Solving Large Scale Arc Routing Problems

Diogo Miguel Ferreira de Oliveira

Thesis to obtain the Master of Science Degree in

Mechanical Engineering

Supervisors: Prof. João Miguel da Costa Sousa

Examination Committee

Chairperson: Prof. Carlos Baptista Cardeira
Supervisor: Prof. João Miguel da Costa Sousa
Member of the Committee: Prof. José Rui de Matos Figueira

December 2021

Resumo

O *Capacitated Arc Routing Problem* (CARP) é um problema de otimização combinatória muito importante com uma vasta gama de aplicações tais como recolha de resíduos, manutenção de estradas e espalhamento de sal para prevenir neve. Nesta tese, propomos um Algoritmo Memético com Mutação por Divisão e Conquista (MADCoM) para resolver o *Mixed Capacitated Arc Routing Problem* (MCARP), uma variante do CARP mais adequada às redes de estradas do mundo real. É dada ênfase à resolução de instâncias de grande escala, uma vez que a maioria das aplicações abrange uma cidade inteira. MADCoM é um algoritmo genético com controlo de diversidade adaptativo hibridizado com procura local eficiente. Introduzimos um novo operador de mutação que consiste em aplicar a uma solução duas heurísticas de dividir para conquistar, *Route Cutting Off Decomposition* e *Hierarchical Decomposition*. Demonstramos que conduz a um melhor desempenho do algoritmo quando combinado com procura local, principalmente para instâncias maiores. Além disso, *Hierarchical Decomposition* é utilizado como um método de inicialização. Mostramos que gera uma população diversificada com soluções de qualidade. Definimos um dos seus parâmetros como dependente da dimensão do problema, o que leva a uma redução do tempo computacional sem afectar a qualidade das soluções. Testamos o desempenho do MADCoM nas referências clássicas para CARP e MCARP, bem como nas referências mais recentes de grande dimensão. Encontramos novas melhores soluções para 8 instâncias de MCARP, 2 das quais são soluções ótimas, e novas melhores soluções para 12 instâncias de grande dimensão do CARP.

Palavras-chave: Capacitated Arc Routing Problem, algoritmo genético, procura local, dividir para conquistar

Abstract

The Capacitated Arc Routing Problem (CARP) is a very important combinatorial optimization problem with a wide range of applications such as waste collection, road maintenance and winter gritting. In this thesis, we propose an algorithm called Memetic Algorithm with Divide-and-Conquer Mutation (MADCoM) to solve the Mixed Capacitated Arc Routing Problem (MCARP), a variant of CARP more suited to real-world street networks. An emphasis is placed on solving large-scale instances, as most applications span an entire city. MADCoM is a genetic algorithm with adaptive diversity control hybridized with efficient local search. We introduce a novel mutation operator which consists of applying to a solution two state-of-the-art divide-and-conquer heuristics, Route Cutting Off Decomposition and Hierarchical Decomposition. We demonstrate that it leads to improved performance of the algorithm when coupled with local search, particularly for larger instances. Furthermore, Hierarchical Decomposition is used as an initialization method. We show that it generates a diverse population with quality solutions. We define one of its parameters to be dependent on the problem size, which leads to reduced computational time without affecting solution quality. We test the performance of MADCoM on the classical benchmarks for CARP and MCARP, as well as the more recent large-scale benchmarks. We find new best solutions for 8 instances of MCARP, 2 of which are optimal solutions, and new best solutions for 12 large-scale CARP instances.

Keywords: capacitated arc routing problem, genetic algorithm, local search, divide-and-conquer

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Nowadays, over 55% of the world's population lives in cities. By 2050, the urban population is expected to double, with two-thirds of the world living in urban areas [1]. To accommodate the growing population, cities will expand both vertically and outwards. Total urban area is projected to nearly triple until 2030, an increase of 1.2 million km^2, equivalent to an area the size of South Africa [2].

Critical to a city's expansion is the development of its infrastructure, to ensure continued economic growth and quality of life for its citizens. As more housing and industry is built, the city's road network will expand and services such as waste collection and mail delivery will have an increasingly complex task of providing for the entire city.

The latest advancements in technology have made it possible to collect enormous amounts of data. The city of the future will be a smart city, where interconnectivity and sensor data will transform operations, with the end goal of improving the quality of the service provided and reducing costs. In waste collection, for example, sensors installed in garbage bins allow for real-time monitoring of filling levels that can be used to optimize collection routes and reduce carbon emissions [3].

Although access to real-time data will bring many benefits, it will also limit the computational time budget available, as operations will become increasingly dynamic. Routing algorithms must become more efficient at finding good solutions, and with ever-expanding cities route planning becomes more complex, meaning an effort must be made to tackle large-scale problems.

1.2 Objectives and Contributions

The objective of this thesis is to develop a competitive algorithm to solve the Mixed Capacitated Arc Routing Problem (MCARP). The MCARP is an extension of the Capacitated Arc Routing Problem (CARP) that better models the street network of a city. An emphasis is placed on solving large-scale problems, as the CARP and MCARP are used to model applications that span an entire city, such as waste collection [4, 5], road maintenance [6], and winter gritting [7].

We propose a new algorithm to solve large-scale MCARP called Memetic Algorithm with Divide-and-Conquer Mutation (MADCoM). MADCoM combines two state-of-the-art methods: a genetic algorithm with adaptive diversity control hybridized with efficient local search, and divide-and-conquer heuristics tailored for large-scale CARP. The heuristics are expanded to MCARP and used in a novel divide-and-conquer mutation operator. We show that it leads to improved performance of the algorithm when coupled with local search. We also use one of the heuristics to initialize the population, and analyse its effect on the quality and diversity of the population. We define one of its parameters to be dependent on the problem size, which leads to reduced computational time and does not affect solution quality significantly.

We test MADCoM on the classical benchmarks for CARP and MCARP, as well as the more recent large-scale benchmarks. We find new best solutions for 8 instances of MCARP, 2 of which are optimal solutions, as they match the best lower bound in the literature. We also find new best solutions for 12 large-scale CARP instances.

1.3 Thesis Outline

In this section, we give an outline of the remaining contents of this thesis, as a way to summarize the contents of each chapter. In chapter 2, we start by describing the Mixed Capacitated Arc Routing Problem and provide a mathematical formulation. Several variants of CARP and their applications are also presented to demonstrates its applicability to real world problems. The chapter ends with a review of solution methods used to solve MCARP.

In chapter 3, we describe in detail each component of MADCoM. The description is supplemented with figures, schematics and examples to fully elucidate the concepts and algorithms.

Chapter 4 contains the results of MADCoM on the CARP and MCARP benchmarks, as well as an analysis of the benefits of the large-scale heuristics for the algorithm.

Lastly, in Chapter 5 we review the obtained results and provide the next steps for further improving this work.

Chapter 2

Capacitated Arc Routing Problems

2.1 Introduction

The Capacitated Arc Routing Problem (CARP) is a combinatorial optimization problem that was first proposed by Golden and Wong in 1981 [8]. It is defined on an undirected weighted graph where each edge can be traversed in two directions. A subset of the edges have a demand that is required to be serviced by a vehicle. Each vehicle has a maximum capacity and the sum of the demands in the vehicles' route can not exceed it. The objective is to minimize the total distance travelled by all vehicles, while servicing every required edge and respecting the vehicle's capacity constraints.

To exemplify, Figure 2.1 shows the problem graph of instance C16 from the benchmark set *bmcv*. The thicker connections between nodes represent the edges that need to serviced by a vehicle. The thinner connections do not need to be serviced, but can still be traversed by a vehicle. The squared yellow node represents the depot, a special node from where all vehicles must start and end their routes. In Figure 2.2, the optimal solution of instance C16 is shown. Each color illustrates a route performed by a vehicle. The direction of each arrow shows in which direction the edge is traversed by the vehicle. Filled arrows denote the edges serviced by the vehicle and the dashed and curved arrows represent the deadheading links, that is, edges that are only traversed and where no service is performed by that vehicle.

The CARP is NP-hard, that is, one can not prove that a solution to the problem is optimal in polynomial computational time. As a consequence, the objective shifts to finding quality solutions in a reasonable amount of time. However, this becomes increasingly difficult with larger problems, as the solution space increases exponentially with the number of required edges.

The CARP is similar to the Capacitated Vehicle Routing Problem (CVRP), their main difference lies in where the demand is placed. In the CVRP the demand is associated to nodes of the problem graph and in the CARP it is associated to edges. As such, the CARP is used to model applications where the edge represents a street, the problem graph represents the street network of a city and the demand encompasses a limited quantity that must be collected, delivered or otherwise serviced. Some applications include waste collection [4, 5], street mapping [9], road maintenance [6], meter reading [10] and winter gritting [7].

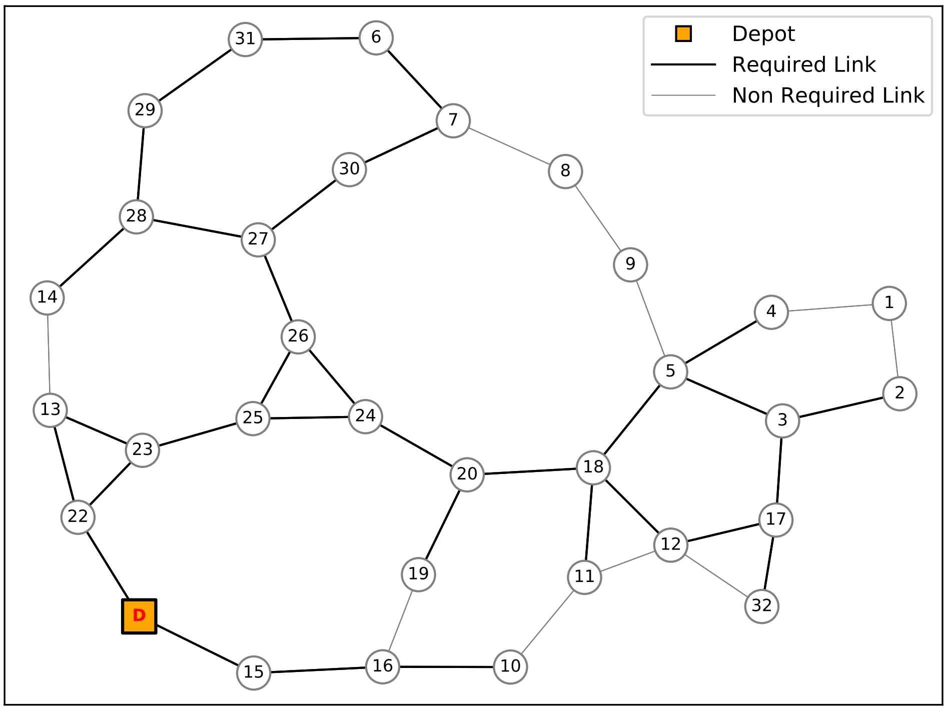


Figure 2.1: Problem graph of instance C16 from benchmark set *bmcv*.

