FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

David Zeman

Approximate Techniques for Dynamic Vehicle Routing Problems

Faculty of Mathematics and Physics

Supervisor of the bachelor thesis:	prof. RNDr. Roman Barták, Ph.D.
Study programme:	Computer Science
Study branch:	General Informatics

Prague 2024

I wish to thank my supervisor prof. Barták for his guidance and support during the work on this thesis.

Title: Approximate Techniques for Dynamic Vehicle Routing Problems

Author: David Zeman

Department of Theoretical Computer Science and Mathematical Logic: Faculty of Mathematics and Physics

Supervisor: prof. RNDr. Roman Barták, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: This thesis studies the capacitated dynamic vehicle routing problem with changing vehicle availability. We are motivated by the increasing demand for fast and reliable delivery services in recent years. First, we analyze the problem and build its formal model. Then, we propose several policies to process dynamically appearing orders. Next, we implement exact and heuristic algorithms, such as insertion heuristics, mixed integer programming, evolutionary algorithms, and ant colony optimization. Finally, we compare the algorithms with different policies and parameters on Kilby's dataset.

Keywords: vehicle routing optimization on-line

Contents

Introduction

The vehicle routing problem (VRP) is a well-known combinatorial optimization problem studied extensively in the literature. The VRP involves designing a set of optimal routes for a fleet of vehicles to service a set of customers, subject to various constraints such as vehicle capacity, time windows, and travel time. The VRP can consider several objectives, including minimizing the total travel distance, the number of vehicles used, or the travel time. The original VRP is a static problem, meaning all the required data are known in advance. However, in practice, the data for the VRP can be dynamic and change over time.

That is usually called the dynamic vehicle routing problem (DVRP). Dynamism in the DVRP can arise from several factors, such as customer demands, travel times, vehicle availability, and changes in the operating environment. In recent years, the demand for fast and reliable delivery services has increased, so the DVRP has become an essential problem in logistics and transportation and has many applications.

This thesis studies a version of the DVRP with vehicle capacity constraints, orders revealing over time, and changing vehicle availability. The changing vehicle availability is assumed simplified - the vehicle can become unavailable after the delivery of an order. However, the number of vehicles is unlimited so that all orders can be delivered. The objective of CDVRP is to minimize the total travel distance and satisfy capacity constraints. Since the VRP is known to be NP-hard, the CDVRP is even more challenging to solve. Because of that, we do not focus only on exact algorithms but also study heuristic and meta-heuristic algorithms.

The goal of this thesis is to implement several algorithms for the CDVRP with changing vehicle availability, use them with a policy, and compare their performance under different scenarios.

Structure of the thesis

In the first chapter, we introduce the problem and motivation for the study. In the second chapter, we provide a literature review of the VRP and DVRP and briefly discuss the algorithms and approaches used to solve these problems. The third chapter defines the problem precisely, including the mathematical formulation and constraints. It also discusses problems with optimality in dynamic environments. Then, we propose several policies on how to deal with dynamism. The fourth chapter presents the algorithms used to solve the CDVRP, including exact, heuristic, and meta-heuristic algorithms. The last chapter presents the results of the experiments and compares the algorithms for different periods, policies, and order distributions. Finally, we conclude the thesis and discuss possible directions for future research.

1. Vehicle routing problem

1.1 Traveling salesman problem

The vehicle routing problem (VRP) is a generalization of the traveling salesman problem (TSP), which was already known in the 19th century. It was first formulated probably by W.R. Hamilton in 1859, who also defined the Hamiltonian cycle on graphs. The goal of the classical TSP is to find the shortest closed route that visits all places. Since it is known to be NP-hard, there is a lot of research on approximation algorithms and heuristics.

Formulations

A TSP can be naturally formulated using graph theory. Let $G = (V, E)$ be a graph. Then the set of vertices $V = \{v_1, v_2 \dots v_n\}$ corresponds to places, and the set of edges $E \subseteq \binom{V}{2}$ corresponds to connections between them. Usually $E = \binom{V}{2}$ is assumed, so $G \cong K_n$. Finally, let $d : V^2 \to \mathbb{R}$ be a distance function. If d is symmetric, the problem is known as symmetric TSP; if d is a metric, it is called a metric TSP. The task is to find cycle $C \subseteq E$ that visits each vertex and $\sum_{e \in C} d(e)$ is minimal. If G is complete, then it is equivalent to finding a permutation of vertices p that $\sum_{v_i, v_{i+1} \in p} d(v_i, v_{i+1})$ is minimal.

The TSP can also be formulated as an integer program. Two main formulations exist. The first one was introduced in Dantzig et al. [1954], often called the Dantzig–Fulkerson–Johnson (DFJ) formulation. The second formulation was introduced in Miller et al. [1960], which is called Miller–Tucker–Zemlin (MTZ) formulation. Both formulations come with decision variables $x_{ij} \in \{0, 1\}$, which indicate whether an edge between vertices i and j is used. The objective is then:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} d(i, j) x_{ij}$$

Figure 1.1: MIP objective for TSP

The main difference between these formulations is that DFJ formulation states exponentially many constraints to avoid cycles. That is very impractical since even processing the program requires exponential time. The MTZ formulation solves this issue using additional integer variables $u_1 \dots u_n$ that keep values of order visits.

Approximation algorithms

The TSP is known to be NP-hard but not NP-complete. It is not NP-complete because it can be reduced to the problem of finding the Hamiltonian path, which is not NP-complete since even its decision version is. Because of its hardness, we should be motivated to look for approximation algorithms. No approximation algorithm can exist in non-symmetric TSP since its optimum may be 0, meaning that any approximation with approximation ration k algorithm must find the optimal solution.

Two main deterministic approximation algorithms for metric TSP exist. The first one finds a minimum spanning tree S and then "walks it around". Let H denote the optimal solution of TSP on a graph $G \cong K_n$. Since the $d(S) \leq d(H)$, then the length of the path around S is at most $2d(S) \leq 2d(H)$. If d is metric, where triangular inequality holds, we can even shorten the path by not going back to the same vertex but directly to the next one.

The second algorithm comes from Christofides [1976]. It is a little more sophisticated, and it is called the Christofides algorithm. It starts the same as the previous one but then improves a solution, finding minimum-weight perfect matching M on vertices of odd degrees. The first algorithm can use each edge from S at most twice, but the Christofides uses each edge from S at most once, and then it uses edges from minimum-weight perfect matching. Since $d(M) \leq d(H)/2$ then $d(S) + d(M) \leq \frac{3}{2}d(H)$. Both minimum weight spanning tree and minimum weight perfect matching can be found in polynomial time.

1.2 Vehicle routing problem

The VRP is a natural generalization of TSP where multiple vehicles are available, so multiple routes can be planned. The VRP was first formally described in Dantzig and Ramser [1959]. Many generalizations and variants of VRP with different constraints and objectives exist. This section introduces some classical ones. Several excellent surveys were already written, like Caceres Cruz et al. [2014] or Ojeda Rios et al. [2021].

Variants and constraints

The specific variant of VRP determines constraints that a solution must hold. Generally, two types of constraints are distinguished, hard and soft. Hard constraints must be satisfied in every feasible solution. On the other hand, soft constraints do not have to hold every time. There are many known techniques to handle soft constraints, like penalty functions or constraint hierarchies.

Typical hard constraints guarantee connectivity of roads or sufficient capacity of a vehicle, while soft constraints may deal with time windows, service quality or response time.

In the capacitated vehicle routing problem (CVRP), each vehicle has a limited amount of resources it can carry. The capacity may be equal for all vehicles or not. Also, each customer may require a specific amount of resources. When it is impossible to serve all customers, we may allow multiple trips for one vehicle. We can also assume partial deliveries, meaning part of the customer's order is delivered by one vehicle and the rest by another.

In the vehicle routing problem with time windows (VRPTW), each customer must be served only in the specific time window. We also may take service time into account. Working with a time dimension here is essential, making the problem much harder and more difficult to model.

If there are multiple depots, each with its fleet of vehicles, then it is called a multi-depot vehicle routing problem (MDVRP). Each depot may also have limited resources to distribute among customers. The fleet of vehicles may be heterogeneous, meaning each vehicle has a different capacity or capabilities.

In some real-life scenarios, the resource source may not be a depot, but we must transport resources between nodes. That is called the pickup and delivery vehicle routing problem (PDVRP). In PDVRP, we can also allow transfers of resources between depots.

If the information needed for solving VRP reveals or changes over time, it is called the dynamic vehicle routing problem (DVRP). One variant of DVRP is called the dial-a-ride problem (DARP), a dynamic version of the pickup and delivery vehicle routing problem.

One recent variant of VRP is the green vehicle routing problem (GVRP), which focuses on minimizing environmental impact. On the other hand, it tries to keep the service reliable and profitable. To minimize environmental impact, we may want to use routes that avoid heavy traffic areas, or we may prioritize electric vehicles or public transport usage.

It is not exactly a variant, but a similar problem is the vehicle rescheduling problem (VRSP) that solves the task of updating the current schedule on unexpected occasions. It was studied in Li et al. [2007].

Finally, if there are some uncertainties in the environment or the results of actions are probabilistic, then we call it the stochastic vehicle routing problem (SVRP).

The variants above are just the most essential ones, and most real-life VRP-like problems combine them. For example, let's think about meal delivery service in a city. We definitely have to use capacity and time-window constraints. Then, there are multiple resources to deliver that are located in different places. Also, orders for delivery reveal over time, and the city environment is rather stochastic than deterministic. And lastly, delivery couriers may start and end their shifts unexpectedly.

The more constrained the problem is, the more difficult it is to find a feasible solution. That is a huge difference from classical VRP, where any permutation of vertices is feasible. It causes complications in, e.g., evolution-based algorithms where a large amount of random valid solutions must be generated in the beginning, which is usually hard. If they are generated with some greedy heuristic, it often leads to low diversity and a degenerated population.

Objective function

Another important part of any vehicle routing problem variant is its objective function. It is a function that takes the solution and returns real value proportional to solution quality. Some techniques, like mixed integer programming or gradient descent, require the objective function to be in special (linear) form, or its derivative must exist. The objective function also directs meta-heuristics-based algorithms toward better solutions, like in ant colonies, evolutionary algorithms, or advanced search algorithms. Here, it is often called the fitness function because many of those algorithms are inspired by nature, where an individual's fitness determines their chance to survive, respectively, to be a good solution.

The simplest objective for classical VRP is minimizing total distance or cost. This is the same as the objective of the integer program formulation of TSP.

Another common objective is to minimize total number of vehicles needed. To achieve that, we may minimize the number of vehicles directly or set up vehicle usage costs. In some pickup and delivery variants, the total waiting time or lateness may be minimized. Sometimes, it is easier to maximize total profit than to minimize total costs, e.g., when we are not required to serve all customers. For such complicated scenarios, we may use multi-objective optimization, where we try to find a trade-off between multiple objectives.

In the stochastic vehicle routing problem, the objective is usually considered an expected objective value over probabilistic distribution.

1.3 Dynamic VRP

Dynamic vehicle routing problem extends classical VRP with information revealed over time. It usually means that customer orders come as time passes. Another dynamic component can be travel time affected by real-time traffic, unexpected travel costs, or vehicle availability. In many dynamic scenarios, it is necessary to deal with time since orders can be served only in a specific time window. There is also more pressure on algorithm speed since a customer often requires a response as soon as possible, which is not the case in static VRP, where a plan is created before execution.

Measuring dynamism

Measure degree of dynamism may help us better understand the problem's essence and select the optimal solution method. Lund et al. [1996] defined the *degree of dynamism* as ratio of orders added over time n_d and total number of orders n. This measure is very simple and does not tell us much about the problem.

$$\delta = \frac{n_d}{n}$$

Another approach was introduced in Larsen [2000] as the *effective degree of dynamism*. It is the average of orders O disclosure times t_o normalized by a number of all orders n. The value δ^e increases with more orders revealed later, but does not consider their distribution.

$$\delta^e = \frac{1}{n} \sum_{o \in O} \frac{t_o}{T}$$

The δ^e can also be extended for dynamic vehicle routing problems with time windows where l_o denotes the end of the time window of order $o \in O$. The formula is then extended as follows:

$$\delta^e = \frac{1}{n} \sum_{o \in O} \frac{l_o - t_o}{T}$$

The difference $l_o - t_o$ corresponds to reaction time, so orders with longer reaction time contribute to a total degree of dynamism less than orders with short reaction time.

Policy

A policy is a set of rules on when and how to update a plan when new information is revealed. There are two general policies mentioned in Pillac et al. [2013].

The first policy is *periodic policy.* It stores new orders in a buffer and periodically solves static instances of a problem. The main advantage is that standard solution techniques from static VRP can be used. On the other hand, it does not support immediate response. This method is great for DVRP instances with relatively low levels of dynamism.

The second one is *online policy*, which updates a plan every time a new order arrives. It holds the plan in memory, and every time a new order arrives, it exploits current knowledge and inserts the new order into existing routes. Usually, the physical vehicle knows just its following target since the route changes over time. The advantage is that we do not have to solve the whole instance repeatedly, but it is more challenging to implement. For implementing continuous policy, we need an algorithm that can hold one or more possible solutions in memory and adapt them to new information. This method is great for DVRP, where the level of dynamism is relatively high.

Practical aspects

There are several practical aspects of the dynamic vehicle routing problem. At first, we need to communicate with vehicles and customers in real-time, which may be a challenge. This is usually possible via the Internet or other networks in populated areas, but it may be a problem in the countryside or rural countries. Second, it is necessary to have reliable information about the vehicle's location and possibly about the current traffic situation. Last, we should be able to deal with unexpected situations like crashes or traffic jams.

Applications

The dynamic vehicle routing problem has numerous applications across various industries and sectors, reflecting its relevance to real-world logistics and transportation challenges. Some of these applications include last-mile delivery, on-demand transportation, emergency response, and waste collection, among others.

In the context of last-mile delivery, as e-commerce continues to grow, efficient last-mile delivery has become increasingly important. DVRP can help optimize the routing of delivery vehicles, taking into account real-time changes in customer demands, traffic conditions, and vehicle availability. By addressing these dynamic factors, DVRP algorithms can improve delivery efficiency, reduce transportation costs, and enhance customer satisfaction.

On-demand transportation has gained prominence with the rise of ride-hailing services and on-demand transit systems. This creates a need for dynamic routing algorithms that can handle fluctuating customer orders and traffic conditions. DVRP can be used to optimize vehicle dispatching and routing in these systems, ensuring that customers are picked up and dropped off in a timely and efficient manner.

Emergency response is another critical application area for DVRP. In situations such as natural disasters, medical emergencies, or search and rescue operations, the ability to quickly adapt to changing circumstances is crucial. DVRP algorithms can help optimize the routing and dispatching of emergency vehicles and resources, taking into account real-time information about the locations of incidents, traffic conditions, and the availability of resources.

Waste collection is yet another application where DVRP can be beneficial. Municipalities and waste management companies need to efficiently plan and execute waste collection routes, considering factors such as varying waste generation rates, vehicle capacities, and traffic conditions. DVRP can help optimize waste collection routes, minimize the distance traveled, and reduce operational costs while ensuring timely service to all customers.

Overall, the DVRP plays a vital role in addressing the real-world challenges of various industries and sectors. By developing and applying effective DVRP algorithms, significant improvements can be achieved in the efficiency and performance of logistics and transportation systems, ultimately leading to better services, reduced costs, and enhanced sustainability.

2. Related work

Due to its importance in various real-world applications, the dynamic vehicle routing problem has gained significant research attention over the years. In this chapter, we review the related work on DVRP, focusing on the different problem variants, modeling approaches, and solution methodologies proposed and analyzed in the literature.

One key aspect of DVRP research is the various problem variants that have been studied. These variants often arise from the combination of dynamic features with well-known VRP extensions already mentioned above.

Modeling approaches for DVRP have been diverse, including mathematical programming formulations, constraint programming, and agent-based models. These approaches often aim to capture the complexity and uncertainty associated with the dynamic nature of the problem while ensuring the reliability and computational efficiency of the resulting models.

Solution techniques for DVRP have ranged from classical heuristics and metaheuristics, such as local search, tabu-search, and evolutionary algorithms, to more recent approaches that leverage machine learning and artificial intelligence techniques. Researchers have also explored hybrid methods that combine the strengths of different algorithms and strategies and parallel and distributed computing frameworks to address the computational challenges of DVRP.

Several brilliant reviews on DVRP already exist, like Pillac et al. [2013] and more recent Ojeda Rios et al. [2021].

2.1 Heuristics

Heuristics are usually based on simple rules to update a solution. They are quite popular in dynamic problems since they usually provide both good quality solutions and short runtime. A heuristic can be strictly deterministic, or *ϵ-greedy* approach can be used to generate multiple solutions and select the best one. Heuristics can also be used to generate initial solutions for evolution-based algorithms.

Insertion heuristics

Insertion heuristics construct a solution by adding nodes according to a rule. Randall et al. [2022] have introduced five insertion heuristics for the capacitated dynamic vehicle routing problem. The *Sequential insertion* inserts requests greedily to a route until no order can be inserted. Then it creates a new route and repeats the process. The *Quasi-sequential insertion* tries to insert a new request into any route instead of just into the current one. The *Naive parallel insertion* tries to insert a new request into every route and select the best option. The *Parallel insertion with seeds* behaves the same as the naive version, but first m requests, where m is a number of vehicles, are "seeded" into m different routes to avoid degeneration. The fifth heuristic is called *Sum of squares insertion*. It is similar to the naive approach but minimizes squared route lengths instead. The minimization of squared route lengths causes all routes to be similarly long.

