johnson. "Approximation algorithms for combinatorial problems". In: *Proceedings of the fifth annual ACM symposium on Theory of computing*. 1973, pp. 38–49.

[KB16] Wen-Yang Ku and J. Christopher Beck. "Mixed Integer Programming models for job shop scheduling: A computational analysis". In: *Computers & Operations Research* 73 (2016), pp. 165–173. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2016.04.006. URL: https://www.sciencedirect.com/science/article/pii/S0305054816300764.

[Kre] Alex Kressner. *Vehicle_Routing_Problem/solver/base_solver.py at master · AlexKressner/Vehicle_Routing_Problem — github.com*. https://github.com/AlexKressner/Vehicle_Routing_Problem/blob/master/solver/base_solver.py. [Accessed 23-05-2024].

[Kuž23] Simona Kuželová. "Evoluční návrh klasifikátoru EEG dat [online]". Diplomová práce. Vysoké učení technické v Brně, 2023 [cit. 2024-05-23]. URL: https://theses.cz/id/r7k790/.

[LH12] Zhipeng Lü and Jin-Kao Hao. "Adaptive neighborhood search for nurse rostering". In: *European Journal of Operational Research* 218.3 (2012), pp. 865–876.

[Lim] Staff Roster Solutions Limited. *Staff Roster Solutions - Fast, intelligent scheduling automation - Downloads — staffrostersolutions.com*. https://www.staffrostersolutions.com/downloads.php. [Accessed 24-05-2024].

[LO23] Michal Lukeš and Tomáš Omasta. *General Optimization Solver*. https://github.com/Omastto1/General-Optimization-Solver. 2023.

[Met+53] Nicholas Metropolis et al. "Equation of state calculations by fast computing machines". In: *The journal of chemical physics* 21.6 (1953), pp. 1087–1092.

[MJP21] Aye Thant May, Chattriya Jariyavajee, and Jumpol Polvichai. "An Improved Genetic Algorithm for Vehicle Routing Problem with Hard Time Windows". In: *2021 International Conference on Electrical, Computer and Energy Technologies (ICECET)*. 2021, pp. 1–6. DOI: 10.1109/ICECET52533.2021.9698698.

[MK19] David Meignan and Sigrid Knust. "A neutrality-based iterated local search for shift scheduling optimization and interactive reoptimization". In: *European Journal of Operational Research* 279.2 (2019), pp. 320–334.

[MM19] Florian Mischek and Nysret Musliu. "Integer programming model extensions for a multi-stage nurse rostering problem". In: *Annals of Operations Research* 275 (2019), pp. 123–143.

[Oma24] Tomáš Omasta. *General Solvers used on Combinatorial Problems*. Diplomová práce. 2024. URL: https://dspace.cvut.cz/handle/10467/113324.

[PF23] Laurent Perron and Vincent Furnon. *OR-Tools*. https://developers.google.com/optimization/. Version v9.6. Google, Mar. 13, 2023.

[PPF18] Hari Prasetyo, Anandistya Lisa Putri, and Gusti Fauza. "Biased random key genetic algorithm design with multiple populations to solve capacitated vehicle routing problem with time windows". In: *AIP Conference Proceedings*. Vol. 1977. 1. AIP Publishing. 2018.

[Pym] Pymoo. *pymoo - Hyperparameters — pymoo.org*. https://pymoo.org/algorithms/hyperparameters.html. [Accessed 23-05-2024].

[RAL17] Erfan Rahimian, Kerem Akartunalı, and John Levine. "A hybrid integer and constraint programming approach to solve nurse rostering problems". In: *Computers & Operations Research* 82 (2017), pp. 83–94.

[RBL96] Jacques Renaud, Fayez F Boctor, and Gilbert Laporte. "An improved petal heuristic for the vehicle routeing problem". In: *Journal of the operational Research Society* 47.2 (1996), pp. 329–336.