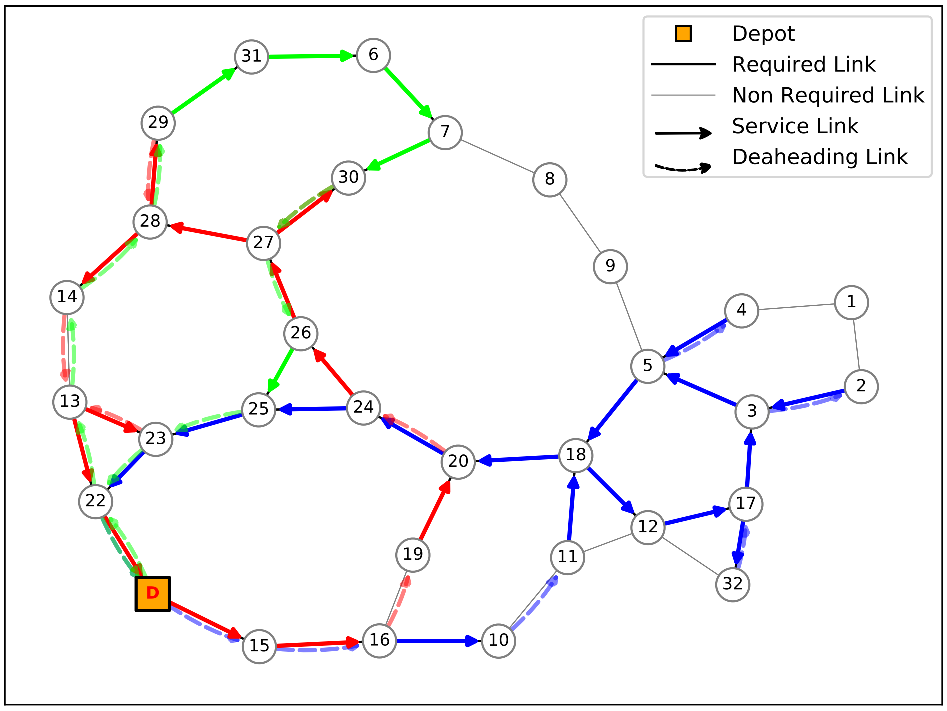


Figure 2.2: Optimal solution of instance C16.

Most of these applications do not use the CARP to model the problem being solved, but instead a variant of it that introduces additional constraints or changes to the formulation. For example, in the classical CARP, an edge can be traversed by a vehicle in both directions, however, in real world street networks, most streets can only be traversed in one direction. For this reason, Belenguer et. al [11] proposed the Mixed Capacitated Arc Routing Problem (MCARP). The MCARP is defined on a mixed graph, that is, a graph with both edges and arcs. Edges can be traversed by a vehicle in both directions, while arcs only allow one direction. In the remainder of this thesis, we focus on solving this variant of CARP, with an emphasis on problems with a large number of links. A mathematical formulation is given in section 2.2.

2.2 Mathematical Formulation

The presented formulation is based on the MCARP formulation given by Constantino et. al in [12].

The Mixed Capacitated Arc Routing Problem (MCARP) is defined on a mixed network (N, A_D, A_R, E_R) which includes deadheading links, with no demand, and required links, with demand. N is the set of nodes and all vehicles start and end their routes at a special node called the depot, which is given the index 0. All deadheading links are represented by arcs in the set A_D. The required links, also called services or tasks, are represented by arcs and edges in the sets A_R and E_R respectively. Each link has a traversal cost $d_{ij} > 0$. For required links there is also a demand $q_{ij} > 0$ and a service cost $s_{ij} >= d_{ij}$.

The initial network is transformed into a directed graph $G = (N, A)$. The set of all arcs A is given by $A = A_D \cup R$, where the set of required arcs R is defined as $R = A_R \cup A_{E_R}$. The set A_{E_R}, defined as $A_{E_R} = \{(i, j), (j, i) : \{i, j\} \in E_R \wedge i < j\}$, is created by replacing each required edge by two opposite arcs, one for each service direction, with the same costs and demand as the original edge. It is only required to service one of these arcs, as they both represent the edge in the original network.

The fleet is composed of P identical vehicles, each with capacity Q. Each vehicle used incurs a fixed cost F. For most benchmarks, there is no limit on the number of vehicles, which is equivalent to setting $P = |E_R| + |A_R|$, as one vehicle per required link is the maximum possible number of vehicles in a solution.

A solution to MCARP consists of a set of routes that services each required link, uses a number of vehicles smaller than the fleet size, and each route respects the capacity constraint. The optimal solution minimizes the objective function, which is composed of the fixed cost of each vehicle, the cost of servicing each required link and the cost of deadheading the links between services, that is, traversing the links without servicing. To complete the mathematical formulation, the following variables are also needed:

- x_{ij}^p is a binary variable that equals 1 if the required arc (i, j) is serviced by vehicle route p and 0 otherwise.

- y_{ij}^p is the number of times an arc (i, j) is deadheaded by route p.

- f_{ij}^p is the flow in arc (i, j), related with the remaining demand in route p.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{p=1}^{P}\left(\sum_{(i,j)\in A} d_{ij}y_{ij}^{p} + \sum_{(i,j)\in R} s_{ij}x_{ij}^{p} + F\sum_{(0,j)\in A} y_{0j}^{p}\right) && && \text{(2.1a)}\\
\text{subject to} \quad & \sum_{p=1}^{P} x_{ij}^{p} = 1, && \forall (i,j)\in A_R, && \text{(2.1b)}\\
& \sum_{p=1}^{P}(x_{ij}^{p} + x_{ji}^{p}) = 1, && \forall (i,j)\in A_{E_R}, && \text{(2.1c)}\\
& \sum_{j:(i,j)\in A} y_{ij}^{p} + \sum_{j:(i,j)\in R} x_{ij}^{p} = \sum_{j:(j,i)\in A} y_{ji}^{p} + \sum_{j:(j,i)\in R} x_{ji}^{p}, && \forall i\in N, \forall p, && \text{(2.1d)}\\
& \sum_{(0,j)\in A} y_{0j}^{p} \le 1, && \forall p, && \text{(2.1e)}\\
& \sum_{j:(j,i)\in A} f_{ji}^{p} - \sum_{j:(i,j)\in A} f_{ij}^{p} = \sum_{j:(j,i)\in R} q_{ij}x_{ji}^{p}, && \forall i\in N\setminus\{0\}, \forall p, && \text{(2.1f)}\\
& \sum_{(0,j)\in A} f_{0j}^{p} = \sum_{j:(j,i)\in R} q_{ij}x_{ji}^{p}, && \forall p, && \text{(2.1g)}\\
& f_{ij}^{p} \le W(x_{ij}^{p} + y_{ij}^{p}) && \forall (i,j)\in R, \forall p, && \text{(2.1h)}\\
& f_{ij}^{p} \le W(y_{ij}^{p}) && \forall (i,j)\in A_D, \forall p, && \text{(2.1i)}\\
& x_{ij}^{p} \in \{0,1\} && \forall (i,j)\in R, \forall p, && \text{(2.1j)}\\
& y_{ij}^{p} \ge 0 \text{ and integer} && \forall (i,j)\in A, \forall p, && \text{(2.1k)}\\
& f_{ij}^{p} \ge 0 && \forall (i,j)\in A, \forall p && \text{(2.1l)}
\end{aligned}
$$

The first term in the objective function (2.1a) is the cost of deadheading the arcs, the second term is the cost of servicing the required arcs, and the third term is the fixed cost of the vehicles, where the term $\sum_{p=1}^{P}\sum_{(0,j)\in A} y_{0j}^{p}$ is the number of vehicles used. Equations (2.1b)–(2.1c) ensure that every required link is serviced by only one vehicle. Equation (2.1d) imposes the connectivity of routes at each node and equation (2.1e) is needed to adequately charge the fixed cost in the objective function. (2.1f)–(2.1g) are the flow conservation constraints and (2.1h)–(2.1i) are the linking constraints, which together guarantee the connectivity of the routes. Equations (2.1h)–(2.1i) are the capacity constraints. (2.1j)–(2.1l) are the domain constraints.

2.3 CARP Variants

In most real world situations, the CARP is not sufficient to model all the characteristics of a problem. New constraints must be introduced which increase the complexity of the problem, but in turn provide more adequate routing solutions. The new constraints originate new variants of the Capacitated Arc Routing Problem and are detailed below.

Heterogeneous Fleets

In the classical CARP it is assumed that the fleet is homogeneous, that is, every vehicle has the same capacity. In reality, fleets are rarely homogeneous which increases the complexity of the routing problem. With an heterogeneous fleet constraint, several vehicle types exist, each with a different number of available vehicles, different capacities. The cost of a route is calculated based on a fixed cost and a variable cost per distance travelled that are dependent on the vehicle type. This variant has been extensively studied for the CVRP and most techniques are easily adaptable to CARP. A literature review of heterogeneous techniques for the CVRP is available at [13].

Multiple Depots

The depot node usually represents a garage, from where vehicles start and end their routes. However, in many applications several garages may exist and vehicles can start their routes from more than one location. The Multi Depot Capacitated Arc Routing Problem (MD-CARP) considers multiple depots, and the most common formulation is that a vehicle must start and end its route on the same depot. An example is available at [14].

Periodic Routing

In periodic routing, a time horizon is considered and a set of routes must be found for each time period. Each required edge has a demand generation and typically has to be serviced more than once within the time horizon to ensure all demand is fulfilled. This situation arises often in waste collection, where the collection is performed several times per week on a fixed schedule. The objective is then to minimize the cost function over the entire time horizon. A mathematical formulation and several algorithms are given by Chu et al. in [15].

Time Windows

The Capacitated Arc Routing Problem with Time Windows (CARPTW) introduces the additional constraint that each edge must be serviced within a time interval. If the time window is hard, then the service cannot occur outside of it; if it is soft then a penalty is incurred when not respecting the time interval. In winter gritting [7], some streets must be serviced within a few hours originating hard time windows, and in street mapping [9], taking pictures in the direction of the sun can lead to unusable photographs, which can be modelled as a soft time window.

Stochastic Demands

In some applications, such as waste collection, the exact demand on a street might not be known a priori. To model this, the Stochastic CARP (SCARP) was proposed by Fleury et. al [16], where the demand on each edge is a random variable. A routing solution is not only evaluated in terms of expected cost, but also in its robustness to variations in the demand.

2.4 Existing Approaches

2.4.1 Constructive Heuristics

Constructive heuristics were among the first algorithms proposed to solve the CARP. An heuristic is an algorithm that is not guaranteed to find the optimal solution, but nevertheless finds a feasible solution in a short amount of time.

Two of the first heuristics developed were Augment-Merge [8], by Golden and Wong in 1981, and Path-Scanning [17], by Golden, DeArmon and Baker in 1983. Augment-Merge is inspired by the Clarke and Wright Heuristic [18] for the CVRP. The first phase, called Augment, starts by building a route for each required edge and then joins two routes together if the required edge of the shortest route appears in the longest route. After the Augment phase, follows the Merge phase, where two routes are merged if it is beneficial. Path-Scanning builds routes one-by-one using a simple rule to decide which required edge to add to the end of the route. Five different rules are used and the best solution among them is returned. Both of these heuristics have been improved throughout the years and a comprehensive review of heuristics for the CARP can be found in [19].