Reoptimization heuristics

In the reoptimization heuristic, a solution is updated each time a new request arrives. One such approach was introduced in Steever et al. [2019]. The authors created the MIP model that may be solved exactly using standard branch-bound or branch-cut methods. However, the time required to solve large instances of DVRP makes this direct approach unusable. Instead, they introduced two auction-based methods that solve the problem for each vehicle separately, and overlapping routes are then adjusted by auction results. The first method is called *Myopic variant* and uses just the highest bid auction, where a request is assigned to the vehicle with the highest bid. The height of a bid is proportional to the vehicle's objective function. The second method, called *Proactive variant*, uses the same auction, but it also takes measures of decentralization or dispersion into account.

2.2 Exact methods

Exact methods ensure the optimal or near-optimal solution in the static vehicle routing problem. But in a dynamic environment, it is not that clear what *optimal* solution is. One point of view may be that the optimal solution of DVRP is a subset of the static VRP solution with all information. Another possibility is to define optimally concerning currently known information. Most of the authors use the second variant since it is more practical to evaluate solutions over time.

Mixed integer programming

The most common exact method used is mixed integer programming (MIP).

The advantages of MIP are that the formulations offer great modeling flexibility, as they can easily incorporate complex constraints and relationships. This allows for the modeling of various DVRP variants, including those with time windows, stochastic requests, and multiple depots. This flexibility enables researchers to adapt the model to a wide range of practical applications. Additionally, MIP methods can provide optimal or near-optimal solutions, depending on the solver's stopping criteria. This feature is particularly useful when benchmarking heuristics or meta-heuristics, as the MIP solution serves as a reference for evaluating their performance. Moreover, there are several powerful commercial solvers available, such as SCIP, CPLEX, and Gurobi, designed to efficiently solve MIP problems, which can take advantage of advanced algorithms and parallel computing capabilities

Authors in Liu [2019] introduced the MIP model with an online dispatch algorithm for drone delivery. A similar approach was used in Gaul et al. [2022] for the dial-a-ride problem. They created a MIP model and used a rolling event-based horizon algorithm. Every time a new request arrives, MIP is solved within a fixed time limit, and the request is either accepted or declined.

One major disadvantage of MIP methods for DVRP is their computational complexity. As the problem size increases, the number of decision variables and constraints in the MIP formulation can grow exponentially, leading to long computation times and high memory requirements. This limits the applicability of MIP methods to small or moderately sized DVRP instances. Another disadvantage is the difficulty in incorporating real-time information and dynamic changes in the problem. Traditional MIP methods are designed for static optimization problems and may require significant modifications or resolving the problem from scratch when new information becomes available. This can be computationally expensive and impractical for highly dynamic scenarios where quick response is essential.

2.3 Meta-heuristics

In Sörensen and Glover [2013], a meta-heuristic is defined as a high-level and problem-independent algorithmic framework that provides guidelines or strategies to develop heuristics for specific optimization problems.

They are particularly useful for combinatorial optimization problems, where the solution space is large, and exact methods may be computationally infeasible. Meta-heuristics must balance the trade-off between exploration and exploitation to converge toward the global optimum. Exploration means searching for new regions of the solution space, and exploitation stands for improving the solutions within the current region. This is especially important for problems where information is revealed over time since previous solutions can be exploited.

Swarm algorithms

Swarm algorithms are a family of mostly nature-inspired optimization techniques that rely on collective intelligence. These algorithms mimic the behavior of social insects, bird flocks, or fish schools to solve optimization problems. The main principle of swarm algorithms is to allow a population of simple agents to interact with each other and their environment, following simple rules to achieve global optimization.

ACO

Ant colony optimization (ACO) is a nature-inspired meta-heuristic optimization algorithm based on the foraging behavior of ants. They search the solution space iteratively and deposit and follow pheromones on paths depending on how good a solution is. It was first proposed in the early 1990s, e.g., in Colorni et al. [1991], where it was used for solving the TSP.

There exist many variants based on ACO. In ant colony systems (ACS), ants place pheromones on trails in real time, not only at the end of a run. In the min-max ant systems (MMAS), the minimum and maximum amount of pheromone on the path is bounded by constants. Also, elitism can be used to preserve the best solutions over generations.

Schyns [2015] introduced an algorithm based on the ant colony system to solve the capacitated dynamic vehicle routing problem with time windows. They adapted the classical ACO algorithm for static VRP to a dynamic version and added constraints for capacities and time windows. One of their goals was to provide a response as soon as possible, and their experiments showed that ACO made it possible.

PSO

Particle Swarm Optimization (PSO) is a population-based meta-heuristic optimization algorithm inspired by the social behavior of bird flocking and fish schooling. Introduced by Kennedy and Eberhart [1995], PSO has gained significant popularity for its simplicity, ease of implementation, and effectiveness. The algorithm represents each candidate solution as a particle in a multidimensional search space, and particles iteratively update their positions based on their own best-known positions and the best-known positions of other particles in the swarm. Through this process, particles explore the search space, converging toward the optimal or near-optimal solution.

A PSO-based approach was introduced by Okulewicz and Mańdziuk [2019]. The paper tested a hypothesis that the selection of proper search space is more important than the choice of solving technique. The authors compared PSO and differential evolution (DE) in two different continuous search spaces with classical discrete implementation with GA and showed that continuous representation outperformed discrete one.

Evolutionary algorithms

Evolutionary algorithms (EAs) are a family of optimization algorithms inspired by natural evolution and have been successfully applied to various optimization problems, including the dynamic vehicle routing problem (DVRP). The general structure of EAs consists of initialization, evaluation, selection, variation, replacement, and termination stages.

In the context of DVRP, evolutionary algorithms can be designed to handle dynamic changes in the problem environment, including new customer requests, travel times, or vehicle breakdowns. Techniques like incorporating memory structures, using adaptive variation operators, or employing multi-objective optimization approaches can be used to balance conflicting objectives, such as minimizing travel time and maximizing customer satisfaction.

Although evolutionary algorithms have shown promising results in solving DVRP instances of varying complexity, they may require careful tuning of algorithm parameters and problem-specific heuristics to achieve high-quality solutions within reasonable computational times. EAs offer a robust and adaptable approach to solving dynamic vehicle routing problems, making them a popular choice among researchers and practitioners.

There are also several problems related to EAs. At first, they need a large and diverse population of solutions in the beginning, which may not be easy to create, especially in hardly constrained problems. Second, it is hard to design genetic operators that do not break any constraints, so we usually need to deal with infeasible solutions. Those infeasible solutions may be thrown away or repaired with procedures generally called repair operators.

Hanshar and Ombuki-Berman [2007] proposed genetic algorithm for dynamic vehicle routing problem. An event scheduler runs the algorithm on static VRP every time an event occurs. The algorithm uses variable chromosome lengths with two types of genes. The first type, represented by a positive integer, corresponds to the customer that has not been assigned to any vehicle yet. The second type, represented by a negative integer, corresponds to a group of clustered customers that have already been committed to a vehicle. Authors proposed a specific crossover called *best-cost route crossover* that generates valid chromosomes. Next, they used common inversion mutation, *k-tournament* selection, and 1% elitism.

Tabu-search

Tabu-search was developed by Glover [1986]. It is a meta-heuristic optimization algorithm designed to solve combinatorial optimization problems across various domains. It uses memory structures, known as tabu lists, to guide the search process by avoiding cycling and promoting exploration of the solution space. The algorithm starts from an initial solution and iteratively explores the neighborhood of the current solution. Balancing exploration and exploitation is achieved by adjusting the tabu tenure parameter and incorporating other search strategies or heuristics.

The success of tabu search in solving a wide range of optimization problems is attributed to its ability to escape local optima and effectively explore the solution space. However, the algorithm requires careful tuning of parameters, such as the tabu tenure and neighborhood structure, to achieve optimal performance. Additionally, tabu search can be computationally expensive, particularly for large-scale problems.

Ferrucci and Bock [2016] proposed a tabu search for the dynamic vehicle routing problem with soft time windows constraints. They evaluated the algorithm on a real dataset and showed that it found the near-optimal solution in a very short time.

2.4 Kilby's dataset

One of the most popular datasets for DVRP is the Kilby dataset introduced in Kilby et al. [1998]. Source files are now available in Okulewicz [2021] with additional information; original source does not work anymore. The dataset is based on static VRP instances, and availability times are generated uniformly randomly. Since the dataset is very popular and optimal or near-optimal values are known, we will also use it for our experiments.

3. Problem formulation and its analysis

In this chapter, we precisely define the version of the capacitated dynamic vehicle routing problem with changing vehicle availability. First, we list its most essential properties and create a formal model and notion that will be used in the following algorithms for clarity. Second, we state the constraints of a feasible solution and the objective function of the problem. Next, we show the solution cost upper bound for dynamic TSP in relation to static TSP and briefly mention problems with defining optimality in DVRP. Last, we propose several policies that can deal with dynamically revealing orders and changing vehicle availability.

3.1 Properties

In this work, we will examine the dynamic vehicle routing problem with the following properties:

- Single depot
- Orders reveal over time
- Homogenous fleet of vehicles with fixed capacity and constant speed
- Availability of vehicles changes over time
- Metric space

We assume a single-depot variant, but most of the presented algorithms can be easily extended to a multi-depot version.

The first dynamic component of the problem is the revelation of orders over time. Each order must be served completely by one vehicle. We assume the orders are served immediately, so we do not include additional service time. That is not very realistic, but all algorithms can be easily extended to consider it, and the omission simplifies the implementation. As time passes, vehicles are on the way. So, with new orders revealed, the algorithm cannot change the parts of routes with already served orders.

The second dynamic component is the changing availability of vehicles. More precisely, after a vehicle serves an order, it may become unavailable, so it cannot serve any other orders. The event is not known in advance. However, the total number of vehicles in the depot is not limited, so all orders can be served. All vehicles have a fixed capacity and a constant speed of one distance unit per tick. Last, we want all vehicles to return to the depot at the end of the planning period.

All algorithms and support procedures assume metric space where distances are symmetric, and the triangular inequality holds.

Since the problem is dynamic, we need to deal with time. That is done by a policy - a strategy that decides when to run the optimization procedure.

3.2 Formal model

In this section, we describe a formal model for the capacitated dynamic vehicle routing problem with changing vehicle availability. We have V homogenous vehicles with capacity C that are located in a single depot in the beginning. The depot is a node with an index 0. Let O denote the set of orders to be scheduled and served. Orders are indexed from 1 to n. The set O is updated every time a new order is revealed. Each order $o \in O$ has its requested amount of a resource noted $r(o)$. Next, we have a distance matrix D, which provides distances between orders and the depot. Finally, we define P as the probability that a vehicle becomes unavailable after serving an order and $u(v)$ as the time when it happens.

- $t \in \mathbb{N}$ - current time
- $T \in \mathbb{N}$ - total time of planning
- $V \in \mathbb{N}$ - number of vehicles
- $C \in \mathbb{N}$ - capacity of a vehicle
- $O = \{1, 2 \dots n\}$ - set of orders to be scheduled
- $r(o) \in \mathbb{N}$ - requested amount of a resource by order $o \in O$
- $a(o) \in \mathbb{N}$ - time when an order $o \in O$ is placed
- $s(o) \in \mathbb{N}$ - time when an order $o \in O$ was or will be served
- $N = O \cup \{0\}$ - set nodes, where 0 is the depot
- $D \in \mathbb{R}^{|N| \times |N|}$ - distance matrix
- $d(i, j) := D_{ij}$ - distance function
- $\mathcal{R}$ - set of routes, each route is an ordered tuple of nodes
- $P \in [0, 1]$ - probability that vehicle becomes unavailable
- $\mathcal{R}_f$ - set of finished routes, where vehicle became unavailable
- $u(v) \in \mathbb{N}$ - time when vehicle v became unavailable, default ∞

3.3 Solution and objective function

Next, we describe a solution we expect and how to measure its feasibility and quality. We will use the following constraints in optimization algorithms to check the solution feasibility and the objective function to evaluate its quality.

The solution S consists of a set of routes $\mathcal{R}$ where the following constraints hold.

$$\forall R \in \mathcal{R} : \sum_{o \in R} r(o) \leq C \tag{3.1}$$

$$\forall o \in O : a(o) \leq s(o) \tag{3.2}$$

$$(\forall R_v \in \mathcal{R})(\forall o \in R_v) : s(o) \leq u(v) \tag{3.3}$$

The constraint 3.1 ensures that the capacity of the vehicle on every route is not exceeded. The constraint 3.2 says that each order must be served after it is revealed. The constraint 3.3 ensures that unavailable vehicles do not serve any orders.

Last, the objective 3.4 is to minimize the total length of all routes.

$$\min \sum_{R \in \mathcal{R}} \sum_{i=0}^{|R|-1} d(R_i, R_{i+1}) \tag{3.4}$$

In dynamic CVRP, we also need to be able to decide whether an order $o_i \in O$ on route $R = (o_1 \dots o_{i-1}, o_i \dots)$ can be rescheduled or not at time t. That can be done by checking if $t \leq s(o_{i-1})$. If it holds, the order can be rescheduled. Otherwise, the vehicle from o_{i-1} is already on the way to o_i, so we cannot schedule o_i to another vehicle.

The delivery time $s(o_i)$ of an order $o_i \in R$ is given by the following recursive formula:

$$s(o_i) = \max\{s(o_{i-1}), a(o_i)\} + d(o_{i-1}, o_i) \tag{3.5}$$

3.4 Optimality upper bound

This section shows how the number of orders revealed at a time influences the optimality of the dynamic vehicle routing problem compared to the static one. We use a simple VRP model with one depot, one vehicle, no capacities, and no other constraints and with n orders in metric space, which is de facto the metric TSP model. We also allow returns to the depot.

First, note OPT_s as the length of the optimal tour for static TSP. Second, assume the same input for dynamic TSP with one order revealed at the time. In the worst case, we will plan n tours for each order separately, and the length of each tour is bounded by OPT_s. So the total solution length is bounded by $n * OPT_s$, which means the solution for dynamic TSP is at most n times worse than for static one. Finally, assume dynamic TSP with m orders revealed at the time. In the worst case, we will plan $\frac{n}{m}$ tours separately, and the length of each tour is still bounded by OPT_s. So the total length of all tours is bounded by $\frac{n}{m} * OPT_s$. Observe that as $m \to n$, the optimality ratio tends to 1.

We can formulate the following observation:

Observation 1. *The route length of dynamic TSP with m orders revealed at the time and n orders in total is at most $\frac{n}{m}OPT_s$ where OPT_s is the optimal route length of the static problem.*

We have shown that the more information we have at the time, the better the solution can be expected.

3.5 Optimality problems

The concept of optimality is not well defined in the dynamic vehicle routing problem. If we consider the optimal solution as the one the static problem would produce, then the optimal solution is usually not achievable in the dynamic case. On the other hand, if we consider the optimal solution to be the one that minimizes the total length with respect to current information, then other problems appear. For example, the algorithm may make a bad decision with respect to current information, but it shows up to be great with respect to future orders. Because of that, the optimal algorithm for the dynamic vehicle routing problem should be non-deterministic, but then the solution would be the same as for the static problem.

Regardless of those problems, our objective remains to minimize the total length of all routes. We may not be able to say whether the results are optimal, but we can still compare different algorithms and approaches.

3.6 Policies

This section introduces policies for CDVRP. The main task of the policy is to deal with dynamically revealed orders and changing vehicle availability and decide when to run the optimization procedure. They are inspired by Pillac et al. [2013]. The first policy is based on an online approach, where the algorithm reacts to new orders immediately. The second policy is based on a periodic approach, where the algorithm reacts to newly revealed orders and vehicle availability changes periodically. Last, we briefly mention other possible options.

Online policy

First, we introduce an online policy. The algorithm consists of a loop that runs the optimization procedure every time a new order is revealed. Then it checks whether a vehicle becomes unavailable after serving an order. If so, it disables the vehicle and reschedules its orders to other vehicles.

Algorithm 1 CDVRP online policy

```
1: $S \leftarrow \emptyset$                                   ▷ Initial solution
2: for $t \leftarrow 1 \ldots T$ do
3:     if order $o$ received then
4:         Optimize($S, \{o\}$)
5:     end if
6:     if order $o$ on route $R_v$ served at time $t$ then
7:         if Rand($0, 1$) $< P$ then
8:             $\mathcal{R}_f \leftarrow \mathcal{R}_f \cup R_v$          ▷ Vehicle becomes unavailable
9:             $X \leftarrow \{o : o \in R_v \wedge s(o) > t\}$
10:            Optimize($S, X$)                   ▷ Reschedule remaining orders
11:        end if
12:    end if
13: end for
```

Periodic policy

Second, we introduce a periodic policy with period T_p. Observe that the period parametrizes the response delay and the amount of information available to the optimization procedure. As we have shown in observation 1, the longer the interval is, the better solution can be expected. In fact, the online policy is a special case of the periodic policy with $T_p = 1$.