[RM16] Michael Römer and Taıeb Mellouli. "A direct MILP approach based on state-expanded network flows and anticipation for multi-stage nurse rostering under uncertainty". In: *Proceedings of the 11th international conference on the practice and theory of automated timetabling*. 2016, pp. 549–551.

[Rui+19] Efrain Ruiz et al. "Solving the open vehicle routing problem with capacity and distance constraints with a biased random key genetic algorithm". In: *Computers & Industrial Engineering* 133 (2019), pp. 207–219. ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2019.05.002. URL: https://www.sciencedirect.com/science/article/pii/S0360835219302694.

[SBM23] Israel Pereira Souza, Maria Claudia Silva Boeres, and Renato Elias Nunes Moraes. "A robust algorithm based on Differential Evolution with local search for the Capacitated Vehicle Routing Problem". In: *Swarm and Evolutionary Computation* 77 (2023), p. 101245. ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2023.101245. URL: https://www.sciencedirect.com/science/article/pii/S2210650223000196.

[Sola] Marius M. Solomon. *Benchmarking Problems — web.cba.neu.edu*. http://web.cba.neu.edu/~msolomon/problems.htm. [Accessed 04-Jun-2023].

[Solb] Marius M. Solomon. *Solomon benchmark — sintef.no*. https://www.sintef.no/projectweb/top/vrptw/solomon-benchmark/. [Accessed 04-Jun-2023].

[SSM21] Gaurav Srivastava, Alok Singh, and Rammohan Mallipeddi. "NSGA-II with objective-specific variation operators for multiobjective vehicle routing problem with time windows". In: *Expert Systems with Applications* 176 (2021), p. 114779. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2021.114779. URL: https://www.sciencedirect.com/science/article/pii/S0957417421002207.

[Tim14] Rong Qu Tim Curtois. *Nurse Rostering Benchmark Instances*. http://www.schedulingbenchmarks.org/nrp/. [Accessed 20-05-2024]. 2014.

[TSB15] Ioannis X Tassopoulos, Ioannis P Solos, and Grigorios N Beligiannis. "A two-phase adaptive variable neighborhood approach for nurse rostering". In: *Computers & operations research* 60 (2015), pp. 150–169.

[Uni] Ghent University. *Nurse Scheduling — Operations Research & Scheduling Research Group — projectmanagement.ugent.be*. https://www.projectmanagement.ugent.be/research/personnel_scheduling/nsp. [Accessed 21-05-2024].

[VHA11] Henno Vermeulen, Han Hoogeveen, and JM van den Akker. "Solving the no-wait job shop problem: an ILP and CP approach". In: *CPAIOR 2011 Late Breaking Abstracts* 44 (2011).

[Zva21] Adam Zvada. “Vehicle Routing Problem with Time Windows solved via Machine Learning and Optimization Heuristics”. Diplomová práce. CTU FIT, 2021. URL: https://dspace.cvut.cz/handle/10467/94492.

[ZZS18] Simone Zanda, Paola Zuddas, and Carla Seatzu. “Long term nurse scheduling via a decision support system based on linear integer programming: A case study at the University Hospital in Cagliari”. In: *Computers & Industrial Engineering* 126 (2018), pp. 337–347.

Concents of the attachment

- `text` — Folder with source files for the text
  - `plots` — Figures referenced in the text
- `General-Optimization-Solver` — Relevant parts of the repository
  - `data` — Folder for the processed instances and solutions
  - `raw_data` — Folder for the unprocessed instances
  - `src`
    - `nrp`
      - `solvers` — Folder with all the implemented solvers
      - `plots` — Folder for figures
      - `starters` — Examples for evaluation
      - `problem.py` — General VRP code
      - `stats.ipynb` — Tools for analysis
      - `utils.ipynb` — VRP utils
    - `nrp`
      - `solvers` — Folder with all the implemented solvers
      - `problem.py` — General NRP code
      - `stats.ipynb` — Tools for analysis
      - `utils.ipynb` — NRP utils
  - `src` — Folder for the unprocessed instances