In 1985, Ulusoy [20] proposed a route-first, cluster-second method for the CARP. Instead of building routes one-by-one, this method first builds a giant tour, a route with infinite capacity that services all required edges. This giant tour is formed using a method to solve the Chinese Postman Problem (CPP), the arc routing equivalent of the Travelling Salesman Problem (TSP), where the objective is to find a route with minimal cost that traverses all edges of the problem graph. To transform the giant tour into routes that do not violate the capacity constraint, the Split procedure is applied. Split takes as input a sequence of required links and using an auxiliary Directed Acyclic Graph (DAG) divides it into routes in an optimal way. The Split procedure has since become an essential building of block of several metaheuristics and will be detailed in section 3.3.1.

More recently, Wøhlk proposed FastCARP [5], a heuristic specifically designed for large-scale CARP. It starts by building a giant tour, partitions the graph into districts and builds routes for each district. From there, adjacent districts are merged, the routes of the new district are optimized and the district is then split again. This process continues iteratively until a time limit is reached.

A constructive heuristic's biggest advantage is speed, thus for applications where the computational budget is very limited, they can be the best method. However, if the computational budget is reasonable, exact methods and metaheuristics can outperform constructive heuristics.

2.4.2 Exact Methods

Exact methods start by defining a Mixed Integer Programming (MIP) mathematical formulation and applying algorithms like branch-and-bound to solve it. As opposed to heuristics and metaheuristics, given enough time exact methods are guaranteed to find the optimal solution. Some approaches use a mathematical formulation designed for the CARP, while others transform CARP into a CVRP. Several transformations exist [21–23], but all of them increase the problem size, that is, a CARP with n required edges is transformed into a CVRP with at least $2n + 1$ nodes. The advantage of this transformation is that techniques developed for the CVRP can be readily applied to solve the CARP. The recently most successful algorithms are variants of branch-and-bound that implement cutting-plane and column-generation methods.

Cutting-plane methods iteratively add cuts, i.e. linear inequalities, to the the mathematical formulation to refine the set of feasible solutions. When combined with branch-and-bound the method is called branch-and-cut. This method was used by Baldacci and Maniezzo [22], where they transformed the CARP into a CVRP, and Belenguer and Benavent [24] who applied it to a one-index formulation for the CARP.

Column-generation methods exploit the idea that in large MIP problems, most variables in the optimal solution will be zero and as a result only a subset of the variables will actually improve the objective function. When combined with branch-and-bound this method is called branch-and-price. After transforming to a CVRP, Longo, Poggi de Aragão and Uchoa [23] combined both cutting-planes and column-generation into a branch-and-price-and-cut to solve the CARP. Bode and Irnich [25] also combined both methods into a cut first branch-and-price second algorithm that leverages the sparsity of real world street networks.

Exact methods have also been applied to solve several CARP variants [26]. In particular, Gouveia, Mourão and Pinto [27] use flow variables to derive a compact formulation for the MCARP, which is detailed in section 2.2.

The main drawback of exact methods is scalability: when the problem size increases, so does the number of variables and constraints leading to very large MIP problems, that are too time consuming to solve for most practical applications.

2.4.3 Metaheuristics

Metaheuristics are ”solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space” [28]. Metaheuristics do not guarantee optimality, however, due do their architecture, they are capable of finding high quality solutions in a short amount of time.

The first metaheuristics applied to CARP were a Simulated Annealing (SA) algorithm by Eglese [7] in 1994 and a Tabu Search (TS) algorithm by Eglese and Li [29] in 1996. In 2000, Hertz, Laporte and Mittaz proposed CARPET [30], a TS algorithm that encodes routes as a list of all edges traversed.

Since then, these and several other metaheuristics were applied to solve the CARP and its variants. In 2008, Polacek et al. designed a simple and efficient Variable Neighborhood Search (VNS) [31] and Brandão and Eglese [32] a very effective TS, similar to CARPET, but fully deterministic. Wøhlk [33] proposed a SA algorithm called DYPSA that represents solutions as a giant tour and applies a Split procedure with Flips, that is, a Split procedure that also optimally decides the best orientation to service each required edge. Beullens et al. [34] designed a Guided Local Search (GLS) that represents the network as a symmetric directed graph where each edge is replaced by two arcs, one for each service orientation, and routes are encoded as a list of required arcs. In 2010, Santos et al. published an Ant Colony Optimization (ACO) algorithm [35] where each ant builds a giant tour, that is transformed into a full solution using a Split procedure, and afterwards undergoes local search.

Another metaheuristic extensively applied to CARP are Memetic Algorithms (MA). Memetic algorithms are hybrid metaheuristics that combine genetic algorithms with local search. Genetic algorithms are a population based heuristic inspired by genetics, reproduction and natural selection. Solutions are encoded as chromosomes and new solutions are formed by combining the chromosomes of other solutions. Local search methods search for improvements to a solution in its neighborhood, where the neighborhood is defined by a set of moves that alter a small part of the solution.

The first MA proposed for the CARP was by Lacomme, Prins and Ramdane-Chérif [36]. The chromosome representation is a giant tour, which is then decoded using Split. In 2009, Tang, Mei and Tao introduced a Memetic Algorithm with Extended Neighborhood Search (MAENS) [37] that uses Split as a large neighborhood move. The chromosomes are obtained by concatenating the required arcs of each route, separated by a copy of the depot. In 2014, Vidal et al. published Unified Hybrid Genetic Search (UHGS), a MA with advanced diversity management that can solve a variety of CVRP variants. In 2017, Vidal extended UHGS to CARP and introduced constant time local search move evaluations, that is, independent of problem size, as was the case with previous works. UHGS also uses giant tours as a chromosome representation and both the Split procedure used and the local search moves decide optimally the service direction. UHGS is currently the best performing algorithm on the classical instances.

With several metaheuristics capable of efficiently solving the classical instances, the challenge remains for solving large-scale instances with thousands of required edges. The latest algorithms solve large-scale CARP by using a divide-and-conquer approach. In 2014, Mei, Li and Yao [38] published RDG-MAENS, a Cooperative Coevolution framework that iteratively uses the best-so-far solution to decompose the problem into smaller subproblems that are solved independently using MAENS. The best solutions for each subproblem are then joined together to form a solution to the main problem. Tang et al. [39] proposed a scalable approach based on Hierarchical Decomposition (HD). They defined virtual tasks as a permutation of required edges and cluster them based on a distance measure. The virtual tasks that belong to the same cluster are joined into a new virtual task in an order decided by a greedy heuristic. The process repeats until a single virtual task is left, representing a giant tour. The authors embedded HD into a individual-based search method called SAHiD, that iteratively decomposes the routes of the current solution to form a virtual task set for HD, thereby generating a new solution which is then improved by local search. Zhang and Mei improved both methods with a new decomposition scheme, Route Cutting Off (RCO) decomposition [40] that uses a task rank matrix to find good and poor links to decompose the routes of a solution. RCO-SAHiD is currently the metaheuristic with the best results on large scale instances, alongside UHGS.

Chapter 3

Solving Large-Scale MCARP

3.1 Introduction

To solve Large-Scale MCARP we propose a Memetic Algorithm with Divide-and-Conquer Mutation (MADCoM). MADCoM combines the adaptive diversity control and efficient local search of UHGS with the divide-and-conquer heuristics tailored for large-scale CARP of RCO-SAHiD.

The population of MADCoM is composed of individuals that each represent a solution to the MCARP instance being solved. It is divided into two subpopulations: a feasible subpopulation containing feasible solutions, and an infeasible population containing infeasible solutions used to guide the search towards better solutions. At each iteration, a new individual is generated by mutation with probability P_M, which consists of applying RCO and HD to a randomly selected individual in the population, or by combining two parent individuals using Order Crossover (OX). The Split procedure is applied to the new individual to obtain the cost and routes of its solution. Then, the new individual undergoes local search with probability p_{LS} and is inserted into a subpopulation depending on its feasibility. If it is infeasible, it can undergo a repair procedure with probability p_R, that attempts to transform it into a feasible solution. If a subpopulation reaches its maximum size, survivor selection is triggered and individuals are discarded until the subpopulation is at its minimum size, resulting in a new generation. At the end of the iteration, the penalty for infeasible solutions and the local search and repair probabilities may be adjusted. The algorithm stops after reaching a time limit or It_{NI} iterations without improvement. The outline of the algorithm is shown in Algorithm 1.

The remainder of this chapter will focus on the description of each element of MADCoM. In section 3.2 we detail the search space of CARP and the techniques used to reduce it. Section 3.3 focuses on the particulars of the genetic algorithm and section 3.4 on the local search. Finally, section 3.5 details the large-scale heuristics used when generating an individual by mutation, namely Route Cutting Off Decomposition and Hierarchical Decomposition.

Algorithm 1: MADCoM

```
Initialize sub-populations
while time $< T_{max}$ and number of iterations without improvement $< It_{NI}$ do
    Generate a random number $r \in \{0, 1\}$
    if $r < p_M$ then
        Select an individual based on cost and diversity
        Apply RCO and HD to create an offspring
    else
        Select parents using binary tournament based on cost and diversity
        Create an offspring using Order Crossover (OX)
    Apply Split to obtain the routes and cost
    Generate a random number $s \in [0, 1]$
    if $s < p_{LS}$ then Improve the new individual using Local Search
    if feasible then
        Add to feasible sub-population
    else
        Add to infeasible sub-population
        Generate a random number $t \in [0, 1]$
        if $t < p_R$ then Repair the infeasible individual
    Adjust penalty parameter $\omega$
    Adjust $p_{LS}$ and $p_R$
    if maximum sub-population size reached then
        Select individuals for the next generation based on cost and diversity
    if no improvement for $It_{div}$ iterations then Diversify population
```

3.2 Search Space

An explicit solution to MCARP is a sequence of nodes for each route that represents the path in the problem graph taken by each vehicle, as well as which links are serviced by each route. This representation was used by some of the first metaheuristics [30], but recent algorithms use instead an implicit representation by decomposing the search space of MCARP into four decision subsets [41]:

- ASSIGNMENT - Assign each services to a route
- SEQUENCING - Order the services in each route
- MODE CHOICE - Choose the service direction of each required link
- PATHS - Find the shortest paths between successive services

Each of these decision subsets leads to an exponential number of solutions. The larger the solution space the more difficult it is for a search algorithm to find the optimal solution (or a near-optimal solution). However, we only need to define the ASSIGNMENT and SEQUENCING decision subsets to represent a solution, because if they are known, the PATHS and MODE CHOICE can be derived via dynamic programming algorithms. Although there is an added computational cost, the reduction in the solution space will result in the algorithm finding better solutions in a shorter amount of time.