Algorithm 2 CDVRP periodic policy

```
1: $S \leftarrow \emptyset$                                   ▷ Solution
2: $Q \leftarrow \emptyset$                                   ▷ Queue of orders
3: for $t \leftarrow 1 \dots T$ do
4:     if order $o$ received at time $t$ then
5:         Enqueue($Q, \{o\}$)
6:     end if
7:     if order $o$ on route $R_v$ served at time $t$ then
8:         if Rand$(0,1) < P$ then
9:             $\mathcal{R}_f \leftarrow \mathcal{R}_f \cup R_v$        ▷ Vehicle becomes unavailable
10:            $X \leftarrow \{o : o \in R_v \wedge s(o) > t\}$
11:            Enqueue($Q, X$)                 ▷ Enqueue remaining orders
12:        end if
13:    end if
14:    if $t \mod T_p = 0$ then                      ▷ Period $T_p$
15:        Optimize($S, Q$)
16:        $Q \leftarrow \emptyset$
17:    end if
18: end for
```

Other policies

The online and periodic policies are not the only possible ways how to deal with incoming orders. We may also consider a cumulative policy that runs an optimization procedure after a fixed number of orders are revealed. Other policies may run the optimization procedure only if the total demand is high enough, so the vehicle can be fully utilized. The algorithm then looks the same as the periodic policy, but the condition on the row 14 for running the optimization procedure is different. We will test several policies in the experiments.

4. Optimization algorithms

This chapter describes several optimization algorithms for the capacitated dynamic vehicle routing problem with changing vehicle availability. First, we implement a mixed-integer program to obtain the exact solution for smaller instances or at least feasible solutions for larger ones. Second, we implement an insertion heuristic based on Randall et al. [2022] and extend it. Next, we use meta-heuristic approaches based on evolutionary algorithms and ant colony optimization. All algorithms are modified to deal with dynamic orders and changing vehicle availability.

All the algorithms solve the static problem. They are not online by nature. To solve the dynamic problem, we will use them with some policies described in the previous chapter. The policy then handles the dynamism, and decides, when to run the optimization algorithm.

4.1 Mixed integer programming

First, we create a mixed integer programming model to obtain the optimal or at least feasible solutions. We base the MIP formulation on the one from Munari [2016] for CVRP, which is based on MTZ formulation from Miller et al. [1960]. The program uses binary decision variables x_{ij} to indicate whether the edge between nodes $i \in N$ and $j \in N$ is used. Next, it uses real variables y_i for each $i \in N$ corresponding to cumulated demand on the route to deal with vehicle capacities. Current time is denoted t.

$$\begin{aligned}
&\min && \sum_{i \in n} \sum_{j \in n} x_{ij} D_{ij} && && (4.1)\\
&\text{s.t.} && \sum_{\substack{j \in N\\ i \neq j}} x_{ij} = 1 && \forall i \in O && (4.2)\\
& && \sum_{\substack{i \in N\\ i \neq j}} x_{ij} = 1 && \forall j \in O && (4.3)\\
& && \sum_{i \in O} x_{i0} \leq V && && (4.4)\\
& && y_j \geq y_i + r(j) x_{ij} - C(1 - x_{ij}) && (\forall i \in O)(\forall j \in N) && (4.5)\\
& && y_i \geq r(i) && \forall i \in O && (4.6)\\
& && s(o_{i-1}) < t \implies x_{o_{i-1}, o_i} = 1 && (\forall R \in \mathcal{R})(\forall o_i \in R) && (4.7)\\
& && x_{o_{i-1}, o_i} = 1 && (\forall R_f \in \mathcal{R}_f)(\forall o_i \in R_f) && (4.8)\\
& && x_{ij} \in \{0, 1\} && (\forall i \in N)(\forall j \in N) && (4.9)\\
& && y_i \in \langle 0, C \rangle && \forall i \in N && (4.10)
\end{aligned}$$

The program has $\mathcal{O}(n^2)$ variables and $\mathcal{O}(n^2)$ constraints. That is significantly better than the original Dantzig formulation, where the number of constraints grows exponentially concerning the number of nodes. The 4.1 is the objective function that minimizes the total distance traveled. The 4.2 and 4.3 ensure that the number of incoming and outgoing edges equals one for each order node so that every order is served exactly once. The 4.4 states that the number of incoming routes into the depot is, at most, the total number of vehicles. Since routes are connected, it also bounds the number of outgoing routes. The 4.5 and 4.6 ensure that the vehicle's capacity is not exceeded and all demands are satisfied. The constraint 4.7 ensures that the order is rescheduled only if it has not been served yet or the vehicle is not already on the way to it. Last, the constraint 4.8 sets all already finished routes, where a vehicle became unavailable. Those routes cannot be changed anymore.

4.2 Insertion heuristic

The insertion heuristic algorithm gets an order or a set of orders and then tries to insert or append them into existing routes with minimal cost. If there is no suitable route, then it adds a new one. The insertion heuristic is excellent for use in online policy since it usually runs very fast. But it can also be used in periodic policy. Then, it iterates over new orders, and in each iteration, it adds the one that increases the cost least. The following algorithm is based on Randall et al. [2022]. Specifically, it is slightly modified *parallel insertion*.

Algorithm 3 Insertion heuristic

```
 1: while $O \neq \emptyset$ do
 2:     $i^*, j^*, k^* \leftarrow 0$                    ▷ Store the best order, route and position
 3:     $x^* \leftarrow \infty$                          ▷ The smallest cost increase
 4:     for all $o_i \in O$ do
 5:         for all $R_j \in \mathcal{R} \setminus \mathcal{R}_f$ do
 6:             if $r(R_j) + r(o_i) \leq C$ then
 7:                 for all $o_k \in R_j$ where $s(o_{k-1}) \geq t$ do
 8:                     $x \leftarrow d(o_k, o_i) + d(o_i, n_{k+1}) - d(o_k, o_{k+1})$
 9:                     if $x \leq x^*$ then
10:                         $x^* \leftarrow x$
11:                         $i^*, j^*, k^* \leftarrow i, j, k$
12:                     end if
13:                 end for
14:             end if
15:         end for
16:     end for
17:     insert $o_{i^*}$ into route $R_{j^*}$ between $o_{k^*}$ and $o_{k^*+1}$
18:     $O \leftarrow O \setminus \{o_{i^*}\}$
19: end while
```

The algorithm can be improved using seeding, also proposed in Randall et al. [2022]. That means inserting m empty routes at the beginning. That prevents the algorithm from building routes in a sequential manner.

The algorithm's time complexity is $\mathcal{O}(n^3)$ for $n = |O|$ since at each round, it iterates over all orders, selects the best one, and inserts it. The inner loop, which inserts the best order, runs in $\mathcal{O}(n^2)$, and we insert at most n orders.

ϵ-greedy

The algorithm's most straightforward improvement may be using the ϵ-greedy approach. Instead of always inserting an order with the smallest cost increase, we do so with probability $1 - \epsilon$ where ϵ determines the probability that the local best option is omitted. Since the algorithm runs fast, we iterate it multiple times and select the best solution. For the first iteration, we set $\epsilon \leftarrow 0$ to guarantee that the solution is as good as the deterministic one.

Minimum-weight matching

The previous insertion algorithm is greedy since it always inserts the order with the smallest total cost increase. However, if we want to use it in a periodic manner, where we insert multiple orders at a time, this may be a drawback. We try to overcome it using minimum-weight matching.

First, we compute the cost increase for all new orders and all possible insertions, obtaining a cost tensor C. New orders can be inserted only after orders have already been served, so we do not need to consider all possible insertions.

$$C_{ijk} = \begin{cases} \text{cost increase} & \text{order } i \text{ inserted on position } k \text{ on route } j \\ \infty & \text{otherwise} \end{cases}$$

Then, we will find minimum weight matching between orders i and insertion positions (j, k) where j stands for the route index and k for the position in the route. We do so using the following MIP formulation. O_n denotes the set of new orders.

$$\min \quad \sum_{i,j,k} x_{ijk} C_{ijk} \tag{4.11}$$

$$\text{s.t.} \quad \sum_{j,k} x_{ijk} = 1 \qquad \forall i \in O_n \tag{4.12}$$

$$\sum_{i} x_{ijk} \leq 1 \qquad (\forall j \in \{1 \dots |\mathcal{R}|\})(\forall k \in \mathcal{R}_j) \tag{4.13}$$

$$\sum_{i,k} x_{ijk} r(i) \leq C - r(\mathcal{R}_j) \qquad (\forall j \in \mathcal{R}) \tag{4.14}$$

$$x_{ijk} \in \{0, 1\} \qquad (\forall i \in O_n)(\forall j \in \{1 \dots |\mathcal{R}|\})(\forall k \in \mathcal{R}_j) \tag{4.15}$$

The program has $\mathcal{O}(n^3)$ integer decision variables x_{ijk}, where $x_{ijk} = 1$ if order i is inserted on position k on route j. The 4.12 ensures that each order is inserted exactly once. The 4.13 ensures that each position is filled at most once. The 4.14 ensures that the vehicle's capacity is not exceeded, where $r(\mathcal{R}_j)$ is the total demand of route j.

Note that insertion into routes must be done carefully since insertion positions are valid for original routes. But if we insert multiple orders into a route, the insertion positions change after each insertion. We solve this by inserting orders in the order of insertion positions and increasing by one for all following positions after each insertion.

Also note, that finding the minimum-weight matching is a well-known problem, and there are many algorithms to solve it in polynomial time, which MIP does not guarantee.

4.3 Evolutionary algorithm

In this section, we describe and build an evolutionary algorithm (EA) for solving dynamic CVRP with changing vehicle availability. The algorithm is inspired by Puljić and Manger [2013], which proposes simple EA and compares different mutations and crossovers for VRP.

Encoding

Encoding is an essential aspect of any EA. We use simple permutation encoding known from EAs for TSP, where an individual is represented via a permutation of nodes. Unlike TSP, VRP considers multiple vehicles, and we also need to consider capacities. We do so by building routes sequentially, one after another, and adding a new one when the capacity of the current is exceeded. The advantage of such encoding is that it is possible to reuse known genetic operators that transform permutations into permutations.

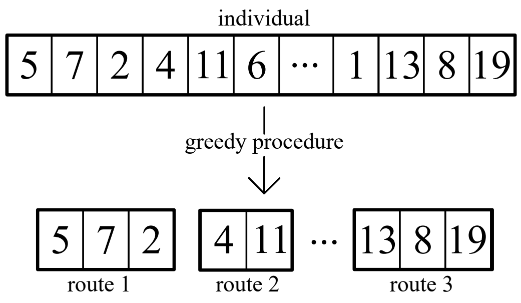


Figure 4.1: Permutation encoding

Since the problem is dynamic, each route has its past and future parts. The past part cannot be changed anymore, and the future part can. Whether an order o_i belongs to the past or future part is determined by the current time t. If the previous order o_{i-1} on the same route is served before the current time, so $s(o_{i-1}) < t$ holds, then o_i belongs to the past part. It is because the vehicle is already on the way to o_i. Otherwise, o_i belongs to the future part.

The idea is shown in the following figure. The current time is $t = 20$. Orders were or will be served at time $s(o)$. The orders in first rectangle belong to the past part, and remaing ones to the future part. Observe that the order 4 also belongs to the past part, since the vehicle is already on the way to it.

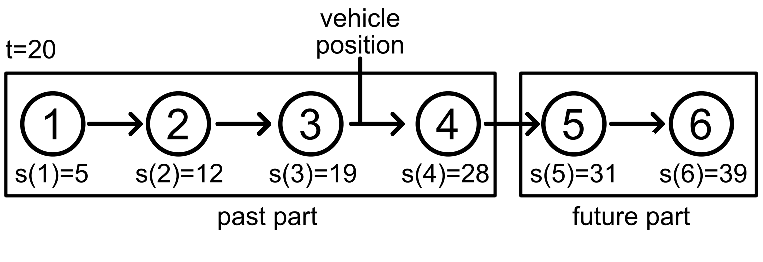


Figure 4.2: Future and past parts of a route

Only orders from the future part are considered in the chromosome. However, for fitness evaluation and the final solution, orders from the chromosome are sequentially connected to past parts of existing routes. The following procedure describes how it is done. It iterates over past parts of routes and appends parts of chromosomes to them.

The procedure has to be run for each fitness evaluation, so we must keep its computational complexity as low as possible. The following one has linear time complexity $O(n)$. The function $r(R)$ denotes the total demand of route $R \in \mathcal{R}$. The set $\mathcal{R}_s$ contains only past parts of existing routes. For simplicity, we assume that $\mathcal{R}_s$ also contains enough empty routes.

Algorithm 4 Procedure to build routes from an inidivdual

```
1: $R \leftarrow \textsc{Next}(\mathcal{R}_s)$                    ▷ Existing routes
2: for all $o \in I$ do                        ▷ For each order in the individual
3:     if $r(R) + r(o) \leq C$ then
4:         $\textsc{Append}(R, o)$
5:     else                                     ▷ Capacity exceeded
6:         $R \leftarrow \textsc{Next}(\mathcal{R}_s)$
7:     end if
8: end for
9: return $\mathcal{R}$
```

Note that orders from finished routes $\mathcal{R}_f$ are not included in the individual and are not considered while building routes or evaluating fitness. They are only appended to the final solution.

Fitness function

The fitness function is the total distance all vehicles travel, so we must minimize it. Since we do not have any soft constraints, we do not have any penalization factors.

$$\min f(I) = \sum_{R \in \mathcal{R}(I)} \sum_{k=1}^{|R|-1} d(R_k, R_{k+1})$$

Figure 4.3: Minimize the fitness function of an individual I

Selection

The next important part of any EA is a parental selection which selects individuals for crossover operators. Two well-known approaches exist, *roulette wheel selection* and *tournament selection*. Since we minimize the fitness, we use a k-tournament selection; a tournament of k individuals where the one with the best (lowest) fitness is selected with high probability. Observe that the larger the value of k, the stronger the environmental pressure on individuals.

We may also use an environmental selection. There are two common strategies called (λ, μ) and $(\lambda + \mu)$, where μ stands for the number of offspring and λ for the number of parents. In (λ, μ), we choose the μ best of λ offsprings to next generation. While in $(\lambda + \mu)$, we choose μ best of $\lambda + \mu$ offsprings and parents. As in environmental selection, we use k-tournament.

Crossovers

The crossover is a genetic operator that takes two or more individuals and produces new ones. It is inspired by the mating of individuals in nature, and the motivation is to combine high-fitness solutions into new ones, also with high fitness. Several crossovers were precisely described and tested in Puljić and Manger [2013]. We implemented two that performed the best - the ordered crossover (OX) and the heuristic version of the alternating edges crossover (AEX) called HGreX. Next, we briefly describe OX and HGreX. A more detailed description with examples is available in Puljić and Manger [2013].

The ordered crossover (OX) first randomly selects two positions within the chromosome. Those positions are the same for both parents. Then it copies part of the first parent into the first offspring and then constructs the remaining part by following node order in the second parent. Then it does so for the second offspring.

The alternating edges (AEX) crossover creates one offspring by alternately choosing arcs from the first and second parent. First, it chooses a random edge from the first parent and inserts it into the offspring. Then it chooses the next edge from the second parent, and so on. If an edge from the parent cannot be selected, it is chosen uniformly randomly from the remaining options. The HGreX crossover works similarly, but it always chooses the shorter feasible edge from both parents instead of alteration. Again, if no feasible edge exists, it selects an edge randomly.

Mutations

The mutation operator should help and support crossover to escape from local minima. It mainly drives the exploration of new solutions. We use three simple mutations mentioned in Puljić and Manger [2013].

The first one is inversion mutation (IM) which takes a part of an individual and reverses it. The second, reinsertion mutation (RM), removes a random node and reinserts it into a new random position. Last, the swap mutation (SM) randomly selects two nodes and swaps them.

Algorithm

Finally, we describe the whole evolutionary algorithm and its parameters.

- $G \in \mathbb{N}$ - generations number
- $N \in \mathbb{N}$ - population size
- $C = (c_o, c_x)$ - vector of crossover probabilities
    - $c_o \in [0, 1]$ - ordered crossover probability

    - $c_x \in [0,1]$ - HGreX crossover probability

- $M = (m_i, m_r, m_s)$ - vector of mutation probabilities
    - $m_i \in [0,1]$ - inversion mutation probability
    - $m_r \in [0,1]$ - reinsertion mutation probability
    - $m_s \in [0,1]$ - swap mutation probability

Algorithm 5 EA for dynamic CVRP

```
 1: input: $S$ solution                          ▷ solution with already served orders
 2: $\mathcal{P} \leftarrow$ InitPopulation($S$)
 3: $E \leftarrow \emptyset$                                                  ▷ elite
 4: for $g \leftarrow 1 \dots G$ do
 5:     ComputeFitness($\mathcal{P}, S$)
 6:     $E \leftarrow$ Elity($\mathcal{P}$)
 7:     $\mathcal{O} \leftarrow$ Mate($\mathcal{P}, C$)
 8:     Mutate($\mathcal{O}, M$)
 9:     $\mathcal{P} \leftarrow$ EnviromentalSelection($\mathcal{P}, \mathcal{O}$)
10:     $\mathcal{P} \leftarrow \mathcal{P} \cup E$
11: end for
12: $B \leftarrow \arg\min_{I \in E} f(I)$                           ▷ best individual
13: return $B$
```

4.4 Ant colony optimization

Last, we describe and implement an algorithm based on the ant colony optimization technique. It is based on Colorni et al. [1991] but modified for capacitated DVRP with changing vehicle availability.

Solution representation

In ACO, each ant usually represents one complete problem solution. Again, we need to deal with dynamism. Similarly to EA, we consider each route to have two parts: past and future. The ant then starts from the ends of the past parts and builds routes. The route is terminated when we run out of available nodes or vehicle capacity is exceeded. Routes are built for one ant in a parallel manner. Each time, we select an order to insert into any route concerning pheromone and attractiveness.

The idea is shown in the following figure. The solid nodes are past parts of routes except the ends. The dashed nodes are initially the ends of past parts of routes. The bolted nodes are initially the orders to be scheduled - new ones and from future parts. The algorithm always selects one edge between the dashed and bolted sets with respect to pheromone and attractiveness. Then the dashed node becomes solid and the bolted node dashed. The process is repeated until all orders are scheduled.