The PATHS decision subset can abstracted by using a distance matrix between arcs instead of the original distance matrix between nodes of the problem graph. Each edge $(i, j) \in E_R$ is replaced by two arcs (i, j) and (j, i), one for each mode, with the same cost as the original edge, deadheading arcs are no longer considered and the depot is represented by an arc that starts and ends at the depot node. Each arc is given an id k and the depot arc is represented by the id 0. The distance $d(k,l)$ between two arcs $k=(a,b)$ and $l=(c,d)$ is defined as the distance of the shortest path between nodes b and c. The shortest paths between nodes can be precomputed, using an all pairs shortest path algorithm such as the Floyd-Warshall algorithm [42], and used to build $d(k,l)$, avoiding the need to compute when calculating the cost of a solution.

The Mode Choice can be derived in an optimal way from the Sequencing decision subset by finding a shortest path in an auxiliary directed acyclic graph $\mathcal{K}$ [43]. Each service is given an id i, and has a set M_i containing all modes associated with it. $|M_i|=2$ if i is a required edge and $|M_i|=1$ if i is a required arc. The depot is represented by the id 0 and has only one mode $M_0=(0)$. Let a route σ be defined as a sequence of services $\sigma=(0,\sigma(2),...,\sigma(|\sigma|-1),0)$, starting and ending at the depot. The auxiliary DAG $\mathcal{K}$ (Figure 3.1) can be constructed in the following way: a node is added for each mode $k\in M_{\sigma(i)}$ of each service i in the route, including the depot; then, an arc is added from each mode k of service $\sigma(i)$ to each mode l of the next service in the sequence $\sigma(i+1)$ with cost equal to the distance $d(k,l)$ between those modes. The shortest path from depot to depot gives the optimal Mode Choice of each service of the route as well as its deadheading cost.

Figure 3.1 shows an example of the auxiliary DAG $\mathcal{K}$ used to find the optimal mode choice for each service. The number next to each arc represents the distance between the modes that it connects, and the number next to each node represents the cost of the shortest path up to that node. The thicker arcs show the shortest path with total cost 47, and the nodes that are part of it are the optimal mode choice for each service in the route.

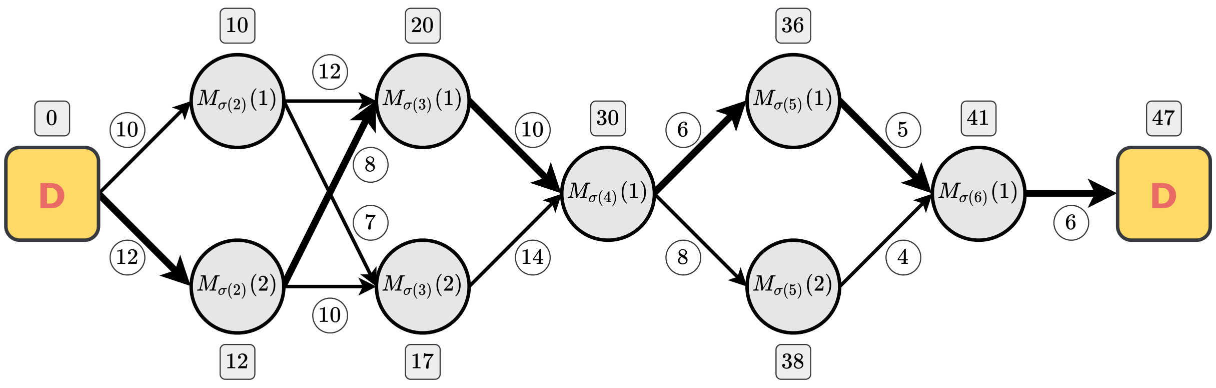


Figure 3.1: Example of the auxiliary DAG $\mathcal{K}$ used to compute the optimal mode choices for a route σ.

The total cost of a route is composed of three elements, the distances between the mode choices for each service, the service costs and the fixed cost F of the vehicle. Let $\rho=(0,\rho(1),\dots,\rho(|\rho|-1),0)$ be the set of mode choices for each service $\sigma(i)$ of route σ. Then, the route cost $C(\sigma)$ is expressed by equation 3.1.

$$C(\sigma)=F+\sum_{i=1}^{|\sigma|}s_i+\sum_{i=1}^{|\sigma|-1}d(\rho(i),\rho(i+1)) \tag{3.1}$$

From the set of possible routes, only a subset will be part of a feasible solution to MCARP, those whose total demand is smaller or equal to the vehicle capacity Q. The remaining routes are infeasible, but instead of being discarded, they are used to guide the search towards better feasible solutions. The optimal solution to an optimization problem often lies close to the boundary of feasibility, as those are the solutions that use the resources available more efficiently. Infeasible routes are penalized using equation 3.2, where $Q(\sigma)$ is the sum of the demands of the required links serviced by route σ and ω is the penalty parameter. Algorithm 2 details the calculation of the penalized cost $C_P(\sigma)$ of route σ, with optimal mode choices. The auxiliary DAG $\mathcal{K}$ is not generated explicitly and during execution only the cost is needed, so the mode choice for each service is left implicit.

$$C_P(\sigma) = C(\sigma) + \omega \max\{0, Q(\sigma) - Q\} \tag{3.2}$$

Algorithm 2: Route Cost

```
Data: Route $\sigma$, arc distance matrix $d(k,l)$, demands $q$, service costs $s$, vehicle capacity $Q$, fixed
      cost $F$, penalty $\omega$
Result: $routeCost$
$load \leftarrow 0$
$serviceCosts \leftarrow 0$
$PrevLabels \leftarrow [(0,0)]$                                        /* (cost,mode) */
for $i \in \sigma$ do
    $load \leftarrow load + q_i$
    $serviceCosts \leftarrow serviceCosts + s_i$
    $NewLabels \leftarrow [\,]$
    for $l \in M_i$ do
        $newCost \leftarrow \infty$
        for $prevCost, k \in PrevLabels$ do
            if $prevCost + d(k,l) < newCost$ then
                $newCost \leftarrow prevCost + d(k,l)$
        Append $(newCost, l)$ to $NewLabels$
    $PrevLabels \leftarrow NewLabels$
$routeCost \leftarrow PrevLabels[0][0] + serviceCosts + F + \omega \max(0, load - Q)$
```

The ASSIGNMENT and SEQUENCING decision subsets constitute the solution space that MADCoM must search. In the genetic algorithm, the ASSIGNMENT decision subset is left implicit as each solution is represented by a giant tour, a permutation of the service ids, reducing the search space to $n!$, where $n = |E_R| + |A_R|$. The Split procedure is then applied to obtain the cost and routes of the solution. During local search, the ASSIGNMENT and SEQUENCING are explicit, as this representation allows for faster move evaluations in $O(1)$.

3.3 Genetic Algorithm

3.3.1 Chromosome Encoding and Decoding

In genetic algorithms, a solution to an optimization problem is encoded as a chromosome. The chromosome representations of two individuals are combined to create a new individual that shares genetic information from both parents. As such, the chromosome representation is one of the most important parts of a genetic algorithm. As discussed in section 3.2, it also constitutes the solution space that the algorithm must explore to find the optimal solution.

MADCoM uses a compact representation known as a giant tour, a route with infinite capacity that services all required links. The giant tour is essentially a permutation of the indices assigned to each required link. To obtain a solution and its cost, the chromosome must be decoded using the Split procedure.

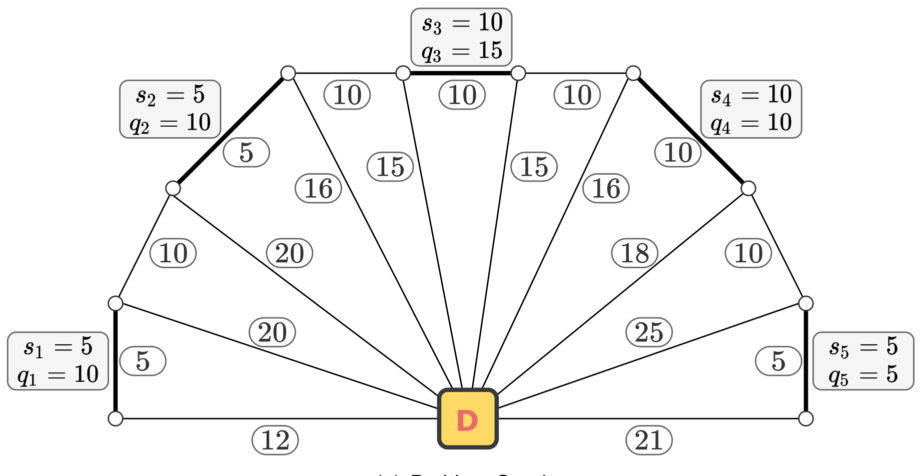
Given a giant tour as input, Split segments the permutation into routes that respect the capacity constraint in an optimal way, that is, from all possible ways to divide the permutation while keeping the order, Split finds the division that originates the solution with the lowest cost. This property assures that the chromosome representation is complete, as any solution that could be represented by the permutation either has a higher cost and therefore is not the optimal solution to the problem, or is the solution returned by Split.

Let δ_i be the i^{th} service in giant tour δ. Split defines an auxiliary DAG $\mathcal{H}$ with $n+1$ nodes, indexed from 0 to n. An arc from node i to node $j > i$ represents a route starting from the depot, fulfilling the demand in services δ_{i+1} to δ_j and returning to the depot. The cost of this arc is equal to the penalized cost of the route. The arc only exists if the capacity of the route does not exceed the maximum capacity $Q_{max} = 1.5Q$, where any solution that contains a route exceeding the vehicle capacity Q is deemed infeasible and is penalized. The shortest path in graph $\mathcal{H}$ from node 0 to node n gives the optimal segmentation of δ into routes.

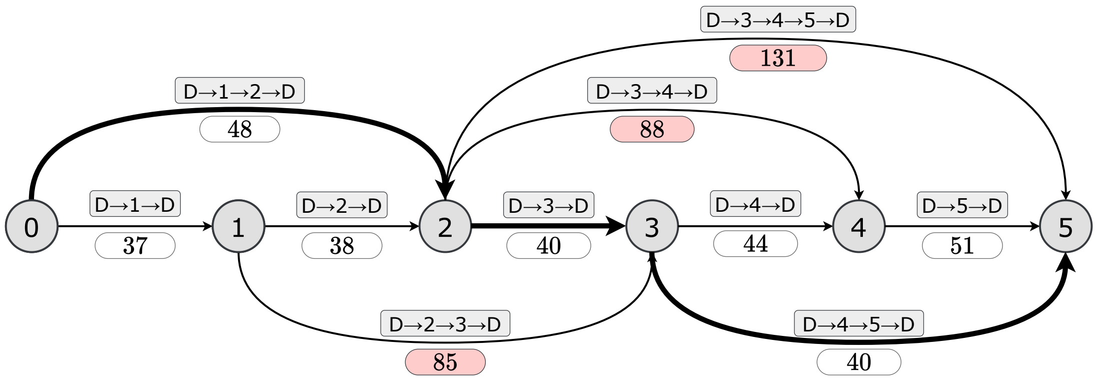
Figure 3.2 shows an example of Split. The problem graph is shown in figure 3.2(a), with a total of 5 services and the deadheading cost of each link is displayed next to it. The vehicle capacity $Q = 20$, therefore $Q_{max} = 30$, the penalty is $\omega = 5$ and $F = 0$. The giant tour that is the input to Split is $\delta = (1, 2, 3, 4, 5)$, which leads to the auxiliary DAG $\mathcal{H}$ in Figure 3.2(b). Each arc represents an allowed route, with its respective cost shown below the arc. Infeasible routes have their cost highlighted in red. The thicker arcs show the shortest path from node 0 to node n, with total cost 150, that corresponds to routes $\sigma_1 = (0, 1, 2, 0)$, $\sigma_2 = (0, 3, 0)$ and $\sigma_3 = (0, 4, 5, 0)$.