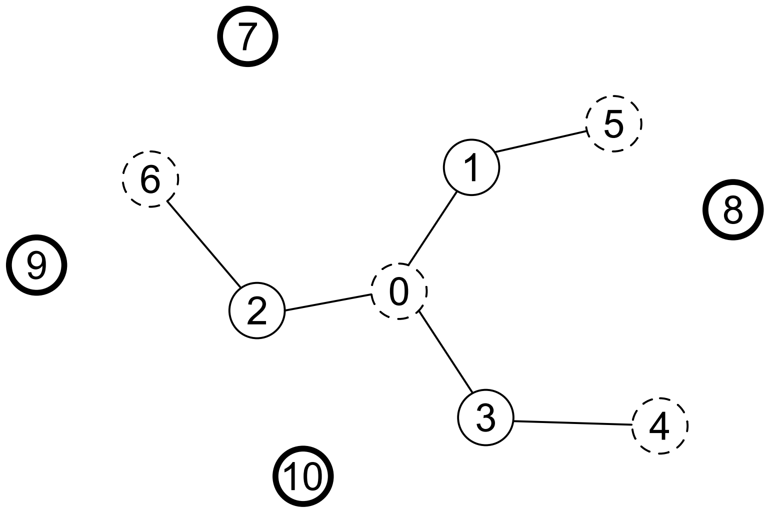


Figure 4.4: Building the solution by an ant

Pheromone and attractiveness

The attractiveness of an edge between nodes i and j will be proportional to the inverse of its distance. The closer nodes are, the more likely the ij-edge will be selected.

$$A_{ij} = \frac{1}{d(i,j)}$$

Figure 4.5: Attractiveness of an ij-edge

The pheromone is initially usually set to 1 and updates over generations. The amount of pheromone on better trails increases, and on worse ones decreases. How the pheromone is updated depends on the setup. We introduce three pheromone update modes. In the first one, all ants update pheromones at the end of a generation. In the second, only the best ant updates pheromones. Last, in the third one, each ant updates pheromones right after it finishes the trail.

The ρ is the evaporation coefficient, and Q is the pheromone update intensity coefficient.

$$P_{ij} = (1-\rho)P_{ij} + \sum_{A \in \mathcal{A}} \delta_{ij}^{A}$$

$$\delta_{ij}^{A} = \begin{cases} Q/d(S) & \text{if ant } S \text{ uses } ij\text{-edge} \\ 0 & \text{otherwise} \end{cases}$$

Figure 4.6: Pheromone update

The probability of selecting an edge ij is then computed as follows. The P stands for pheromone, A for attractiveness, and α and β are parameters that determine the importance of pheromone and attractiveness.

$$Pr(ij) = P_{ij}^{\alpha} A_{ij}^{\beta}$$

Figure 4.7: Probability of selecting ij edge

Overview

Finally, we describe the whole ACO algorithm and its parameters.

- $G \in \mathbb{N}$ - number of generations
- $N \in \mathbb{N}$ - number of ants
- α - pheromone coefficient
- β - attractiveness coefficient
- M - pheromone update mode
  - *all* - all ants update pheromone at the end of generation
  - *best* - only best ant updates pheromone at the end of the generation
  - *instant* - each ant updates the pheromone instantly
- $\rho \in [0, 1]$ - evaporation coefficient
- $Q \in \mathbb{R}$ - pheromone update intensity coefficient

Algorithm 6 ACO for dynamic CVRP

```
 1: input: $S$ solution                                   ▷ solution
 2: $X \leftarrow$ OrderToPlan($S$)           ▷ orders from future parts and new ones
 3: $Y \leftarrow$ StartingPoints($S$)                ▷ ends of past parts of routes
 4: $P \leftarrow$ InitPheromone()
 5: $D \leftarrow$ InitAttractiveness()
 6: $B \leftarrow$ null                                   ▷ best ant
 7: for $g \leftarrow 1 \dots G$ do
 8:     $\mathcal{A} \leftarrow \emptyset$                              ▷ set of ants
 9:     for $a \leftarrow 1 \dots N$ do
10:         $A \leftarrow$ PastPartsOfRoutes($S$)
11:         while $X \neq \emptyset$ do
12:             $E \leftarrow$ PossibleEdges($X, Y, A$)    ▷ possible edges w.r.t. capacity
13:             $(r, o_f, o_t) \leftarrow$ SelectEdge($E, P, D, \alpha, \beta$)      ▷ (route, from, to)
14:             $X \leftarrow X \setminus \{o_t\}$
15:             $Y \leftarrow Y \cup \{o_t\} \setminus \{o_f\}$
16:             ExtendRoute($A, r, o_t$)
17:         end while
18:         $\mathcal{A} \leftarrow \mathcal{A} \cup A$
19:         $P \leftarrow$ UpdatePheromone($P, A, M$)
20:     end for
21:     $P \leftarrow$ UpdatePheromone($P, \mathcal{A}, M$)
22:     $B \leftarrow \arg\min_{A \in \mathcal{A}} d(A)$
23: end for
24: return $B$
```

5. Experiments

The goal of the experiments is to show how the period length, changing vehicle availability and policy influence the objective - the total length of all routes. First, we tune the hyperparameters for the ACO and the Evolutionary algorithm. Then we run the algorithms on Kilby's dataset and randomly generated datasets and compare the results of individual algorithms and the influence of setups. Last, we measure the runtime of the algorithms.

The objective is to minimize the total length of all routes. If not stated otherwise, values in tables and figures are the total lengths of all routes in the solution.

5.1 Parameter tuning

In this section, we run the hyperparameter tuning experiments on the EVO, ACO and Insertion heuristic algorithms. We run them on some of Kilby's datasets, compare the results, and choose the best setups. Then, we use the setups in the following experiments.

Note that the goal of this section is to find some reasonable values of hyperparameters to obtain good-quality solutions. We do not tune each algorithm for the best possible results. If we wanted to do so, we would have to run a grid search, which would be very time-consuming.

Evolutionary algorithm

First, we tune up the parameters for the Evolutionary algorithm. We use the periodic policy with period 50. That means the algorithm accepts new orders and runs the optimization procedure every 50 ticks. We do not tune up the population size and the number of generations since we expect the more individuals and generations, the better the results. The population size is fixed to 200 individuals, and the number of generations is at most 1000. If the solution does not improve for 100 generations, the algorithm terminates.

Mutations

First, we try to find the best probabilities for mutation operators. For that, we fix the probabilities for the crossover operators to 0.3 for both types, ordered crossover and HGreX. We also assume that all mutations are equally important. We use four different setups and run them on every instance 10 times to get the average results.

- m1: InversionMut = 0.01, ReinsertionMut = 0.01, SwapMut = 0.01
- m2: InversionMut = 0.05, ReinsertionMut = 0.05, SwapMut = 0.05
- m3: InversionMut = 0.1, ReinsertionMut = 0.1, SwapMut = 0.1
- m4: InversionMut = 0.2, ReinsertionMut = 0.2, SwapMut = 0.2

In the following table, we see the results of proposed setups on some of Kilby's instances. We can see that the best results are achieved with the m3 and m4 setup, where the probabilities for all mutations are set to 0.1 and 0.2. However, the m3 setup has a smaller standard deviation, so we will use it in the following experiments since it is more stable.

				mean				std
name	m1	m2	m3	m4	m1	m2	m3	m4
instance								
c100	1533	1469	**1441**	1453	66	**42**	59	65
c50	937	936	**927**	927	27	36	**20**	29
c75	1441	**1420**	1427	1424	45	44	**30**	31
f71	416	408	404	**397**	17	21	**17**	18
tai100a	3296	3094	**3026**	3171	**142**	211	208	211
tai100b	2962	3002	3068	**2935**	125	102	**82**	135
tai75a	**2504**	2532	2506	2529	89	125	**43**	73
tai75b	2556	2550	2550	**2532**	81	26	44	**17**
All	1955	1926	**1919**	1921	966	948	**947**	953

Crossovers

Next, we search for the best probabilities of crossover operators. We fix the probabilities for the mutation operators to 0.1 for all types since we have shown that the value guarantees the best results. We use three different setups and run them on every instance 10 times.

- cx1: CXOrdered=0.7, CXHGreX=0
- cx2: CXOrdered=0. CXHGreX=0.7
- cx3: CXOrdered=0.3, CXHGreX=0.3

We can see in the following table that the best results are achieved with the cx3 setup, where the probabilities for both crossovers are set to 0.3. The CXHGreX crossover alone also performed quite well. Regardless, we will use the cx3 setup in the following experiments.

			mean			std
name	cx1	cx2	cx3	cx1	cx2	cx3
instance						
c100	1511	1449	**1412**	**55**	68	59
c50	944	**924**	939	33	32	**27**
c75	1432	**1417**	1426	39	43	**33**
f71	**401**	413	411	**12**	23	18
tai100a	3225	**3221**	3284	132	**104**	312
tai100b	3016	2983	**2963**	**111**	127	187
tai75a	2554	2522	**2476**	73	76	**63**
tai75b	2560	2546	**2544**	115	**14**	18
All	1955	1947	**1932**	969	**964**	975

Diversity

In the evolutionary algorithms, maintaining diversity is the key factor to effective exploration. We try to do so by preferring unique individuals using modified fitness f_m. Instead of the total length of all routes, we use the following formula.

$$f_m(I) = (1 - F * uniqueness(I))f(I)$$

Figure 5.1: Modified fitness of an individual

The $f(I)$ is the original fitness function - the total length of all routes. The $uniqueness(I)$ is the normalized average distance of the individual to all other individuals in the population. The F is the uniqueness factor, which we tune-up. In the modified fitness function, the individuals with the higher uniqueness have a lower fitness value, which means that they are preferred in the selection process.

We use three different setups and run them on every instance 10 times to get the average results. We also allow at most 5000 generations to see the influence of the uniqueness factor on the algorithm's performance. If the solution does not improve for 200 generations, the algorithm stops.

- uniq1: UniquenessFactor=0
- uniq2: UniquenessFactor=0.05
- uniq3: UniquenessFactor=0.15

We can see that the setup uniq2 worked the best, although the differences are not very significant. We can also see, that a high uniqueness factor can lead to worse results because good solutions are often not selected for the next generation.

			mean			std
name	uniq1	uniq2	uniq3	uniq1	uniq2	uniq3
instance						
c100	1455	**1402**	1462	40	44	**39**
c50	928	929	**918**	**19**	28	21
c75	1421	1419	**1411**	44	**29**	50
f71	397	**394**	406	**13**	20	16
tai100a	**3139**	3155	3177	228	**206**	218
tai100b	2997	**2974**	2976	**72**	162	180
tai75a	2559	**2503**	2530	71	**60**	66
tai75b	2551	2558	**2548**	44	53	**13**
All	1931	**1917**	1929	**960**	960	962

The following figure shows the average population uniqueness for dataset c100 at time 250. The uniqueness of an individual is the average number of differences in a chromosome compared to others. We can see that in setup uniq1, the population diversity is very low. That leads to the early degeneration of the population.

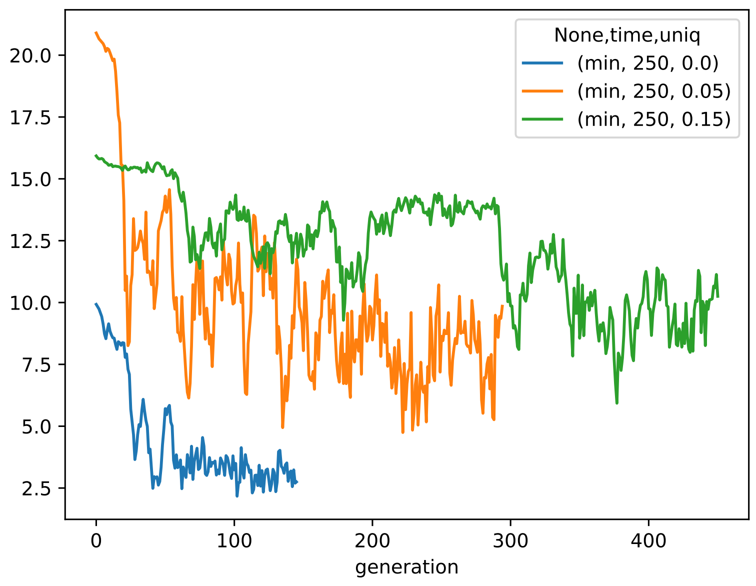


Figure 5.2: Average population uniqueness for dataset c100 at time 250

Conclusion

In previous experiments, we have shown that the best results are achieved with setups m3 and cx3, where the probabilities for mutations are set to 0.1 and for crossovers to 0.3. We also tried to maintain diversity in the population by preferring unique individuals in the selection process, which helped a little; setup uniq2 worked the best, but the differences were not very significant.

Ant colony optimization

Next, we tune up the parameters for the Ant colony optimization algorithm. We use the periodic policy with period 50, 200 ants, and at most 1000 generations. The evaporation coefficient is set to 0.2. As in the Evolution algorithm, if the solution does not improve for 100 generations, the algorithm stops. We run each setup 10 times on every instance to get the average results.

Pheromone update mode

First, we try to find the best pheromone update mode. In the ACO algorithm in 4.4, we have proposed three update modes - *all*, *best* and *instant*. In the *all* mode, the pheromone is updated for all ants, in the *best* mode, the pheromone is updated only for the best ant, and in the *instant* mode, the pheromone is updated instantly after each ant. We test three different setups.

- acoAll: PheromoneUpdateMode=All, PheromoneUpdateCoef=0.1
- acoBest: PheromoneUpdateMode=Best, PheromoneUpdateCoef=1
- acoInst: PheromoneUpdateMode=Instant, PheromoneUpdateCoef=0.1

In the following table, we see the results of the proposed setups. We can see that the best results are achieved with the acoAll setup, where the pheromone is updated by all ants with the update coefficient set to 0.1.

			mean			std
name	acoAll	acoBest	acoInst	acoAll	acoBest	acoInst
instance						
c100	**1608**	1699	1645	73	77	**62**
c50	931	943	**926**	45	**35**	50
c75	1407	1429	**1402**	49	**45**	51
f71	**380**	405	391	22	26	**17**
tai100a	2717	2716	**2708**	112	**92**	98
tai100b	**2711**	2752	2752	**63**	97	79
tai75a	**2267**	2285	2280	87	**84**	94
tai75b	**1779**	1832	1853	**61**	144	86
All	1749	1758	**1745**	788	**783**	788

Pheromone and attractiveness

Second, we try to find the best pheromone and attractiveness coefficients, usually denoted as α and β. Since we have shown that the best results are achieved with the acoAll setup, we will use it in this experiment. We will use three different setups.

- acoAttr: $\alpha = 1$, $\beta = 2$
- acoBoth: $\alpha = 1$, $\beta = 1$
- acoPhe: $\alpha = 2$, $\beta = 1$

The results are in the following table. We can see that the best results are achieved with the acoAttr setup, where the attractiveness coefficient β is set to 2 and the pheromone coefficient α is set to 1. Since the attractiveness is the inverse value of the distance, the higher exponent makes it even smaller. That means that the algorithm prefers the pheromone over the distance, which is a good sign.

			mean			std
name	acoAttr	acoBoth	acoPhe	acoAttr	acoBoth	acoPhe
instance						
c100	**1345**	1608	1545	**38**	61	53
c50	**896**	929	1011	**29**	49	70
c75	**1362**	1396	1437	**45**	97	80
f71	**363**	364	425	**10**	18	35
tai100a	**2581**	2774	2859	**37**	69	98
tai100b	**2686**	2759	2771	**26**	52	104
tai75a	**2231**	2245	2276	**80**	128	82
tai75b	1859	**1858**	1924	**92**	131	129
All	**1638**	1732	1762	**774**	805	800

Pheromone persistence

Each time the optimization procedure is run, the pheromone is initially set to 1 and updated during the run. However, the optimization procedure is run on the same problem multiple times, only with some new orders revealed. That means that the pheromone from the previous run can be used in the next run. We examine two setups - with and without pheromone persistence. In the persistence setup, the pheromone at the beginning is set to the value from the previous run. On the new edges is set to 1.

In the following table, we can see that pheromone persistence does not help at all. The best results are achieved with the noStore setup, where the pheromone is not stored between runs.

	mean		std	
name	noStore	store	noStore	store
instance				
c100	**1365**	1375	44	**32**
c50	901	**893**	29	**29**
c75	**1349**	1363	**32**	34
f71	**360**	361	**12**	14
tai100a	2612	**2594**	92	**59**
tai100b	**2687**	2696	**35**	43
tai75a	**2228**	2234	103	**99**
tai75b	**1854**	1855	82	**68**
All	**1670**	1671	777	**775**

Conclusion

In previous experiments, we have shown that the best results are achieved with the setups acoAll and acoAttr, where the pheromone is updated by all ants with the update coefficient set to 0.1 and $\alpha = 1$, $\beta = 2$. We have also shown that pheromone persistence does not help find better solutions.

Insertion heuristic

Last, we tune up the Insertion heuristic. In section 4.2, we have proposed two strategies. One inserts orders sequentially, and the other inserts orders in parallel using minimum-weight matching. In this experiment, we want to find the better one. In sequential insertion, we use the ϵ-greedy approach with $\epsilon = 0.1$ and 100 iterations. The first iteration is done with $\epsilon = 0$, so the solution is guaranteed to be as good as the deterministic one. We do not tune up these parameters since we expect that the more iterations, the better the results. We will run each setup 10 times on every instance to get the average results.

In the following table, we can clearly see that the sequential insertion works better on almost all datasets except the instance c75. The parallel insertion usually inserts equally many orders to each route, which is usually not very effective. However, in the c75 instance, the new orders appear in circles with a center in the depot, so inserting one order into each route works quite well. Regardless, we will use sequential insertion in the following experiments.

	mean		std	
name	match	seq	match	seq
instance				
c100	1947	**1695**	**0**	33
c50	1245	**994**	**0**	36
c75	**1458**	1548	**0**	32
f71	1354	**439**	**0**	4
tai100a	5567	**3105**	**0**	156
tai100b	4822	**3183**	**0**	92
tai75a	3485	**2598**	**0**	80
tai75b	**2461**	2527	**0**	126
All	2792	**2011**	1566	**945**

5.2 Kilby's dataset

In this section, we run the algorithms on Kilby's dataset from Okulewicz [2021]. We try several periods and probabilities of vehicle availability change to show the influence of these parameters on the algorithms. The objective is to minimize the total length of all routes in the solution. We use the following optimization algorithms with the best parameter setups found in the previous section:

- MIP: runtime limit 30 s
- EVO: m3, cx3, uniq2, 1000 gens, 200 individuals
- ACO: acoAll, acoAttr, noStore, 1000 gens, 200 ants
- Insertion: seq, ϵ-greedy, $\epsilon = 0.1$, 100 iterations

Period length

First, we show the influence of the period length. We use periods 1, 20, 50, 100, and 200. For period 1, which is equivalent to the online policy, only the Insertion algorithm is used. It is because the runtime of the other algorithms would be too long since the algorithm would be run for every new order. For the longer periods, all algorithms are used. We want to test the hypothesis that the longer the period, the better the results should be. It is a generalization of the observation 1, which concerns TSP only.

The table 5.1 shows the average objective for every instance, algorithm, and period. Each algorithm is run 10 times for every period and instance to obtain average results. We can see that the longer the period, the better the results are. That is in accordance with the hypothesis and the observation 1. The best results are achieved with the ACO algorithm and the MIP. This is probably because the ACO algorithm was originally designed for VRP-like problems, so it can search solution space quite effectively. The MIP, on the other hand, works with an abstract model and does not know anything about the problem. However, the power of the effective solver CP-SAT is enough to find good solutions.

																	mean
period	1				20				50				100				200
name	ins-o	aco	evo	ins-p	mip	aco	evo	ins-p	mip	aco	evo	ins-p	mip	aco	evo	ins-p	mip
instance																	
c100	1748	1462	1590	1710	1382	1377	1410	1649	1363	1200	1307	1478	1211	1205	1196	1380	**1055**
c100b	1163	1077	1223	1127	1246	948	1158	1203	1019	951	1076	1221	984	972	1171	1299	**912**
c120	1262	1366	1404	1420	1439	1448	1354	1571	1716	**1176**	1184	1653	1451	1267	1246	1449	1395
c150	2639	2146	2240	2535	1901	1983	2039	2402	1862	1830	1877	2244	1664	1827	1833	1776	**1419**
c199	3566	2568	2820	3167	2255	2456	2526	2903	2270	2343	2379	2489	2074	2424	2553	2157	**1847**
c50	1179	1006	953	1093	944	901	925	1078	928	773	832	925	844	749	694	835	**690**
c75	1878	1447	1544	1786	1442	1345	1411	1689	1324	1242	1281	1554	1261	1138	1112	1354	**1027**
f134	20756	20632	21750	20865	22235	20226	21370	20811	22647	**20163**	21299	22843	23100	21033	21069	21381	22834
f71	408	412	432	415	422	359	407	440	386	311	355	357	370	**309**	354	357	332
tai100a	3192	2802	3397	3029	2826	2591	3166	3213	2678	2511	2701	3300	2507	**2482**	2610	3251	2540
tai100b	3207	2831	3086	3294	2495	2681	2966	3227	2498	2474	2664	3359	2617	**2370**	2477	3292	2377
tai100c	2344	2174	2554	2363	1928	1983	2568	2540	1780	1962	2373	2727	1917	1767	1937	2785	**1721**
tai100d	3204	2445	2992	3026	2468	2435	3019	3077	2367	2260	2601	3013	2382	2038	2108	2794	**2007**
tai150a	5156	3942	5172	4674	4348	3959	4971	5244	4143	4119	4404	5910	4313	**3849**	3965	5960	3965
tai150b	4454	3597	4537	4568	3380	3412	3986	4327	3327	3332	3928	4787	3562	3249	3477	4636	**3244**
tai150c	4221	3303	4748	4160	3076	2934	4245	4292	2998	3008	3556	4604	**2916**	2994	3423	5157	3084
tai150d	4435	3758	4856	4890	nan	3581	4631	4596	3503	3471	4211	5310	3579	**3314**	3549	5253	3481
tai75a	2448	2311	2727	2334	2160	2224	2474	2641	2269	2017	2222	2601	2069	**1845**	1956	2417	1919
tai75b	2158	1981	2348	2238	1985	1822	2498	2119	2066	1786	2255	2040	1970	**1688**	1906	2115	1838
tai75c	2122	2001	2426	2060	2091	1892	2079	2103	1838	1730	2108	2251	1935	1686	1667	2329	**1651**
tai75d	1859	1946	2422	2356	nan	1802	2045	2093	2022	1710	1821	2183	1977	**1642**	1667	2277	1705
All	3495	3105	3593	3481	3159	2969	3393	3487	3095	2875	3163	3659	3081	**2850**	2951	3536	2907

Table 5.1: Average objectives for different periods.

Influence of the period length

The following figures show average objectives (total lengths of all routes) for different periods for each algorithm. We show only the results for tai* datasets for better clarity. For the ACO and EVO algorithms, we see that the longer the period, the better the results usually are. This is because both algorithms can exploit longer planning horizons and thus can build routes more efficiently. The same applies to the MIP algorithm, but the results are not that clear on large instances. It is probably caused by the 30s time limit, which may not be enough. On the other hand, the Insertion algorithm sometimes works better for shorter periods. It is probably because the algorithm cannot handle too many orders in one run. After all, it always chooses a locally optimal solution and does not look ahead in any way. It seems that the order of orders in the dataset actually helps the Insertion algorithm to find better solutions. Surprisingly, even the online policy with the Insertion algorithms provides the shortest total distance for some datasets like tai150d or tai75d.

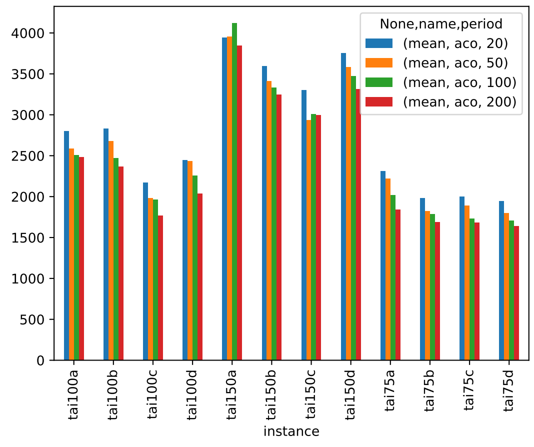


Figure 5.3: ACO

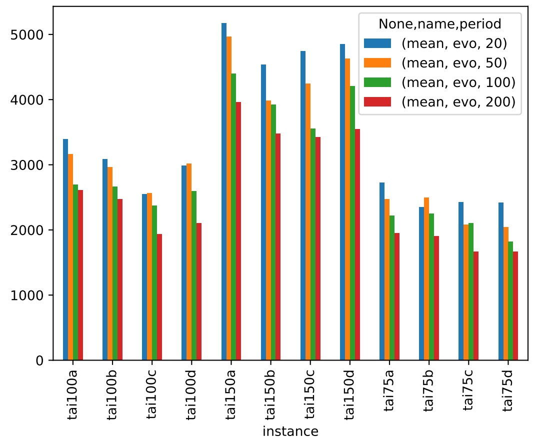


Figure 5.4: EVO

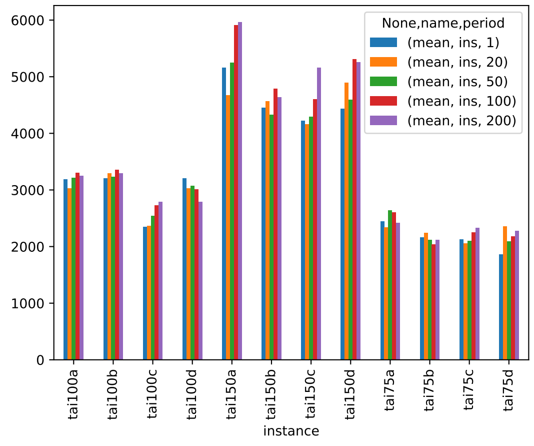


Figure 5.5: Insertion

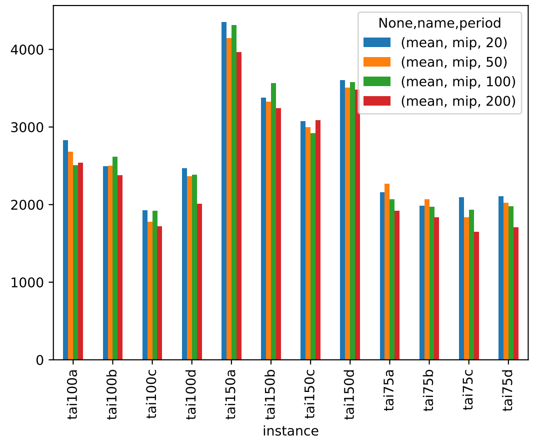


Figure 5.6: MIP

Figure 5.7: The period length's influence on each algorithm's results.

Comparison of algorithms

The following figures show the comparison of average objectives (total lengths of all routes) of algorithms for each period separately. We can see that the best results are usually achieved with the ACO algorithm and the MIP. The Insertion algorithm performs similarly well for shorter periods, especially for smaller instances. However, as the period length increases, the results of the Insertion heuristic are much worse. We have mentioned the reason before - the algorithm always chooses the locally best order to insert, so it cannot handle too many orders in one run. The EVO algorithm behaves the opposite. Its results are terrible for shorter periods, but for longer ones, they are almost as good as ACO or MIP.

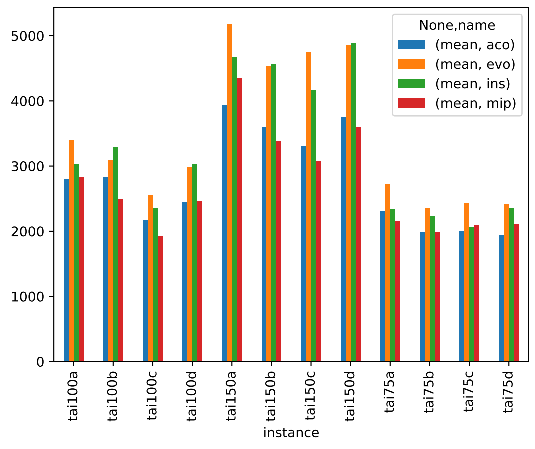


Figure 5.8: $T = 20$

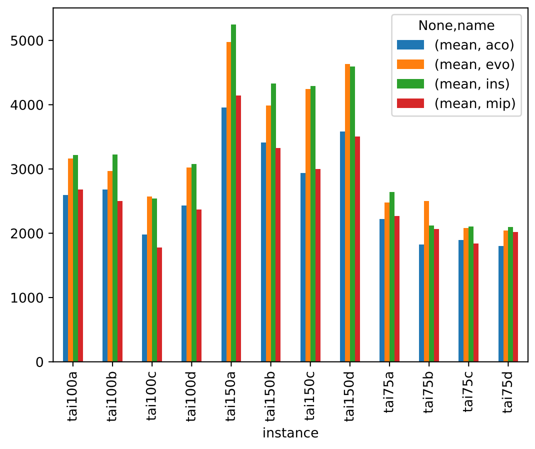


Figure 5.9: $T = 50$

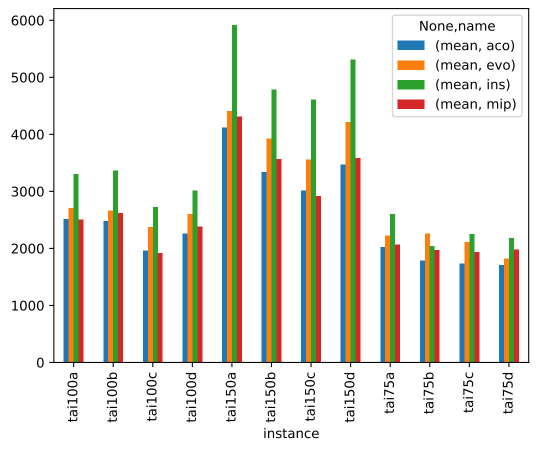


Figure 5.10: $T = 100$

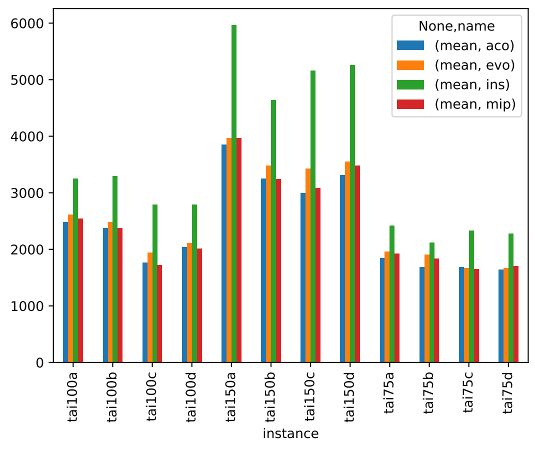


Figure 5.11: $T = 200$

Figure 5.12: Comparison of algorithms for different periods.

Another surprise is the dataset c120, where the Insertion heuristic with online policy worked really well. This is probably because the dataset c120 has a specific structure - orders are revealed in small clusters that are far from each other. Each cluster can be served by one vehicle, so even the online policy can handle it quite well.

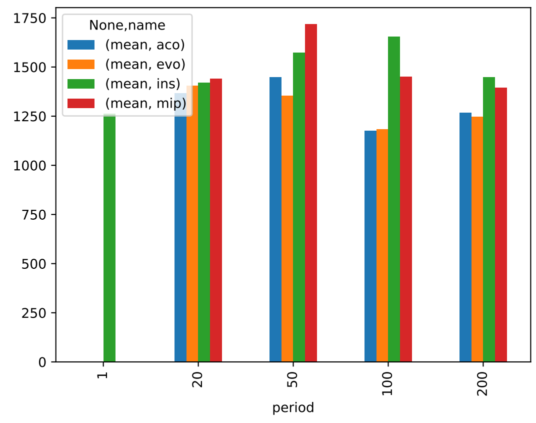


Figure 5.13: Objectives for c120

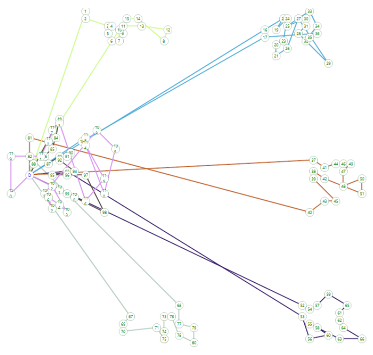


Figure 5.14: c120 structure

Conclusion

In this section, we have shown the influence of the period length. The results are in accordance with the observation 1 - the longer the period, the better the results should be for TSP. From the results, we can say that the same holds for DVRP. However, only for the ACO, EVO, and MIP algorithms. Those algorithms are able to exploit the advantage of the long planning horizon, explore many possible solutions and avoid local optima. The Insertion heuristic provides better results for shorter periods and smaller instances. It is because it is just a straightforward local search.

Changing vehicle availability

In this section, we show the influence of changing vehicle availability on the objective - total length of all routes. We use the same algorithms and setups as in the previous section with period $T = 50$. We use the following values of P - the probability that a vehicle becomes unavailable after serving an order. We expect that the higher the value of P, the worse the results will be.