The time complexity of Split is $O(m)$, where m is the number of arcs in H. In the worst case $m = n(n+1)/2 = O(n^2)$, however a more accurate can complexity can be achieved by considering that each route contains at most $b = \lfloor Q_{max}/q_{min} \rfloor$ services, where q_{min} is the minimum demand of a required link, and therefore each node has at most b outgoing arcs and the complexity of Split becomes $O(nb)$.

The optimal MODE CHOICE for each service can also be derived in parallel with the Split Procedure. This variant is called Split with Flips and has the same time complexity of the standard Split. Since $\mathcal{H}$ is a DAG, a linear time shortest path algorithm can be used [42], which evaluates the nodes in increasing order and relaxes every outgoing arc from a node i before advancing to node $i + 1$. In Split, each of the outgoing arcs is evaluated in order from the smallest to the largest. From one arc to the next, the only difference is the addition of another service at the end of the route. Therefore, if we calculated the optimal mode choice for route c_{i+1} to c_j, we can calculate the optimal mode choices for route c_{i+1} to c_{j+1} by continuing the computation of the shortest path in the auxiliary graph $\mathcal{K}$ from the modes of service c_j



(a) Problem Graph



(b) Auxiliary DAG

Figure 3.2: Example of Split Procedure.

instead of restarting from the depot node. This trick is possible due to Bellman's Principle of Optimality.

The pseudocode of Split with Flips is available in Algorithm 3. The first loop in the algorithm corresponds to the calculation of the shortest path from 0 to n without explicitly building the auxiliary DAG $\mathcal{H}$, where the array p stores the distance of the shortest path to each node in the graph. At each iteration, the algorithm computes the penalized cost of route $\sigma = (0, \delta_{t+1}, \ldots, \delta_i, 0)$, using algorithm 2 and taking advantage of the trick previously mentioned. It then tries to relax the arc to see if it a new shortest path to node i has been found. In the second loop, the routes are retrieved using a system of predecessors, where the array $pred$ stores the previous node in the shortest path to each node in the DAG. If the current node is j and the predecessor of j in the shortest path is b, then we know by definition that the arc (b, j) represents a route servicing δ_{b+1} to δ_j. Array e stores the excess load of each route of the optimal solution and is used to check its feasibility.

3.3.2 Fitness

An individual I is composed of two parts: the giant tour δ_I and the set of routes R_I. The cost $C(I)$ of the solution that individual I represents is given by the sum of the penalized costs of each of its routes (equation 3.3).

Algorithm 3: Split with Flips

```
Data: Giant tour $\delta$, arc distance matrix $d(k,l)$, demands $q$, service costs $s$, vehicle capacity $Q$,
      fixed cost $F$, penalty $\omega$
Result: $Routes, solutionCost, feasible$
$p[0] \leftarrow 0$
$pred[0] \leftarrow 0$
$e[0] \leftarrow False$
for $t = 1$ to $n$ do
    $p[t] \leftarrow \infty$
    $pred[t] \leftarrow 0$
    $f[t] \leftarrow False$
$Q_{max} \leftarrow 1.5Q$
for $t = 0$ to $n - 1$ do
    $load \leftarrow 0$
    $serviceCosts \leftarrow 0$
    $PrevLabels \leftarrow [(0,0)]$                                         /* (cost,mode) */
    $i \leftarrow t + 1$
    while $i \leq n$ and $load + q_{\delta_i} \leq Q_{max}$ do
        $load \leftarrow load + q_{\delta_i}$
        $serviceCosts \leftarrow serviceCosts + s_{\delta_i}$
        $NewLabels \leftarrow [\,]$
        for $l \in M_{\delta_i}$ do
            $newCost \leftarrow \infty$
            for $prevCost, k \in PrevLabels$ do
                if $prevCost + d(k,l) < newCost$ then
                    $newCost \leftarrow prevCost + d(k,l)$
            Append $(newCost, l)$ to $NewLabels$
        $routeCost \leftarrow \infty$
        for $newCost, l \in NewLabels$ do
            if $newCost + d(l,0) < routeCost$ then
                $routeCost \leftarrow newCost + d(l,0)$
        $excessLoad \leftarrow \max(0, load - Q)$
        $routeCost \leftarrow routeCost + serviceCosts + F + \omega \times excessLoad$
        if $d[t] + routeCost < d[i]$ then
            $d[i] \leftarrow d[t] + routeCost$
            $pred[i] \leftarrow t$
            $e[i] \leftarrow excessLoad$
        $i \leftarrow i + 1$
        $PrevLabels \leftarrow NewLabels$
$Routes \leftarrow [\,]$
$solutionCost \leftarrow 0$
$feasible \leftarrow True$
$j \leftarrow n$
while $j > 0$ do
    $b \leftarrow pred[j]$
    $Route \leftarrow (0, \delta_{b+1}, \ldots, \delta_j, 0)$
    $solutionCost \leftarrow solutionCost + d[j] - d[b]$
    if $e[j] > 0$ then $feasible \leftarrow False$
    Append Route to Routes
    $j \leftarrow b$
```

$$C(I) = \sum_{\sigma \in R_I} C_P(\sigma) \tag{3.3}$$

The fitness $BF(I)$ of an individual I is based on two factors: the cost of the solution computed according to equation 3.3 and the diversity contribution of the individual with respect to the rest of the subpopulation.

The diversity contribution of an individual I is calculated as the average distance to its n_{close} closest neighbors. The distance between two individuals A and B is given by the broken pairs distance [44]. The distance $d_{\mathrm{BP}}(A, B)$ is equal to the number of adjacent services in A that are no longer adjacent in B. For example, if the giant tours of A and B are $\delta_A = (1,2,3,4,5,6)$ and $\delta_B = (6,4,5,3,1,2)$, then $d_{\mathrm{BP}}(A, B) = 3$, as the pairs $(2,3)$, $(3,4)$ and $(5,6)$ were broken. This distance measure takes integer values between 0 and $n-1$ and can be computed in $O(n)$.

Let $fit(I)$, with values in $1, ..., n_{ind}$, be the rank of an individual I with respect to its penalized cost in a subpopulation with n_{ind} individuals. The solution with the smallest penalized cost has rank 1 and the one with the largest cost has rank n_{ind}. Similarly, let $dc(I)$ be the rank of an individual I with respect to its diversity contribution, where the solution with the largest diversity contribution has rank 1. Then the biased $BF(I)$ of individual I is given by equation 3.4, where n_{elite} is the number of elite individuals.

$$BF(I) = fit(I) + \left(1 - \frac{n_{elite}}{n_{ind}}\right) dc(I) \tag{3.4}$$

The diversity contribution and the biased fitness are recalculated for every individual anytime an individual is added or removed from a subpopulation.

3.3.3 Parent Selection and Offspring Generation

In MADCoM, there are two ways to generate an offspring: selecting two parents and applying crossover or selecting an individual and mutating it by applying RCO and HD. The individuals that can be selected are always chosen at random from both subpopulations.

The individual that will undergo HD and RCO is selected by tournament selection, with a tournament size of 20, which selects at random 20 individuals and chooses the one with the smallest biased fitness. The large tournament size will frequently select the same solutions, however since HD and RCO are both random, it will not repeatedly generate the same solutions. Also, as selection is based on cost and diversity, if the same solution is selected several times for mutation its diversity will drop, resulting in a higher fitness and less of a chance to be selected for mutation.

The selection of the parents for crossover is done through binary tournament. To determine each parent, two individuals are selected randomly and the one with the smallest biased fitness is chosen. The offspring is generated using Order Crossover (OX), a crossover method suited to permutation schemes that seeks to transmit the relative order of the services from the parents to the offspring. The first step in OX is to choose at random two crossover points and copy the segment between them from one of the parents to the offspring. Then, starting from the second crossover point in the other parent, copy the remaining services in the order that they appear in the second parent, wrapping around when reaching the end of the permutation. The procedure is exemplified in Figure 3.3.

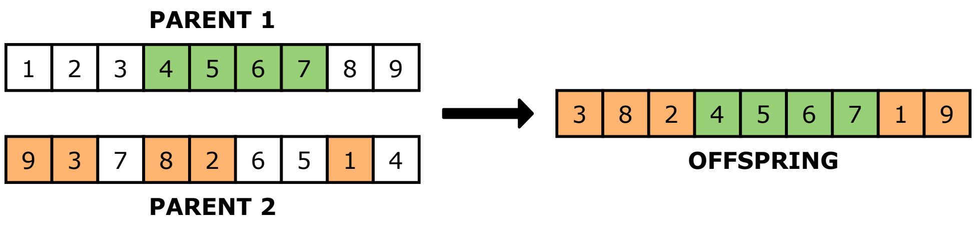


Figure 3.3: Example of Order Crossover (OX).

After the offspring is generated, Split must be applied to determine the cost and routes of the solution. It then undergoes local search with probability p_{LS} if its cost is at most 110% of the best-so-far solution. Afterwards it is added to the correct subpopulation depending on its feasibility. If the offspring is infeasible, it can be repaired with probability p_R to attempt to transform it into a feasible solution. If the solution generated after mutation or crossover improves the best feasible or infeasible solution found so far, it will always undergo local search, and be repaired if it is still infeasible after local search.

The repair operator consists of applying local search with a penalty parameter of 10 times its current value. If the resulting individual is still infeasible, the process is repeated but with a penalty parameter of 100 times its current value. If the repair operator is successful, the repaired offspring is added to the feasible subpopulation. By increasing the penalties for infeasible routes, the local search will prioritize moves that respect the vehicle capacity, as the reduction in cost from infeasible routes will not offset the increased penalties.

3.3.4 Survivor Selection

The size of a subpopulation n_{ind} is kept between μ and $\mu + \lambda$ individuals, where μ is the minimum population size and λ is the number of offspring per generation. When a subpopulation reaches the maximum size $\mu + \lambda$, survivor selection occurs.

Survivor selection chooses μ individuals from the initial $\mu+\lambda$ to remain in the population and continue to the next generation. At each iteration, the individual with the largest biased fitness is eliminated from the subpopulation, and, to favour a diverse population, individuals that share the same penalized solution cost, denominated clones, are eliminated first. In addition, the n_{elite} best individuals in terms of penalized solution cost are guaranteed to proceed to the next generation, due to the definition of biased fitness [45]. A pseudocode of survivor selection is displayed in Algorithm 4.