- $P = 0$
- $P = 0.05$
- $P = 0.1$
- $P = 0.25$

The table 5.2 shows the objective values for each instance, algorithm, and value of P. We see that the higher the value of P, the worse the results are, which is reasonable. This is because each time a vehicle becomes unavailable, the remaining vehicles have to serve more orders, or a new vehicle has to be sent from the depot. That leads to longer total route distances in general.

																mean
P				0				0.05				0.1				0.25
name	aco	evo	ins-p	mip	aco	evo	ins-p	mip	aco	evo	ins-p	mip	aco	evo	ins-p	mip
instance																
c100	1377	1410	1649	1363	1420	1456	1621	**1357**	1443	1446	1721	1367	1569	1597	1793	1480
c100b	**948**	1158	1203	1019	1020	1202	1275	1169	1095	1304	1368	1096	1279	1406	1582	1269
c120	1448	**1354**	1571	1716	1625	1652	1899	1872	1890	1799	2289	2200	2182	2454	2662	2677
c150	1983	2039	2402	**1862**	2020	2118	2410	1884	2124	2133	2565	1931	2377	2280	2555	2013
c199	2456	2526	2903	**2270**	2575	2659	2918	2338	2678	2776	2949	2527	3008	2950	3179	2598
c50	**901**	925	1078	928	934	945	1096	982	964	965	1094	971	1015	983	1113	1090
c75	1345	1411	1689	1324	1369	1448	1696	**1253**	1395	1506	1707	1453	1510	1530	1738	1513
f134	**20226**	21370	20811	22647	22251	24517	23751	25353	24639	25172	25596	30733	33643	31318	33053	43303
f71	**359**	407	440	386	365	429	490	427	397	446	511	390	443	451	535	430
tai100a	**2591**	3166	3213	2678	2855	3307	3591	2989	3084	3490	3882	3264	3977	3980	4382	4347
tai100b	2681	2966	3227	**2498**	2772	3160	3447	3022	2920	3332	3553	2642	3778	3846	4220	3738
tai100c	1983	2568	2540	**1780**	2156	2435	2695	2093	2264	2725	2749	2074	2669	2982	3379	2681
tai100d	2435	3019	3077	**2367**	2486	2961	3084	2492	2673	3004	3349	2623	3135	3202	3615	3321
tai150a	**3959**	4971	5244	4143	4596	5291	5558	4572	4858	5513	5945	5704	6002	6301	7262	6241
tai150b	3412	3986	4327	**3327**	3770	4405	4749	4330	4237	4780	4955	4365	5652	5548	6451	5767
tai150c	**2934**	4245	4292	2998	3413	4088	4662	3329	3872	4432	5036	4442	4450	4916	5515	4799
tai150d	3581	4631	4596	**3503**	3855	4658	5288	3892	4347	4970	5294	4498	5150	5345	6336	5184
tai75a	**2224**	2474	2641	2269	2418	2619	2810	2394	2503	2665	2831	2279	2872	3041	3354	3067
tai75b	**1822**	2498	2119	2066	1951	2449	2131	2164	2056	2580	2358	2099	2288	2448	2702	2313
tai75c	1892	2079	2103	**1838**	1959	2184	2443	2013	2047	2296	2460	2243	2412	2441	2589	2467
tai75d	**1802**	2045	2093	2022	1967	2018	2240	2022	2190	2153	2414	2304	2523	2669	2531	2497
All	**2969**	3393	3487	3095	3227	3619	3796	3426	3508	3785	4030	3867	4378	4371	4653	4895

Table 5.2: Average objectives for different values of P.

Influence of changing vehicle availability

The following figures show the average objectives for each algorithm and the value of P. The results are apparent - the higher the value of P, the worse they are. Also, all algorithms seem to be affected by the changing vehicle availability very similarly.

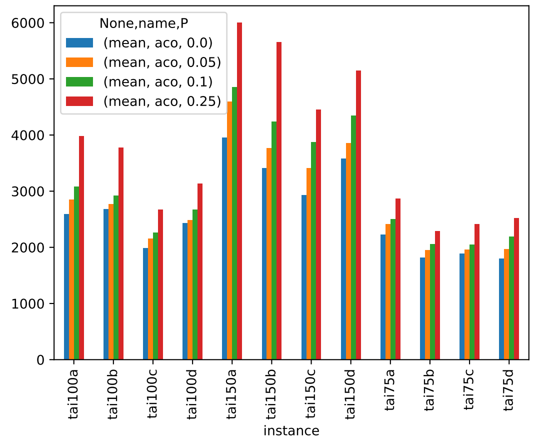


Figure 5.15: ACO

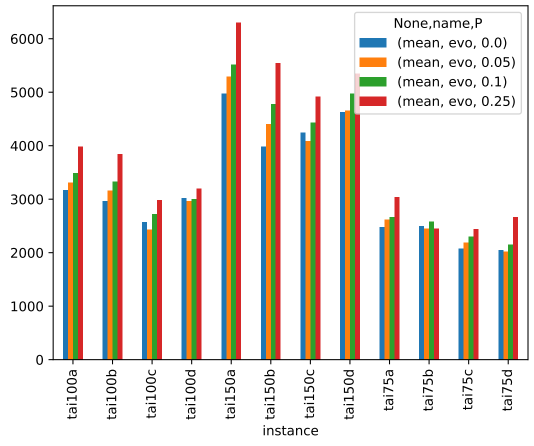


Figure 5.16: EVO

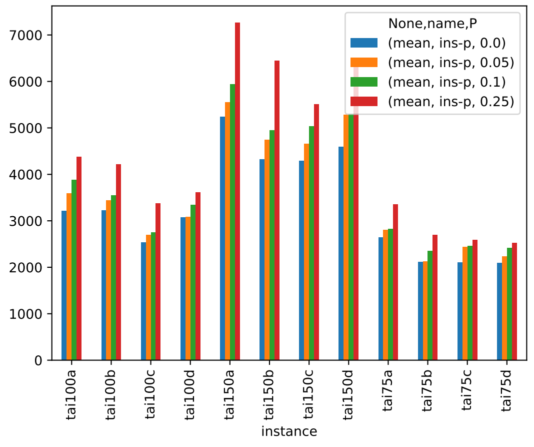


Figure 5.17: Insertion

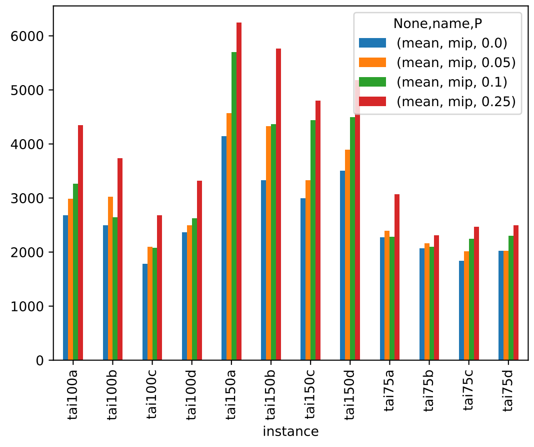


Figure 5.18: MIP

Figure 5.19: The P's influence on each algorithm's results.

Comparison of algorithms

The following figures show the comparison of average objectives (total lengths of all routes) for different values of P for each algorithm separately. We see that the best results are for every value of P achieved with the ACO algorithm and the MIP. The increasing value of P leads to worse results for all algorithms.

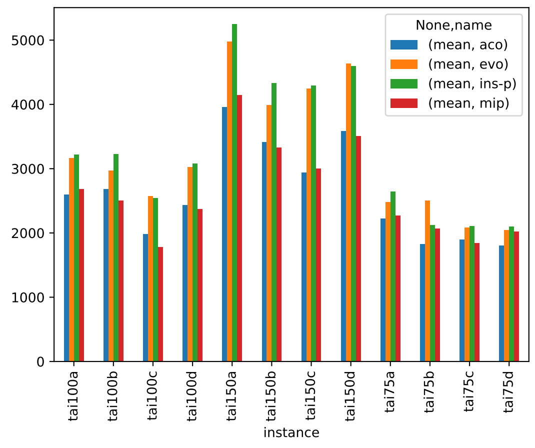


Figure 5.20: $P = 0$

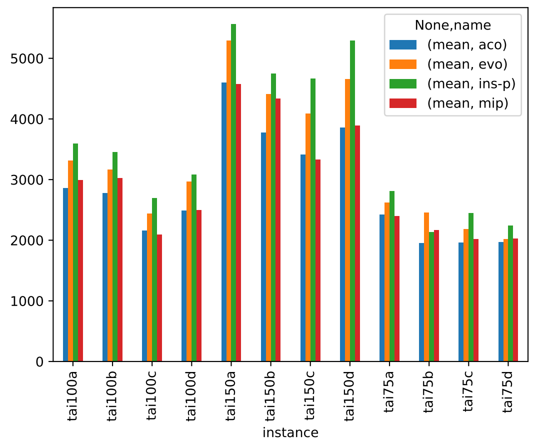


Figure 5.21: $P = 0.05$

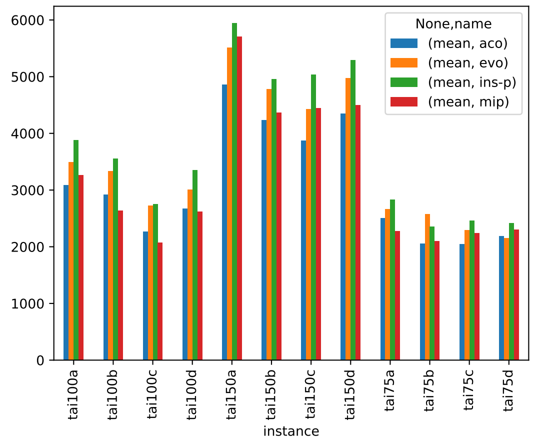


Figure 5.22: $P = 0.1$

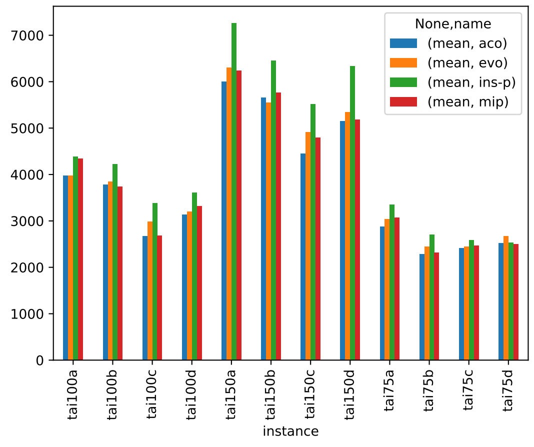


Figure 5.23: $P = 0.25$

Figure 5.24: Comparison of algorithms for different values of P.

Conclusion

In this section, we have shown the influence of changing vehicle availability for different values of P. We have shown that the higher the value of P, the worse the results are, which is reasonable. The best results are achieved with the ACO algorithm and the MIP. We did not find any significant differences between the algorithms in this case.

Policies

In this section, we examine the following policies: online, periodic, demand-based, and order-based. In the online policy, the optimization procedure is run every time a new order is revealed. Because of the runtime, we use only the Insertion algorithm with the online policy. In the periodic policy, the optimization procedure is run every $T_p = 50$ ticks. In the demand-based policy, the optimization procedure is run every time the total demand of all orders exceeds a certain threshold, specifically $C * 3/4$. In the order-based policy, the optimization procedure is run every time the number of orders is ≥ 10.

The following table shows the average objective values (total lengths of all routes) for each instance, algorithm, and policy. Most of the best results are achieved with the demand-based policy and the ACO algorithm. However, good results are also achieved with the MIP algorithm with the order-based policy.

												mean
name			aco			evo				ins		mip
policy	dem	ord	per	dem	ord	per	dem	onl	ord	per	dem	ord	per
instance													
c100	1363	1429	1377	1456	1348	1410	1588	1748	1631	1649	1313	**1187**	1363
c100b	1075	1066	**948**	1190	1388	1158	1279	1162	1167	1203	1107	952	1019
c120	**1172**	1274	1448	1253	1359	1354	1405	1262	1435	1571	1352	1698	1716
c150	2049	1856	1983	2150	2138	2039	2455	2649	**1567**	2402	1912	1908	1862
c199	2562	2502	2456	2833	3056	2526	3216	3556	**2069**	2903	2355	2398	2270
c50	917	877	901	925	887	925	1063	1179	1004	1078	933	**828**	928
c75	1381	1343	1345	1450	1462	1411	1740	1880	1687	1689	1514	1418	**1324**
f134	18228	17689	20226	20792	19480	21370	23350	20756	16874	20811	18582	**15712**	22647
f71	**349**	371	359	386	385	407	435	408	405	440	411	378	386
tai100a	**2464**	2604	2591	3065	2831	3166	3223	3268	3226	3213	2653	2773	2678
tai100b	2462	2501	2681	2767	2738	2966	3301	3233	3082	3227	**2417**	2652	2498
tai100c	1888	1988	1983	2065	2075	2568	2424	2386	2813	2540	1860	1809	**1780**
tai100d	2190	2403	2435	2513	2293	3019	2983	3342	2694	3077	2332	**1817**	2367
tai150a	**3808**	4164	3959	4566	4861	4971	5460	5156	4803	5244	4005	4186	4143
tai150b	**3271**	3756	3412	4198	5238	3986	4563	4450	4560	4327	3335	3902	3327
tai150c	3083	3395	**2934**	3879	4552	4245	4547	4213	4233	4292	3069	3497	2998
tai150d	3438	**3392**	3581	4295	4243	4631	5017	4429	4610	4596	3800	3857	3503
tai75a	2032	**1989**	2224	2170	2135	2474	2423	2448	2567	2641	2061	2224	2269
tai75b	**1640**	1686	1822	1942	2134	2498	2128	2157	2157	2119	1863	1829	2066
tai75c	**1723**	1794	1892	1894	1770	2079	2005	2138	2412	2103	1902	1740	1838
tai75d	1694	1707	1802	**1650**	1893	2045	2036	1944	2136	2093	1883	1895	2022
All	2799	2847	2969	3211	3251	3393	3650	3513	3197	3487	2889	**2793**	3095

Table 5.3: Average objectives for different policies.

Influence of policy

The following figures show the comparison of average objectives for each algorithm for different policies. The ACO algorithm works best with the demand-based policy, which runs the optimization procedure when the total demand of new orders exceeds $C * 3/4$. For other algorithms, the results are not very clear. It seems that the objective value depends heavily on the dataset. For example, in datasets c150 and c199, the best results are achieved with the Insertion algorithm and the order-based policy and all other algorithms work very badly. The order-based policy may be a good choice for the Insertion heuristic since we have shown that it works better for fewer orders at a time.

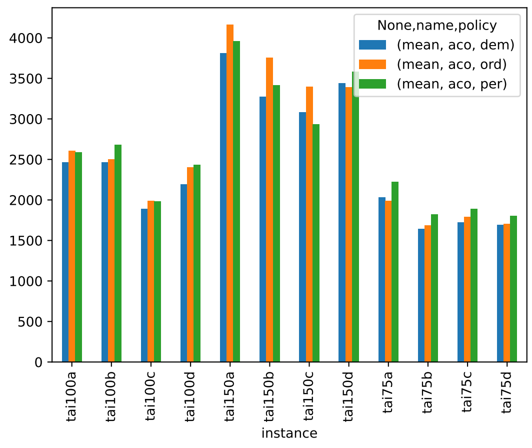


Figure 5.25: ACO

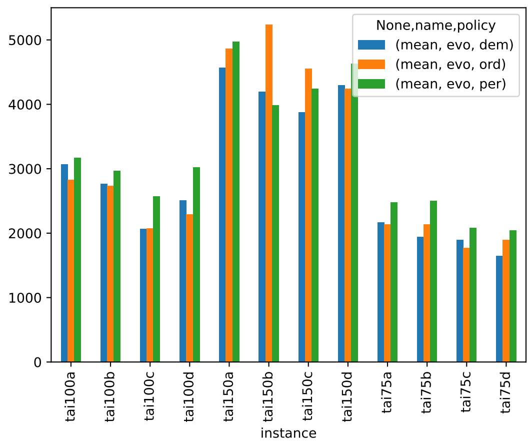


Figure 5.26: EVO

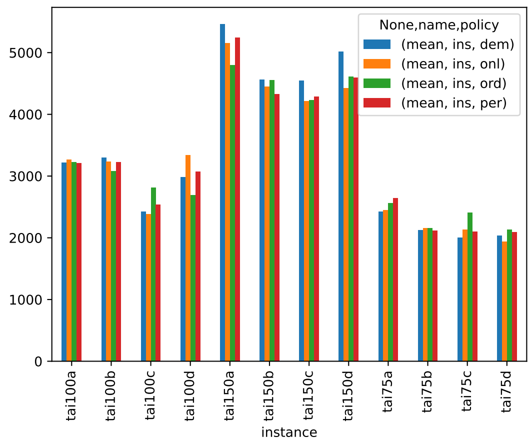


Figure 5.27: Insertion

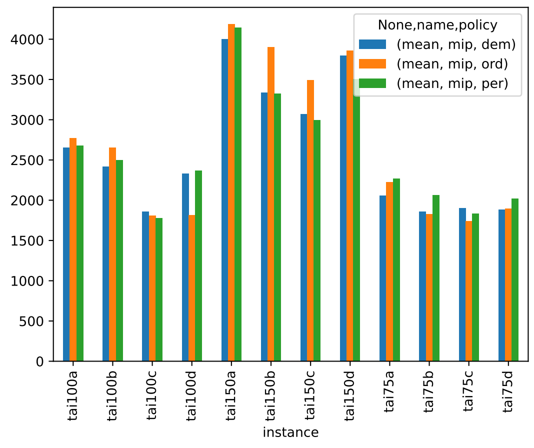


Figure 5.28: MIP

Figure 5.29: The policy's influence on each algorithm's results.

Comparison of algorithms

The following figures compare algorithms' average objectives for different policies. We see that for all policies, the best results are achieved with the ACO algorithm with the demand-based policy followed by the MIP algorithm with the order-based policy. Also, the Insertion algorithm with the order-based policy works well for some datasets.

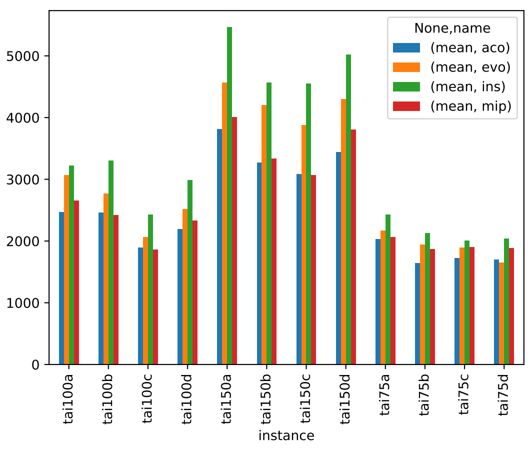


Figure 5.30: Demand-based policy

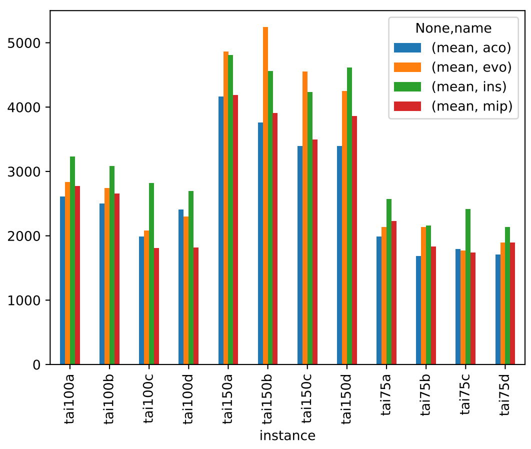


Figure 5.31: Order-based policy

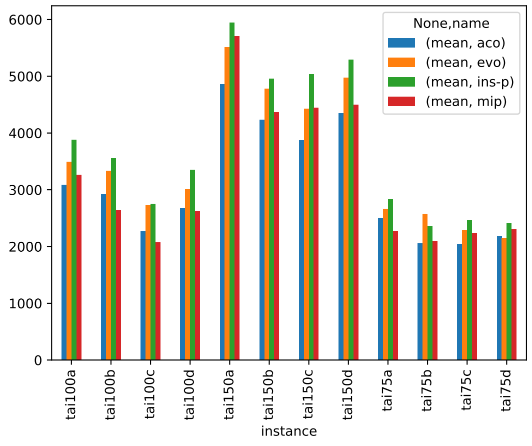


Figure 5.32: Periodic policy

Figure 5.33: Comparison of algorithms for different policies.

Conclusion

In this section, we have shown the influence of different policies on the objective - total length of all routes. It appears that different policies work best for different datasets. It does not seem that one policy outperforms the others in general.

In practice, every policy has its advantages and disadvantages. The online policy is the most computationally demanding, but it can provide an immediate response to the customer. The periodic policy is computationally less demanding, but the response time is slower, and it barely achieves the best results. The demand-based and order-based policies have even greater problems with the response time since we do not know when the threshold will be exceeded. Maybe, the combined approach with the period, order count, and demand threshold could work the best to guarantee quick response and good solutions.

Convergence of the algorithms

In this section, we compare the rate of convergence of ACO and EVO algorithms. We would like to see how fast the algorithms converge to the best solution. The period is fixed to $T = 50$ and $P = 0$. The setups for both algorithms are the same as in the previous section. We run experiments on datasets c100b and c150.

In the following figures, we can see the convergence of the algorithms at each time. On the x-axis is the number of generations, and on the y-axis is the objective value of the best solution - the total length of all routes.

We can see that in the beginning, when the number of orders is low, the EVO algorithm converges faster than the ACO algorithm and provides better solutions. However, when the number of orders increases, the ACO algorithm is able to find a better solution very fast. Although the ACO does not improve the first good solution a lot, it is usually better than any solution found by EVO. We can also see that although the maximum number of generations is set to 1000, the ACO algorithm usually finds the best solution in the first couple of hundreds. The figures show only the runs at times 100 to 300 for better clarity.

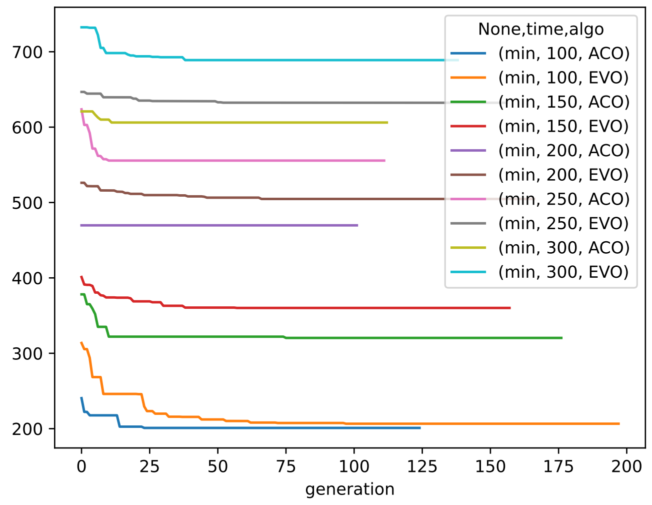


Figure 5.34: Convergence of ACO and EVO on dataset c100b.

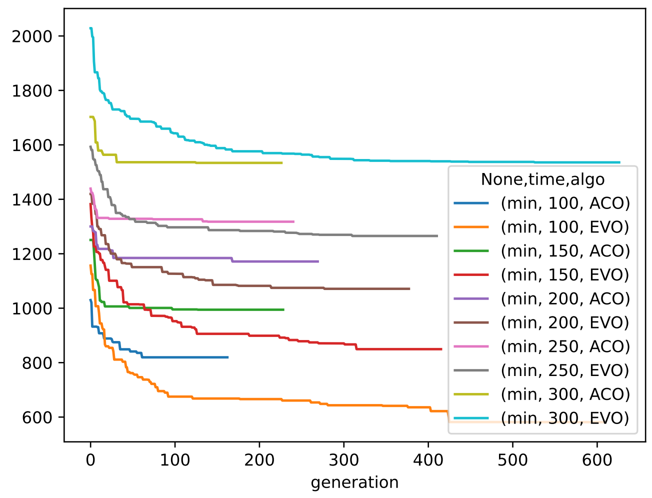


Figure 5.35: Convergence of ACO and EVO on dataset c150.

Comparison with literature

Many papers have used Kilby's dataset to test their algorithms. However, it is hard to compare with them since we use the dataset in a different manner. We assume that vehicles are already on the way during the planning process, so old orders cannot be rescheduled, which worsens the results significantly. On the other hand, we did not assume service times.

In the following table, we can see the comparison of all algorithms with period 50 and $P = 0.05$ compared to the best results in the dataset. The results are the minimal total lengths of all routes. For some datasets, we have run the MIP with unlimited runtime to see the "optimal" results. We have not done so for all datasets because the runtime, especially for larger ones, would be too long. Optimal is quoted since we have mentioned the problems with optimality in 3.5.

The first column shows the average results of all algorithms for each dataset with $T = 50$ and $P = 0$. The second column shows the best results from the Okulewicz [2021]. The third column shows the best results we found by the MIP solver with unlimited runtime.

instance	$P = 0.05$, $T = 50$	Best	MIP
c100	1502	953	1391
c100b	1165	828	-
c120	1716	1084	-
c150	2187	1164	-
c199	2733	1444	-
c50	997	570	928
c75	1508	923	1250
f134	23478	12813	-
f71	428	280	386
tai100a	3233	2178	-
tai100b	3126	2140	-
tai100c	2412	1490	-
tai100d	2849	1820	-
tai150a	5131	3273	-
tai150b	4312	2861	-
tai150c	4037	2512	-
tai150d	4551	2861	-
tai75a	2603	1778	-
tai75b	2176	1396	-
tai75c	2185	1122	-
tai75d	2064	1391	-

Table 5.4: Comparsion with literature.

Although the results are worse than the best ones, the goal of this thesis was rather to compare implemented algorithmic approaches and show the influence of period length, changing vehicle availability and policy. We did not have an ambition to try to achieve the best results.

5.3 Distribution of orders

Now, we randomly generate datasets with different location distributions of orders, and we try to find the best algorithm for each distribution. We use three different distributions - *uniform, clustered* and *quadrants.* In the *uniform* distribution, orders are distributed uniformly on the map. In the *clustered* distribution, orders are distributed in small clusters that can be served by a single vehicle. In the *quadrants* distribution, each next order is generated in the next quadrant of the map. Each distribution is examined in 3 variants numbered 1,2 and 3. Variants vary in the average number of orders per tick. Variant 1 has 0.05 orders per tick, variant 2 has 0.1 orders per tick, and variant 3 has 0.25 orders per tick. We use the same setups as in the previous section with period 50 and $P = 0$ to avoid distortion. The period length is 720 to simulate the 12-hour working day. The capacity of the vehicle C is set to 50. Each algorithm is run 10 times, and a new dataset is generated for each run.

The average lengths of all routes for different order distributions can be seen in the following table. We can see that the ACO algorithm works best for almost all distributions. We can also see that MIP works very badly for instances with a few orders, especially for the clustered1 and uniform1 distribution. On the other hand, for instances with many orders, MIP works quite well. The total distance in *clustered* distribution is usually the smallest, which is reasonable since clusters are served by a single vehicle.

					mean
name	aco	evo	ins-o	ins-p	mip
instance					
clustered1	**294**	348	401	433	718
clustered2	**498**	585	717	584	815
clustered3	**1191**	1588	1612	2056	1641
quadrants1	**702**	900	718	894	797
quadrants2	**1162**	1374	1468	1264	1307
quadrants3	2443	2466	3224	2935	**2310**
uniform1	807	761	752	**637**	1178
uniform2	1249	**1229**	1474	1349	1258
uniform3	**2072**	2429	2758	2530	2109
All	**1158**	1304	1458	1409	1348

Table 5.5: Results for different order location distributions.

5.4 Runtime

Since we handle a dynamic problem, the runtime of the algorithm may be crucial in practice. However, our experiments are just simulations Our implementation is also not optimized for runtime but for simplicity and clarity. Because of that, we did not focus on the runtime in previous experiments.

In this section, we measure the runtime of the algorithms. We use the identical setups as in the previous section with the periodic policy, period $T = 50$ and

$P = 0$. ACO and EVO are terminated after 200 generations of stagnation. The experiment was done on CPU Intel Core i7-12700K with 32GB RAM. The CP-SAT solver used for solving MIP is parallelized, so it uses all available threads. Insertion, ACO and EVO are not parallelized, so they use only one thread.

The following table shows the average runtime of the algorithms for each dataset in seconds. Note that since the period is 50, each algorithm is run multiple times to solve the whole problem. It is clear that the Insertion algorithm is the fastest one and has the best objective-runtime ratio. Surprisingly, the best average runtime is achieved with the MIP algorithm. It is probably thanks to the effective model and the parallelization of the solver. We believe that a more efficient and parallelized implementation of the ACO and EVO algorithms would lead to similar or even better results. The highlighted values are the second-best ones for each dataset. The first one is always the Insertion algorithm.

	instance	ins	mip	evo	aco
0	c100b	0.04	40.36	**10.32**	11.54
1	c100	0.00	27.26	**11.48**	20.02
2	c120	0.01	27.65	**14.71**	28.16
3	c150	0.01	**39.00**	40.16	49.99
4	c199	0.01	**41.85**	78.63	72.80
5	c50	0.00	**1.42**	2.07	5.52
6	c75	0.00	10.68	**5.17**	10.77
7	f134	0.07	**9.54**	96.66	532.06
8	f71	0.01	20.43	**4.62**	10.52
9	tai100a	0.01	22.98	**15.41**	20.52
10	tai100b	0.01	19.80	**16.53**	20.54
11	tai100c	0.01	**5.99**	18.50	28.30
12	tai100d	0.00	**7.61**	15.04	26.97
13	tai150a	0.01	67.47	**58.88**	73.65
14	tai150b	0.01	**40.00**	60.85	77.81
15	tai150c	0.01	**51.29**	52.09	78.73
16	tai150d	0.01	50.29	**41.73**	56.97
17	tai75a	0.00	14.66	**6.80**	11.79
18	tai75b	0.00	**5.78**	8.30	15.13
19	tai75c	0.00	**1.07**	9.08	12.78
20	tai75d	0.00	**6.16**	7.88	13.25

Table 5.6: Average runtime of algorithms in seconds.

5.5 Conclusion of experiments

This chapter aimed to measure the influence of different parameters and compare implemented algorithms. Our measure was the total length of all routes, which we tried to minimize.

Initially, we fine-tuned parameters for the EVO, ACO, and Insertion algorithms. We tried several setups on a subset of datasets and chose the best ones. We first found the best probabilities for genetic operators for the Evolutionary algorithm. Then, we used the modified fitness function to maintain diversity in the population. That helped to improve the results, and we have shown that the diversity of the population was preserved. For the ACO algorithm, we first found the best pheromone update mode, which was to update pheromone trails by all ants. Then, we found the best α and β parameters, determining the importance of pheromones and attractiveness. The algorithm preferred pheromone to attractiveness. Last, we tried to store the pheromone between runs, but it did not help because the problem changed a lot between runs. Last, we compared the sequential and parallel insertion heuristics. The parallel heuristic uses minimum-weight matching to insert orders into routes in a parallel manner. We found that the sequential insertion works better on most datasets.

Then, we run experiments on Kilby's dataset to show the influence of the period length, changing vehicle availability, and policy on results. First, we ran algorithms for five different periods and showed that the longer the period, the better the results are. That is in accordance with the hypothesis and the observation 1. We also compared the performance of individual algorithms. The best results on most of the datasets were achieved with the ACO algorithm and the MIP. One exception was dataset c120, where the Insertion algorithm with the online policy also worked well since the dataset has a specific map structure. Second, we showed the influence of changing vehicle availability, represented by the parameter P, which is the probability that a vehicle becomes unavailable after serving an order. As expected, the higher the value of P, the worse the results were. Again, the best results were achieved using ACO and MIP algorithms. However, for higher values of P, the Insertion algorithm with the online policy worked quite well for some datasets. Last, we examined the influence of different policies. We used the online, periodic, demand-based, and order-based policies. The best results were achieved with the demand-based policy and the ACO algorithm. However, the results were not very clear, and it seems that the best policy depends heavily on the dataset. The comparison with the literature was difficult since we used Kilby's dataset differently. However, we showed that although our approach leads to worse results than the best ones, they are close to the "optimal" ones found by the MIP solver.

Next, we tried to find out whether the distribution of orders on the map affected the results. We used three different distributions - *uniform, clustered* and *quadrants* with varying numbers of orders per tick. We showed that the results are best for the *clustered* distribution where clusters are served by a single vehicle. Furthermore, we also showed that the ACO algorithm works best for almost all distributions.

Last, we measured the runtime of individual algorithms. We showed that the Insertion algorithm is the fastest one since it is straightforward. However, all remaining algorithms also had reasonable runtimes and could be used in practice with some optimization.

The best algorithm in terms of solution quality is the ACO. It is very flexible and works well for short and long periods, with different values of P, policies, and distributions of orders on a map. The ACO is followed by the MIP, which works well for instances with many orders. Surprisingly, for high values of P and some clustered datasets, the Insertion algorithm with the online policy works quite well, and it is always the fastest one.

Conclusion

In this thesis, we studied the capacitated dynamic vehicle routing problem (CD-VRP) with changing vehicle availability. The objective was to minimize the total travel distance. The goal was to implement several algorithms for the CDVRP and compare their performance under different circumstances.

First, we went briefly through the history of the vehicle routing problem and showed some classical mathematical models and approximation algorithms. We also discussed possible constraints and objectives in the VRP. Then, we moved to the dynamic VRP and discussed the challenges and approaches to solve it.

Then, we precisely defined the version of CDVRP studied in the thesis. We considered vehicle capacities, orders revealing over time, and changing vehicle availability. We also introduced the set of constraints that must be satisfied in any feasible solution. Next, we proposed several policies on how to deal with the dynamism in the CDVRP. We considered the online and periodic policy, order-based policy, and demand-based policy.

The most important part of the thesis was the description and implementation of optimization algorithms to solve the CDVRP using some policy. We implemented a mixed-integer programming model, insertion heuristic, evolutionary algorithm, and ant colony optimization. All algorithms considered vehicle capacities and changing vehicle availability.

Last, we run several experiments. We used all algorithms with different policies and parameters and compared them. We showed that the best algorithm w.r.t. the total travel distance is the ACO algorithm. However, the results always depend on the policy, the parameters of the algorithm, and the dataset itself. We also showed that in some scenarios, like clustered orders or high probably of vehicle unavailability, the Insertion heuristic with online policy may be a good choice.

Future work

The first direction for future research is to extend the algorithms for more constraints, like time windows, multiple depots, or heterogeneous fleet. We can also consider another objective, like minimizing delivery time or maximizing the number of served orders.

The second thing may be to improve the implementation itself. Especially the evolutionary algorithm and ant colony optimization can be optimized and parallelized to achieve better runtimes.

Last, we can extend the experiments. We may try to do better parameter tuning using grid search and more possible values of hyperparameters. We can also try to generate more datasets with different characteristics and compare the algorithms on them. For example, we could study the changing vehicle availability with different periods and different policies.

Bibliography

Jose Caceres Cruz, Pol Arias, Daniel Guimarans, Daniel Riera, and Angel Juan. Rich vehicle routing problem: Survey. *ACM Computing Surveys*, 47:1–28, 12 2014. doi: 10.1145/2666003.

Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. *Operations Research Forum*, 3, 1976.

Alberto Colorni, Marco Dorigo, Vittorio Maniezzo, et al. Distributed optimization by ant colonies. In *Proceedings of the first European conference on artificial life*, volume 142, pages 134–142. Paris, France, 1991. URL `https://www.researchgate.net/publication/216300484_Distributed_Optimization_by_Ant_Colonies`.

G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4): 393–410, 1954. ISSN 00963984. URL `http://www.jstor.org/stable/166695`.

G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959. ISSN 00251909, 15265501. URL `http://www.jstor.org/stable/2627477`.

Francesco Ferrucci and Stefan Bock. Pro-active real-time routing in applications with multiple request patterns. *European Journal of Operational Research*, 253(2):356–371, 2016. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2016.02.016. URL `https://www.sciencedirect.com/science/article/pii/S0377221716300364`.

Daniela Gaul, Kathrin Klamroth, and Michael Stiglmayr. Event-based milp models for ridepooling applications. *European Journal of Operational Research*, 301(3):1048–1063, 2022. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2021.11.053. URL `https://www.sciencedirect.com/science/article/pii/S0377221721010043`.

Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986. ISSN 0305-0548. doi: https://doi.org/10.1016/0305-0548(86)90048-1. URL `https://www.sciencedirect.com/science/article/pii/0305054886900481`. Applications of Integer Programming.

Franklin Hanshar and Beatrice Ombuki-Berman. Dynamic vehicle routing using genetic algorithms. *Appl. Intell.*, 27:89–99, 06 2007. doi: 10.1007/s10489-006-0033-z.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995. doi: 10.1109/ICNN.1995.488968.

Philip Kilby, Patrick Prosser, and Paul Shaw. Dynamic vrps: A study of scenarios. *University of Strathclyde Technical Report*, 1(11), 1998.

A. Larsen. *The Dynamic Vehicle Routing Problem.* PhD thesis, Department of Mathematical Modelling, The Technical University of Denmark, Building 321, DTU, DK-2800 Kgs. Lyngby, 2000. URL `http://www2.compute.dtu.dk/pubdb/pubs/143-full.html`.

Jing-Quan Li, Pitu B. Mirchandani, and Denis Borenstein. The vehicle rescheduling problem: Model and algorithms. *Networks*, 50(3):211–229, 2007. doi: https://doi.org/10.1002/net.20199. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/net.20199`.

Yanchao Liu. An optimization-driven dynamic vehicle routing algorithm for on-demand meal delivery using drones. *Computers and Operations Research*, 111:1–20, 2019. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2019.05.024. URL `https://www.sciencedirect.com/science/article/pii/S0305054819301431`.

K. Lund, O.B.G. Madsen, J.M. Rygaard, Danmarks Tekniske Universitet. Institut for Matematisk Modellering, and Technical University of Denmark. Institute of Mathematical Modelling. *Vehicle Routing Problems with Varying Degrees of Dynamism.* IMM-REP. IMM, Institute of Mathematical Modelling, Technical University of Denmark, 1996. URL `https://books.google.cz/books?id=jwiBYgEACAAJ`.

Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7 (4):326–329, 1960.

Pedro Augusto Munari. A generalized formulation for vehicle routing problems. *ArXiv*, abs/1606.01935, 2016. URL `https://api.semanticscholar.org/CorpusID:14459897`.

Brenner Humberto Ojeda Rios, Eduardo C. Xavier, Flávio K. Miyazawa, Pedro Amorim, Eduardo Curcio, and Maria João Santos. Recent dynamic vehicle routing problems: A survey. *Computers and Industrial Engineering*, 160:107604, 2021. ISSN 0360-8352. doi: https://doi.org/10.1016/j.cie.2021.107604. URL `https://www.sciencedirect.com/science/article/pii/S0360835221005088`.

Michał Okulewicz. Dvrp kilby's instances for dynamic vehicle routing problem. *Mendely data*, 2021. doi: 10.17632/3fwc3twwn6.5. URL `https://data.mendeley.com/datasets/3fwc3twwn6/5`.

Michał Okulewicz and Jacek Mańdziuk. A metaheuristic approach to solve dynamic vehicle routing problem in continuous search space. *Swarm and Evolutionary Computation*, 48, 03 2019. doi: 10.1016/j.swevo.2019.03.008. URL `https://www.researchgate.net/publication/331870708_A_metaheuristic_approach_to_solve_Dynamic_Vehicle_Routing_Problem_in_continuous_search_space`.

Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational*

Research, 225(1):1–11, 2013. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2012.08.015. URL `https://www.sciencedirect.com/science/article/pii/S0377221712006388`.

Krunoslav Puljić and Robert Manger. Comparison of eight evolutionary crossover operators for the vehicle routing problem. *Mathematical Communications*, 18, 11 2013. URL `https://www.researchgate.net/publication/268043232_Comparison_of_eight_evolutionary_crossover_operators_for_the_vehicle_routing_problem/citations`.

Matthew Randall, Ahmed Kheiri, and Adam N Letchford. Insertion heuristics for a class of dynamic vehicle routing problems. 2022. URL `https://optimization-online.org/wp-content/uploads/2022/11/dynamic-insertion.pdf`.

M. Schyns. An ant colony system for responsive dynamic vehicle routing. *European Journal of Operational Research*, 245(3):704–718, 2015. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2015.04.009. URL `https://www.sciencedirect.com/science/article/pii/S0377221715002817`.

Zachary Steever, Mark Karwan, and Chase Murray. Dynamic courier routing for a food delivery service. *Computers and Operations Research*, 107:173–188, 2019. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2019.03.008. URL `https://www.sciencedirect.com/science/article/pii/S0305054819300681`.

Kenneth Sörensen and Fred Glover. *Encyclopedia of Operations Research and Management Science*, chapter Metaheuristics, pages 960–970. Springer, 01 2013. ISBN 978-1-4419-1137-7. doi: 10.1007/978-1-4419-1153-7_1167.

List of Figures

List of Tables

A. Attachments

A.1 Implementation

A.1.1 Specification

For running the experiments, we have implemented an application in C#. The application is divided into three projects. The first project `CDVRP` contains the data structures and implemented algorithms. The second project `CDVRPConsole` is the console interface for running the experiments. The third project `CDVRPUI` is the graphical interface for visualizing the results. The application is implemented in C# using .NET 8. Windows Forms framework is used for the graphical interface. Since .NET is a multiplatform, the console application can be run on Windows, Linux, and macOS. The graphical interface is only available on Windows or Linux with Wine.

A.1.2 Algorithms and data structures

The algorithms and data structures are in the `CDVRP` project. Algorithms implementations are stored in the `CDVRP.Algorithms` namespace. Data structures implementations are stored in the `CDVRP.DataStructures` and utility classes are stored in the `CDVRP.Helpers`.

`DataStructures.Instance`

The `Instance` class represents an instance of the CDVRP. Every algorithm requires an instance of the CDVRP in the input. It contains the orders, demands, distance matrix, vehicle capacity, availability times, and the probability of a vehicle becoming unavailable. It has `CloneAtTime(int time)` method that returns a clone of the instance with only the orders that are available at the given time. The method is used to create instances for the CDVRP over time.

The instance can be loaded from a file using the `InstanceReader` class. The class has a `ReadKilbyFile(string path)` method that reads the instance from a file in the format described in Okulewicz [2021]. The method computes the distance matrix considering the Euclidean metric and returns the `Instance`.

The instance can also be generated randomly using the `InstanceGenerator` class according to the given parameters. The class can generate datasets with various order distributions, vehicle capacities, and availability times.

The `InstanceWriter` class can save the `Instance` object to a file in the format described in Okulewicz [2021]. It is used to save generated instances for further experiments.

`DataStructures.Solution`

The `Solution` class represents a solution to the CDVRP. Every algorithm returns a solution to the problem. It contains the set of routes, the set of finished routes, where the vehicles became unavailable, and the total distance of the solution. It has several methods for computing the total distance, the route demand, and the delivery times of orders. The method `ThrowIfIncomplete()` is used to guarantee valid solutions. It also has `OrdersToPlan(int time)` method that returns orders from the solution that may be rescheduled.

`Algorithms.AlgorithmManager`

The `AlgorithmManager` class is responsible for running the algorithms in the dynamic environment. It has a `Run` method that runs the algorithm on the given instance and returns the solution. The method iterates over time and runs the optimization algorithm according to the given policy. It has parameters for the problem instance, the algorithm and its parameters, the policy settings, and output writers. It returns the final solution and a list of its snapshots stored during the run.

To be able to pass an algorithm to `Run` method, each algorithm must implement the `IAlgorithm` interface. Also, each algorithm parameters class must implement the `IAlgorithmParams` interface.

A.1.3 User interface

Console interface

The console interface is in the `CDVRPConsole` project. It is a simple console application that allows experiments to be run from the command line. It can also read and run experiment setups from a file. The application can be run on Windows, Linux, and macOS. The console interface has the following commands:

- `exit` - exits the application
- `help` - prints available commands
- `runalgo` - runs the algorithm on the instance
- `runfile` - executes commands from a file in parallel

The `runfile` command just executes all commands from a file in parallel, so the order is not guaranteed. The most important command is `runalgo`, which runs the algorithm. The command has the following parameters:

parameter	description
runalgo	the command
name	user-defined name of run
numOfRuns	number of runs
outputfile	path to output file with results
logfile	path to log file
algoFile	path to telemetry log of the algorithm
instance	path to the instance in *.vrp format
algorithm	the name of algorithm class, e.g. ACO
policy-period	AlgorithmPolicy.Policies value-Period value
probVehicleUnavailable	probability of vehicle becoming unavailable
–algorithmParamsProps	Parameters of IAlgorithmParams class of the

Table A.1: Parameters of the `runalgo` command.

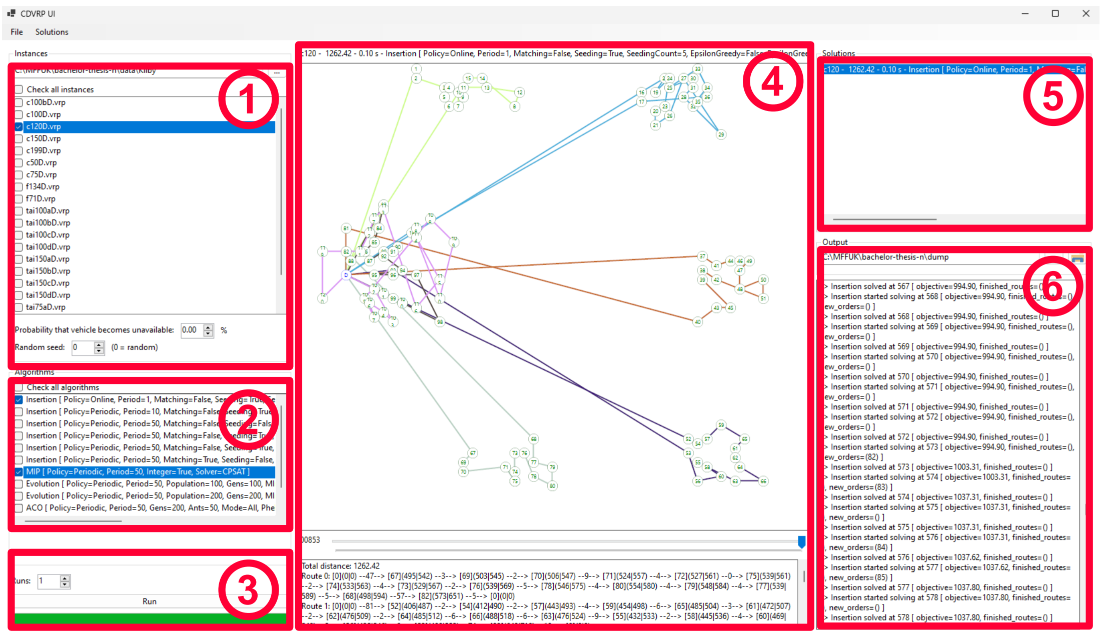
The file with the experiment can be run in the following way. Ensure, that the paths in the file are correct.

```
./CDVRPConsole.exe runfile evo_params_tuning.txt
```

Graphical interface

The graphical interface is in the `CDVRPUI` project. It is a Windows Forms application, so it can be run on Windows only. It should also run on Linux with Wine, but it was not tested.

The application has just one window that is able to run experiments and visualize the results. The application is not meant to be used by the end user. Sections of the window are visualized in the following figure and described below.



C:\MFFUK\bachelor-thesis-n\data\Kilby

Check all instances
- [ ] c100bD.vrp
- [ ] c100D.vrp
- [✓] c120D.vrp
- [ ] c150D.vrp
- [ ] c199D.vrp
- [ ] c50D.vrp
- [ ] c75D.vrp
- [ ] f134D.vrp
- [ ] f71D.vrp
- [ ] tai100aD.vrp
- [ ] tai100bD.vrp
- [ ] tai100cD.vrp
- [ ] tai100dD.vrp
- [ ] tai150aD.vrp
- [ ] tai150bD.vrp
- [ ] tai150cD.vrp
- [ ] tai150dD.vrp
- [ ] tai75aD.vrp

Probability that vehicle becomes unavailable: 0.00 %

Random seed: 0 (0 = random)

Check all algorithms
- [✓] Insertion [Policy=Online, Period=1, Matching=False, Seeding=True, Se
- [ ] Insertion [Policy=Periodic, Period=10, Matching=False, Seeding=True
- [ ] Insertion [Policy=Periodic, Period=50, Matching=False, Seeding=Fals
- [ ] Insertion [Policy=Periodic, Period=50, Matching=False, Seeding=True,
- [ ] Insertion [Policy=Periodic, Period=50, Matching=False, Seeding=True,
- [ ] Insertion [Policy=Periodic, Period=50, Matching=True, Seeding=False,
- [✓] MIP [Policy=Periodic, Period=50, Integer=True, Solver=CPSAT]
- [ ] Evolution [Policy=Periodic, Period=50, Population=100, Gens=100, MI
- [ ] Evolution [Policy=Periodic, Period=50, Population=200, Gens=200, MI
- [ ] ACO [Policy=Periodic, Period=50, Gens=200, Ants=50, Mode=All, Phe

Runs: 1

c120 - 1262.42 - 0.10 s - Insertion [Policy=Online, Period=1, Matching=False, Seeding=True, SeedingCount=5, EpsilonGreedy=False, EpsilonGree

00853

```
Total distance: 1262.42
Route 0: [0](0|0) --47--> [67](495|542) --3--> [69](503|545) --2--> [70](506|547) --9--> [71](524|557) --4--> [72](527|561) --0--> [75](539|561)
--2--> [74](533|563) --4--> [73](529|567) --2--> [76](539|569) --5--> [78](546|575) --4--> [80](554|580) --4--> [79](548|584) --4--> [77](539|
589) --5--> [68](498|594) --57--> [82](573|651) --5--> [0](0|0)
Route 1: [0](0|0) --81--> [52](406|487) --2--> [54](412|490) --2--> [57](443|493) --4--> [59](454|498) --6--> [65](485|504) --3--> [61](472|507)
--2--> [62](476|509) --2--> [64](485|512) --6--> [66](488|518) --6--> [63](476|524) --9--> [55](432|533) --2--> [58](445|536) --4--> [60](469|
```

Solutions

c120 - 1262.42 - 0.10 s - Insertion [Policy=Online, Period=1, Matching=Fal

C:\MFFUK\bachelor-thesis-n\dump

```
> Insertion solved at 567 [ objective=994.90, finished_routes=() ]
> Insertion started solving at 568 [ objective=994.90, finished_routes=(),
ew_orders=() ]
> Insertion solved at 568 [ objective=994.90, finished_routes=() ]
> Insertion started solving at 569 [ objective=994.90, finished_routes=(),
ew_orders=() ]
> Insertion solved at 569 [ objective=994.90, finished_routes=() ]
> Insertion started solving at 570 [ objective=994.90, finished_routes=(),
ew_orders=() ]
> Insertion solved at 570 [ objective=994.90, finished_routes=() ]
> Insertion started solving at 571 [ objective=994.90, finished_routes=(),
ew_orders=() ]
> Insertion solved at 571 [ objective=994.90, finished_routes=() ]
> Insertion started solving at 572 [ objective=994.90, finished_routes=(),
ew_orders=() ]
> Insertion solved at 572 [ objective=994.90, finished_routes=() ]
> Insertion started solving at 573 [ objective=994.90, finished_routes=(),
ew_orders=(82) ]
> Insertion solved at 573 [ objective=1003.31, finished_routes=() ]
> Insertion started solving at 574 [ objective=1003.31, finished_routes=
), new_orders=(83) ]
> Insertion solved at 574 [ objective=1037.31, finished_routes=() ]
> Insertion started solving at 575 [ objective=1037.31, finished_routes=
), new_orders=() ]
> Insertion solved at 575 [ objective=1037.31, finished_routes=() ]
> Insertion started solving at 576 [ objective=1037.31, finished_routes=
), new_orders=(84) ]
> Insertion solved at 576 [ objective=1037.62, finished_routes=() ]
> Insertion started solving at 577 [ objective=1037.62, finished_routes=
), new_orders=(85) ]
> Insertion solved at 577 [ objective=1037.80, finished_routes=() ]
> Insertion started solving at 578 [ objective=1037.80, finished_routes=
), new_orders=() ]
> Insertion solved at 578 [ objective=1037.80, finished_routes=() ]
```

Figure A.1: Graphical interface of the CDVRP application.

In section 1, the user selects a directory with instances in `*.vrp` format. The application then reads all instances using `InstanceReader` from the directory and displays them in the list. The user can select an instance from the list to run experiments on.

In section 2, the user selects an algorithm to run on the instance. More algorithms and setups can be added in source code only in method `LoadAlgorithms` in `FormMain.cs` class.

In section 3, the user selects the number of runs and can run experiments using the `Run` button. All experiments are run in parallel.

Section 4 displays the selected solution from section 5. The solution is visualized on the map with the routes of the vehicles. The depot is colored in blue, the served orders are colored in green, and the unserved orders are colored in red. Each route is colored differently. Below the map is a time slider, so the user can see the solution at different times. At the bottom is a text representation of the solution.

In section 6 users select the output directory. The application saves the results of the experiments in the selected directory in `*.csv` files for further processing. Below is `AlgorithmManager` log.

A.2 attachments.zip file

The attachments to this thesis are available in the attachments.zip file. The file contains the following directories:

- Experiment results - raw data from the experiments
- Experiment setups - configuration files for the experiments
- Implementation - source code of the implemented algorithms, graphical and console interface.
- Kilby - Kilby's instances used in the experiments