3.3.5 Population Initialization

The population is initialized by generating 4μ individuals and assigning to each subpopulation depending on their feasibility. Of these individuals, a fraction $f_{HD} = 0.20$ are generated using Hierarchical Decomposition, producing quality solutions. Following the same rules of offspring generation (subsection 3.3.3), these solutions can also undergo local search and be repaired. The remaining individuals

Algorithm 4: Survivor Selection

```
Data: A subpopulation with $n_{ind}$ individuals
$n_{ind} \leftarrow$ number of individuals in the subpopulation
while $n_{ind} > \mu$ do
    $X \leftarrow$ all individuals having a clone
    if $X = \emptyset$ then
        Remove $I \in X$ with maximum Biased Fitness
    else
        Remove $I$ with maximum Biased Fitness
    $n_{ind} \leftarrow n_{ind} - 1$
    Update the diversity contribution of each individual
    Recalculate the Biased Fitness of each individual
```

are generated randomly to introduce diversity into the population. These solutions do not undergo local search or repair unless they improve on the best feasible or infeasible solution found so far, as it is not beneficial in terms of time or solution quality when compared to improving solutions generated using Hierarchical Decomposition. Section 4.3 demonstrates this result.

3.3.6 Diversification Phase

One of the main problems with genetic algorithms is the premature convergence of the population, that is, when the individuals become very similar to each other and as a result the new offspring will not be very different from their parents and it becomes very difficult for the algorithm to improve the best-so-far solution. Using the biased fitness that promotes diversity mitigates this problem, but does not eliminate it completely. For that reason, MADCoM performs a diversification phase when the best-so-far solution has not been improved for It_{div} iterations. The diversification phase consists of keeping the $\mu/3$ individuals with the smallest penalized cost of each subpopulation and generating 4μ new individuals in the same way as when the population is initialized. The new individuals are added to their respective subpopulation and survivor selection is triggered to reduce each subpopulation to its minimum size.

3.3.7 Parameter Adjustment

The penalty parameter ω is initially set to $\omega = \overline{c}/\overline{q}$, where $\overline{c}$ is the average minimum cost between two services and $\overline{q}$ is the average demand. Every 100 iterations, ω is adjusted with the objective of achieving a target proportion ξ_{REF} of feasible individuals. By reducing penalties, the generation of infeasible solutions is promoted and vice-versa. Whenever the penalty ω is changed, the penalized costs of infeasible individuals are recalculated using the new penalty value. Let ξ be the number of feasible individuals in the last 100 iterations, then ω is adjusted in the following way:

- If $\xi \leq \xi_{REF} + 0.05$, then $\omega = \omega \times 1.2$
- If $\xi \geq \xi_{REF} - 0.05$, then $\omega = \omega \times 0.85$

The local search and repair probabilities, p_{LS} and p_R respectively, are adjusted using a different strategy. Both p_{LS} and p_R are initialized at 0.05 and every It_{LS} iterations without improvement they are increased by 0.10. If an improvement is found, p_{LS} and p_R are reduced by 0.10. By starting with a small value, the algorithm performs more exploration of the solution space as opposed to exploiting the solutions in the vicinity of the best-so-far solution. When an improvement is found, it is a local minimum, most likely in an area of the solution space with few chances for improvement and therefore the probabilities are reduced to increase exploration.

3.4 Local Search

3.4.1 Local Search Moves

The objective of local search is to improve a solution by exploring the neighborhood of solutions around it. The neighborhood is defined by a set of local search moves that alter a small part of the solution. In MADCoM, we consider 4 local search move types, exemplified in figure 3.4:

- SWAP - Swap two disjoint subsequences containing 1 or 2 services from the same route or from different routes.
- RELOCATE - Relocate a subsequence containing 1 or 2 services to another position in the same route or to another route.
- 2-OPT - Reverse a subsequence with at most 5 services.
- 2-OPT* - Swap two subsequences that end at the depot from different routes.

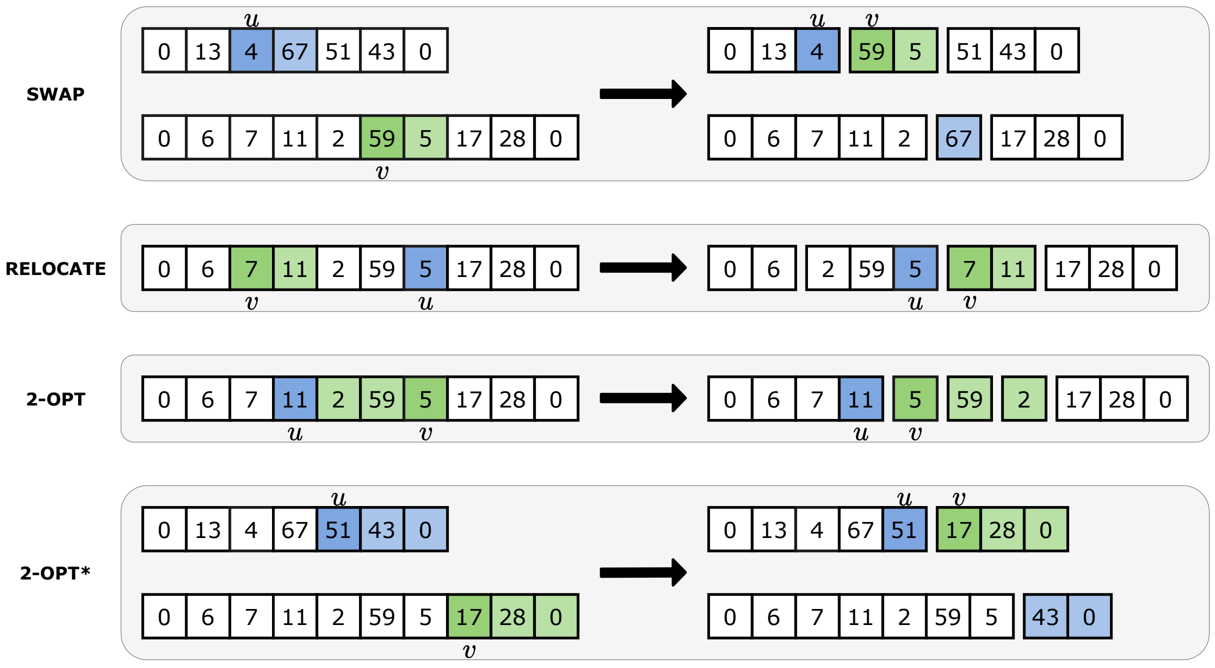


Figure 3.4: Examples of each local search move type.

Considering every possible move from each of the move types would lead to a neighborhood size of $O(n^2)$. Most of these moves would not lead to an improvement of the solution and for large-scale problems, the quadratic neighborhood size quickly becomes intractable. For these reasons, we only consider moves that originate a promising connection, that is, moves that place in sequence services that are close to each other. For a service u, local search will attempt to improve the solution with every move that places a service $v \in \Gamma(u)$ after u, where $\Gamma(u)$ is the set of services with the smallest minimum distance from u. With this restriction, the neighborhood size becomes $O(|\Gamma|n)$, where $|\Gamma| = 40$ is the size of the set $\Gamma(u)$.

3.4.2 Move Evaluations by Concatenation

To evaluate a local search move, the cost of the resulting routes needs to be computed. Also, as the mode choice for each service is not explicit, optimal mode choices need to be computed using algorithm 2, leading to a complexity of $O(n)$ for a move evaluation. However, this complexity can be reduced to $O(1)$ by computing the shortest path problem on a reduced graph (Figure 3.5).

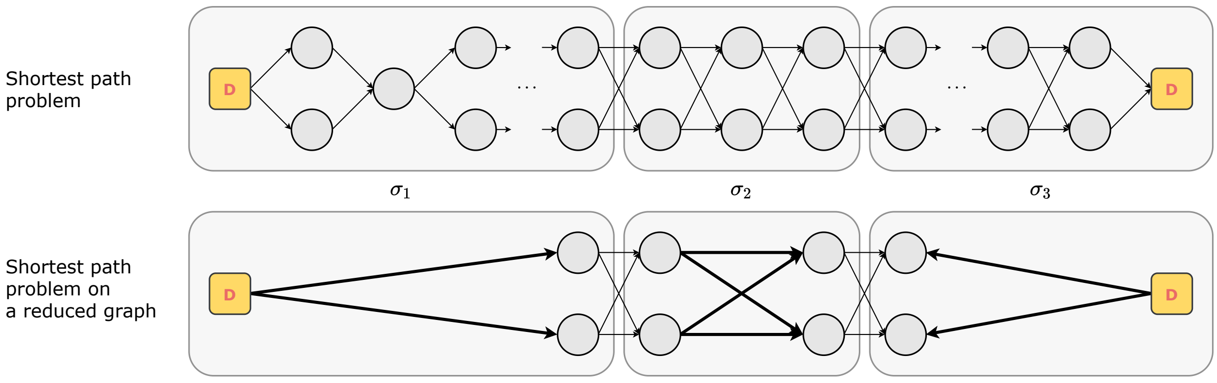


Figure 3.5: Reduced graph $\mathcal{K}$ for faster move evaluation with optimal mode choices.

The routes created by a local search move can be expressed in terms of K subsequences of the original routes, as exemplified by the right side of figure 3.4. The shortest path along a subsequence $\bar{\sigma} \subset \sigma$ from each mode k of service $\bar{\sigma}(1)$ to each mode l of service $\bar{\sigma}(|\bar{\sigma}|)$ can be preprocessed and stored in the auxiliary data structure $C(\bar{\sigma})[k, l]$, along with the demand $Q(\bar{\sigma})$ of the services in the subsequence. To evaluate the shortest path in the reduced graph, we concatenate the K subsequences that make up route $\sigma = \sigma_1 \oplus \cdots \oplus \sigma_K$ by applying equations 3.5 and 3.6 $K - 1$ times. This way, we skip the calculation of the shortest path for the services in each subsequence and compute the cost of the route in $O(1)$. After the cost of the route is computed, the fixed cost F is added and if it exceeds the vehicle capacity Q it is penalized according to equation 3.2.

$$C(\sigma_1 \oplus \sigma_2)[k, l] = \min_{x \in M_{\sigma_1(|\sigma_1|)}} \left\{ \min_{y \in M_{\sigma_2(1)}} \{C(\sigma_1)[k, x] + d(x, y) + C(\sigma_2)[y, l]\} \right\} \tag{3.5}$$

$$Q(\sigma_1 \oplus \sigma_2) = Q(\sigma_1) + Q(\sigma_2) \tag{3.6}$$

To compute the costs $C(\bar{\sigma})[k, l]$ we start with sequences containing only one service, $\bar{\sigma} = (i)$ and assign $C(\bar{\sigma})[k, l] = s_i$ if $k = l$, where s_i is the service cost of i, and $C(\bar{\sigma})[k, l] = +\infty$ if $k \neq l$. From there, the remaining subsequences are computed by concatenation using equations 3.5 and 3.6. This preprocessing step is applied every time the solution changes, so, to avoid excessive computation, we only compute subsequences that start or end at the depot, or with size smaller than 10. By using sequences that start or end at the depot, inter-route moves have at most 3 sequences, speeding up the calculating of these moves.

3.4.3 Lower Bounds on Move Evaluations

To speed up local search, we also filter moves using a lower bound on the cost of the new routes. A local search move changes at most two routes of the solution. Let $C_{\mathrm{LB}}(\sigma')$ be the lower bound on the cost of a new route σ'. Consider a local search move that changes routes σ_1 and σ_2 into routes σ_1' and σ_2'. Then if $C_{\mathrm{LB}}(\sigma_1')+C_{\mathrm{LB}}(\sigma_2') \geq C(\sigma_1)+C(\sigma_2)$, the move is discarded as it is guaranteed to not improve the solution.

Let $C_{\mathrm{MIN}}(\sigma)$, given by equation 3.7, be the minimum distance of a shortest path of subsequence σ among all modes k and l, and let c_{ij}^{MIN}, defined by equation 3.8, be the minimum distance between services i and j, among all possible mode choices of services i and j. The lower bound for a new route $\sigma' = \sigma_1 \oplus \cdots \oplus \sigma_K$ composed of K subsequences can be computed according to equation 3.9.

$$C_{\mathrm{MIN}}(\sigma) = \min_{k \in M_{\sigma(1)}} \left\{ \min_{l \in M_{\sigma(|\sigma|)}} \{C(\sigma)[k,l]\} \right\} \tag{3.7}$$

$$c_{ij}^{\mathrm{MIN}} = \min_{k \in M_i} \left\{ \min_{l \in M_j} \{d(k,l)\} \right\} \tag{3.8}$$

$$C_{\mathrm{LB}}(\sigma_1 \oplus \cdots \oplus \sigma_K) = \sum_{j=1}^{K} C_{\mathrm{MIN}}(\sigma_j) + \sum_{j=1}^{K-1} c^{\mathrm{MIN}}_{\sigma_j(|\sigma_j|)\sigma_{j+1}(1)} \tag{3.9}$$

The minimum subsequence costs $C_{\mathrm{MIN}}(\sigma)$ can be computed simultaneously with $C(\sigma)[k,l]$, while c_{ij}^{MIN} can be preprocessed at the same time as the distance matrix.

The pseudocode of local search is presented in Algorithm 5. For every service u, chosen in random order, the local search tries to place $v \in \Gamma(u)$ next to u, as discussed in subsection 3.4.1. The lower bound is calculated for every move that accomplishes this, and non-improving moves are discarded. For moves with a chance for improvement, the costs of the resulting routes are calculated as in subsection 3.4.2. The move that produces the largest improvement of the solution is applied and the auxiliary data structures $C(\sigma)[k,l]$, $C_{\mathrm{MIN}}(\sigma)$ and $Q(\sigma)$ are recomputed for the changed routes. The algorithm stops when an improving move can not be found, reaching a local minimum.

Algorithm 5: Local Search

```
Data: Individual $I$
$LocalMinimum \leftarrow False$
while $LocalMinimum$ = False do
    $MoveApplied \leftarrow False$
    $LocalMinimum \leftarrow True$
    Update auxiliary data structures
    for each service $u$ do
        for each service $v \in \Gamma(u)$ do
            $z_{\text{BEFORE}} \leftarrow$ sum of the costs of the routes containing $u$ and $v$
            $PM \leftarrow \emptyset$
            for every move $\phi$ that places $v$ after $u$ do
                Calculate the sequences produced by the move $\phi$, $(\sigma_1 \oplus \cdots \oplus \sigma_K)$ and
                 $(\sigma'_1 \oplus \cdots \oplus \sigma'_L)$
                $z_{\text{LB}} \leftarrow C_{\text{LB}}(\sigma_1 \oplus \cdots \oplus \sigma_K) + C_{\text{LB}}(\sigma'_1 \oplus \cdots \oplus \sigma'_L)$
                if $z_{\text{LB}} < z_{\text{BEFORE}}$ then
                    Add the sequences of $\phi$ to $PM$
            if $PM = \emptyset$ then
                continue (to the next service $v$)
            $\phi_{\text{BEST}} \leftarrow None$
            $z_{\text{BEST}} \leftarrow z_{\text{BEFORE}}$
            for every move $\phi \in PM$ do
                $z_{\text{AFTER}} \leftarrow C(\sigma_1 \oplus \cdots \oplus \sigma_K) + C(\sigma'_1 \oplus \cdots \oplus \sigma'_L)$
                if $z_{\text{AFTER}} < z_{\text{BEST}}$ then
                    $\phi_{\text{BEST}} \leftarrow \phi$
                    $z_{\text{BEST}} \leftarrow z_{\text{AFTER}}$
            if $z_{\text{BEST}} < z_{\text{BEFORE}}$ then
                Apply $\phi_{\text{BEST}}$
                $MoveApplied \leftarrow True$
                break
        if $MoveApplied = True$ then
            $LocalMinimum \leftarrow False$
            break
Update the giant tour $\delta_I$ by concatenating the improved routes
```

3.5 Large-Scale Heuristics

3.5.1 Introduction

A divide-and-conquer heuristic will divide the problem into smaller subproblems that are easier to solve due to their reduced size. After solving each subproblem, the solution to the original problem is found by merging the solutions of each subproblem.

The mutation operator of MADCoM consists of applying two divide-and-conquer heuristics to a solution in order to generate an offspring. The division is accomplished by Route Cutting Off Decomposition (RCO). RCO will segment a solutions' routes, outputting a virtual task set. Then, Hierarchical Decomposition (HD) takes as input the virtual task set and joins it together to form a giant tour, thereby generating an offspring. While forming the giant tour, the virtual tasks are ordered, which can be thought of as solving the subproblems. The mutation operator is outlined in algorithm 6. Sometimes, applying both heuristics can generate a clone of the parent. When that happens, we double the cutting probabilities of RCO and repeat the process. We impose a limit of 10 iterations to avoid an infinite loop, which is very rarely reached.

Algorithm 6: Mutation Operator

```
Data: Parent Individual $I$
Result: Offspring $O$
$i = 0$
while $i < 10$ do
    $i+ = 1$
    Apply RCO to $I$ to obtain a virtual task set $VT$
    Form a giant tour $\delta_O$ by applying HD to $VT$
    if $\delta_O = \delta_I$ then
        Double the cutting probabilities of RCO
    else
        Break
```

3.5.2 Route Cutting Off Decomposition

Route Cutting Off Decomposition segments a route into virtual tasks by cutting a link. Here, a link is defined as a sequence of two services (i, j) and cutting a route $\sigma = (0, \sigma_1, \dots, i, j, \dots \sigma_{|\sigma|-1}, 0)$ means generating two subsequences $(\sigma_1, \dots, i)$ and $(j, \dots \sigma_{|\sigma|-1})$, where the depot dummy services were discarded as the objective is to form a giant tour δ with Hierarchical Decomposition.

To choose which link to cut, RCO uses a task rank matrix Θ to evaluate the quality of each link and compares it to the average task rank $\bar{\theta}(S)$ of the solution S. Every row i of Θ contains the ranks of every link (i, j), with $j \neq i$, where the rank is calculated from the minimum distances between services c_{ij}^{MIN}. This means that $\Theta_{ib} = 1$ if the service b is the closest service to i, i.e., the link (i, b) has the lowest c_{ij}^{MIN} for all j. The task rank matrix is not symmetric, even though $c_{ij}^{\mathrm{MIN}} = c_{ji}^{\mathrm{MIN}}$ the ranks can be different as they are dependent on the other links in the row. By using ranks, Θ_{ij} represents the relative quality of having service j after service i in a route, compared to every other service.

It is important to note that the original authors used the average distance between the nodes of services i and j to define the distance between them. Here we use c_{ij}^{MIN} instead, the reason being that for arc services this distance will be skewed, as the path to one of the nodes will pass by the other and since we have optimal mode evaluations, the distance between services i and j in a route will more often than not be c_{ij}^{MIN}.

A good link is defined as a link whose task rank is smaller than the average task rank $\bar{\theta}(S)$ of the solution S, calculated from all the links in each route. Similarly, a poor link (i, j) is defined as a link with $\Theta_{ij} > \bar{\theta}(S)$. For every route of a solution S, RCO identifies the good and poor links. Then, with probability $p_{gl} = 0.05$ cuts one random good link and with probability $p_{pl} = 0.20$ cuts one random poor link. The values of p_{gl} and p_{pl} are kept equal to those of the original paper [40]. The pseudocode of RCO is shown in algorithm 7.

Algorithm 7: Route Cutting Off Decomposition

```
Data: Solution $S$, task rank matrix $\Theta$, cutting probabilities $p_{gl}$ and $p_{pl}$
Result: Virtual task set $VT$
$VT \leftarrow \emptyset$
Calculate the average task rank $\bar{\theta}(S)$ of the solution $S$ based on $\Theta$
for each route $\sigma \in S$ do
    $GL \leftarrow \emptyset$
    $PL \leftarrow \emptyset$
    for each link $(\sigma_i, \sigma_{i+1})$ of $\sigma$ do
        if $\Theta_{\sigma_i \sigma_{i+1}} < \bar{\theta}(S)$ then
            $GL \leftarrow GL \cup (\sigma_i, \sigma_{i+1})$
        else
            $PL \leftarrow PL \cup (\sigma_i, \sigma_{i+1})$
    $gl \leftarrow None$
    $pl \leftarrow None$
    Randomly generate two numbers $r_1, r_2 \in [0, 1]$
    if $r_1 < p_{gl}$ then
        Randomly select a good link $gl$ from $GL$
    if $r_2 < p_{pl}$ then
        Randomly select a poor link $pl$ from $PL$
    Cut off $gl$ and $pl$ to obtain virtual tasks $(VT_1, \dots)$
    $VT \leftarrow VT \cup (VT_1, \dots)$
```

3.5.3 Hierarchical Decomposition

Hierarchical Decomposition (HD) seeks to form a giant tour from an initial virtual task set VT, where a virtual task τ_i^l is a permutation of several services. HD constructs a hierarchical structure (Figure 3.6) where the initial virtual task set forms the bottom layer and the next layer is formed by grouping them. The virtual tasks of each group are ordered and concatenated to form a new virtual task τ_i^2 for layer 2. The procedure continues until only one virtual task remains, a giant tour δ.

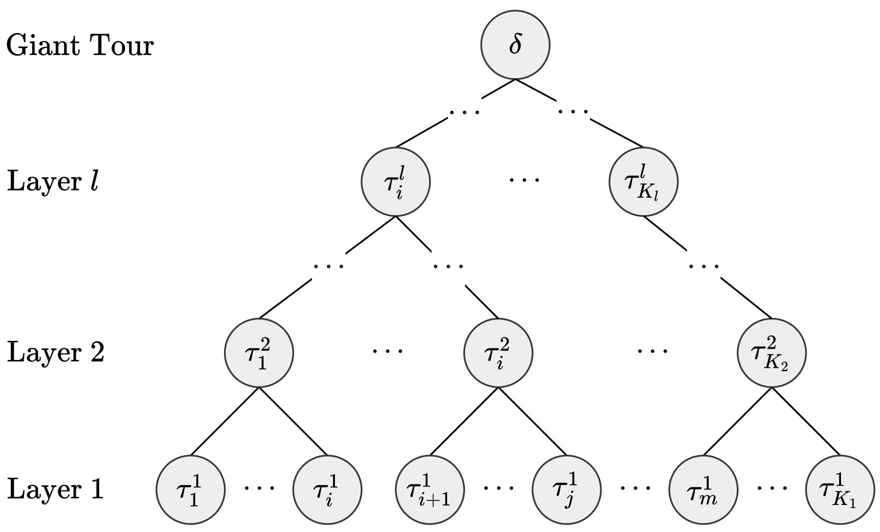


Figure 3.6: Hierarchical Structure of HD.

To form layer $l + 1$, HD selects a random number of clusters $K_{l+1} \in [1, \beta K_l]$, where $\beta = 0.1$ and $K_L = |VT_l|$ is the number of virtual tasks at layer l, and then forms K_{l+1} groups using a clustering algorithm. Equation 3.10 defines the distance measure between two virtual tasks $\tau_i^l = (p, \ldots, q)$ and $\tau_j^l = (r, \ldots, s)$ that will be used by the clustering algorithm to derive the clusters.

$$d(\tau_i^l, \tau_j^l) = \frac{1}{2}\left(c_{qr}^{\mathrm{MIN}} + c_{sp}^{\mathrm{MIN}}\right) \tag{3.10}$$

It is important to note that the original authors of HD used as a distance measure the average distance between the start and end vertices of each virtual task. Using that distance measure would require storing another distance matrix, where as c_{ij}^{MIN} is already stored as it is essential for local search. Furthermore, the reasoning given in section 3.5.2 for using c_{ij}^{MIN} in RCO also applies here, as the distance between virtual tasks is calculated using the distance between services of each virtual task.

The clustering algorithm used is k-medoids with random sampling initialization based on the paper by Schubert and Rousseeuw [46]. K-medoids is similar to k-means, the main difference being that the center of each cluster is chosen to be one of the data points, denominated medoid. Since only a distance matrix is available, the center of a cluster can not be chosen as a point in a coordinate space and we must choose instead one of the data points. K-medoids has a time complexity of $O(n(n-k))$, where n is the number of data points, in this case $n = |VT_l|$ and k is the number of clusters.

Once the virtual tasks are grouped into clusters, they are ordered using the Best Insertion Heuristic (BIH) and then concatenated to form a new virtual task. BIH starts by choosing the virtual task whose first service is closest to the depot and then chooses the virtual task whose first service is closest to the last service of the last virtual task chosen.

Hierarchical Decomposition can also be used as an initialization method, by having the initial virtual task set be the set of all services. The solutions generated like this are typically of high quality, as shown in section 4.3.

The pseudocode of Hierarchical Decomposition is available in algorithm 8.

Algorithm 8: Hierarchical Decomposition

```
Data: Virtual task set $VT$, minimum distance matrix $c_{ij}^{\mathrm{MIN}}$, parameter $\beta$
Result: Giant tour $\delta$
repeat
    Randomly choose the cluster number $K_{l+1} \in [1, \beta|VT_l|]$
    Build the distance matrix $d(\tau_i^l, \tau_j^l)$
    Divide $VT$ into $K_{l+1}$ groups using k-medoids
    Order the virtual tasks in each group using BIH to form new virtual task set $NVT$
    $VT \leftarrow NVT$
until $|VT| = 1$
$\delta \leftarrow VT(1)$
```

Chapter 4

Results

4.1 Experimental Setup

MADCoM was implemented in Python 3.9.7 with the help of the library `DEAP` [47] for the implementation of the genetic algorithm, and the graph algorithms of the library `networkx` [48]. To run MADCoM we used a E2ds_v4 virtual machine from Azure running Windows Server 2019 Datacenter, with 16 GB of RAM and a Intel(R) Xeon(R) Platinum 8272CL processor with a frequency of 2.60 GHz.

To evaluate the performance of MADCoM and compare it to existing algorithms, the benchmarks available in the literature will be used. Table 4.1 gives the characteristics of the instances in the classical benchmarks and Table 4.2 the characteristics of the more recent large-scale benchmarks. For each benchmark, column $\#$ gives the number of instances and column $|N|$ is the minimum and maximum number of nodes. Likewise, E_R is the number of required edges, A_R is the number of required arcs and n is the number of services, i.e., the problem size.

Table 4.1: Characteristics of the instances in the classical benchmarks.

Benchmark	$\#$	$\|N\|$	E_R	A_R	n	Description
GDB [17]	23	[7,27]	[11,55]	0	[11,55]	Random graphs; Only required edges
VAL [49]	34	[24,50]	[34,97]	0	[34,97]	Random graphs; Only required edges
BMCV [34]	100	[26,97]	[28,121]	0	[28,121]	Intercity road network in Flanders
EGL [50]	24	[77,140]	[51,190]	0	[51,190]	Winter-gritting application in Lancashire
MVAL [11]	34	[24,50]	[12,44]	[25,106]	[43,138]	Adapted to MCARP from the VAL benchmark
LPR [11]	15	[28,401]	[0,387]	[11,764]	[50,806]	Random graphs that mimic the shape of street networks

Table 4.2: Characteristics of the instances in the large-scale benchmarks.

Benchmark	$\#$	$\|N\|$	E_R	A_R	n	Description
EGL-L [32]	10	255	[347,375]	0	[347,355]	Larger winter-gritting application
Hefei [39]	10	850	[121,1212]	0	[121,1212]	Generated from the road network of Hefei, China
Beijing [39]	10	2820	[358,3584]	0	[358,3584]	Generated from the road network of Beijing, China
KW [4]	12	[788,6149]	[686,3797]	0	[686,3797]	Based on real-life networks and waste data from five areas in Denmark

4.2 Parameter Tuning

Parameter tuning is a crucial step in the development of metaheuristics. The parameters of metaheuristics can greatly influence its performance on a given instance and, more importantly, its generality, that is, its performance across instances with different sizes and characteristics.

The objective in parameter tuning is to find a configuration of parameters that maximizes some performance measure. Since MADCoM has inherent randomness, the result of two different runs will probably be different. Additionally, the relationship between parameter values and the performance of a configuration is unknown and can't be derived. Therefore, parameter tuning is a stochastic black-box optimization problem [51], where the black-box function we want to maximize is the expected performance of a configuration γ.

Based on the approach used in [52], we use the estimator $\hat{\mu}_\gamma$ (equation 4.1) to estimate the performance of a configuration γ. A run of 20 minutes is performed on each instance $i \in I$ and the gap from the best solution found c_i to the best-known solution c_i^{BKS} is computed according to equation 4.2. The expected performance of a configuration is calculated by averaging the gaps across all instances in set I, which is detailed in table 4.3.

$$\hat{\mu}_\gamma = -\frac{1}{|I|}\sum_{i \in I} \text{GAP}_i(c_i) \tag{4.1}$$

$$\text{GAP}_i(c_i) = \frac{c_i - c_i^{\text{BKS}}}{c_i^{\text{BKS}}} \tag{4.2}$$

Table 4.3: Characteristics of the instances used for parameter tuning.

Instance	Benchmark	$\|N\|$	E_R	A_R	n
egl-s4-C	EGL	140	190	0	190
egl-g2-E	EGL-L	255	375	0	375
Lpr-c-04	LPR	277	362	142	504
Hefei-6	Hefei	850	727	0	727
F1_g-4	KW	812	780	0	780
Beijing-3	Beijing	2820	1075	0	1075

To search for the configuration with the best performance, we apply Bayesian Optimization using the implementation in the library `bayesian-optimization` [53]. Bayesian optimization constructs a posterior distribution of functions that describes the function being maximized. The posterior distribution is then used at each iteration to sample the next configuration to evaluate, meaning each configuration tested informs the search for better configurations.

MADCoM has several parameters that need to be set, related to the genetic algorithm and the divide-and-conquer heuristics used in mutation. We focus on the most important parameters of its genetic algorithm component, described in Table 4.4. For the parameters of HD and RCO, we use the values that showed best performance in the original papers. After 75 iterations of Bayesian optimization, we settled on the parameters in Table 4.4 by averaging the best configurations found.

Table 4.4: Tuned parameters of MADCoM.

Parameter	Description	Value
μ	Minimum population size	20
λ	Number of offspring until survivor selection	30
n_{elite}	Number of elite individuals	5
n_{close}	Number of individuals used for diversity contribution	5
It_{LS}	Number of iterations until p_{LS} and p_R are updated	60
p_M	Mutation probability	0.25

4.3 Comparison of Initialization Methods

When using Hierarchical Decomposition as an initialization method, the size of the initial virtual task set is equal to the problem size n. Given the quadratic complexity of the clustering algorithm, HD can be computationally expensive for larger problems, especially because it requires clustering several times. In the original paper, it was observed that for larger instances the parameter β, that sets the maximum number of clusters in a layer, did not influence significantly the performance of SAHiD. For these reasons, we experiment with imposing a limit on the maximum number of clusters using two alternative variants:

- Maximum of 10 clusters per layer. With $\beta = 0.1$ this results in having only two layers regardless of the problem size, as the second layer will simply order the virtual task set to form the giant tour.
- Maximum of $\sqrt{n}$ clusters per layer. This will generate few layers, as the number of clusters per layer is randomly selected from $[1, \min(\beta|VT|, \sqrt{n})]$.

For control purposes, we also include the classical HD where no limit is imposed, i.e., the number of clusters in each layer is selected from $[1, \beta|VT|]$, where $|VT|$ is the size of the virtual task set at the layer and $\beta = 0.1$. A population of 100 individuals was generated using each method for four instances of different sizes. Figure 4.1 shows the boxplots of the time to generate an individual, for all methods across the four instances. Figure 4.2 shows the boxplots of the gap to the best known solution.

As can be observed, without a limit on the maximum number of clusters, the execution time of HD can be much larger than when a limit is imposed. Also, it presents a higher variability and the larger the instance the more pronounced the effect becomes. In terms of solution quality, not imposing a limit leads to higher variability and on average worse solutions. While for the smallest instance, all variants demonstrate similar performance, it is clear that for larger instances imposing a limit results in a reduction of execution time and an improvement of solution quality.

Another important factor is the diversity of the population. By imposing a limit on the number of clusters in a layer, the number of outcomes of HD is also reduced. Table 4.5 displays the number of individuals that have a clone in the population, for each method and instance. With a maximum of 10 clusters, regardless of the problem size, most solutions will be duplicated. For the other variants the number of clones is smaller, but still larger than zero, indicating that using only HD to initialize the population would likely lead to wasted computational effort.

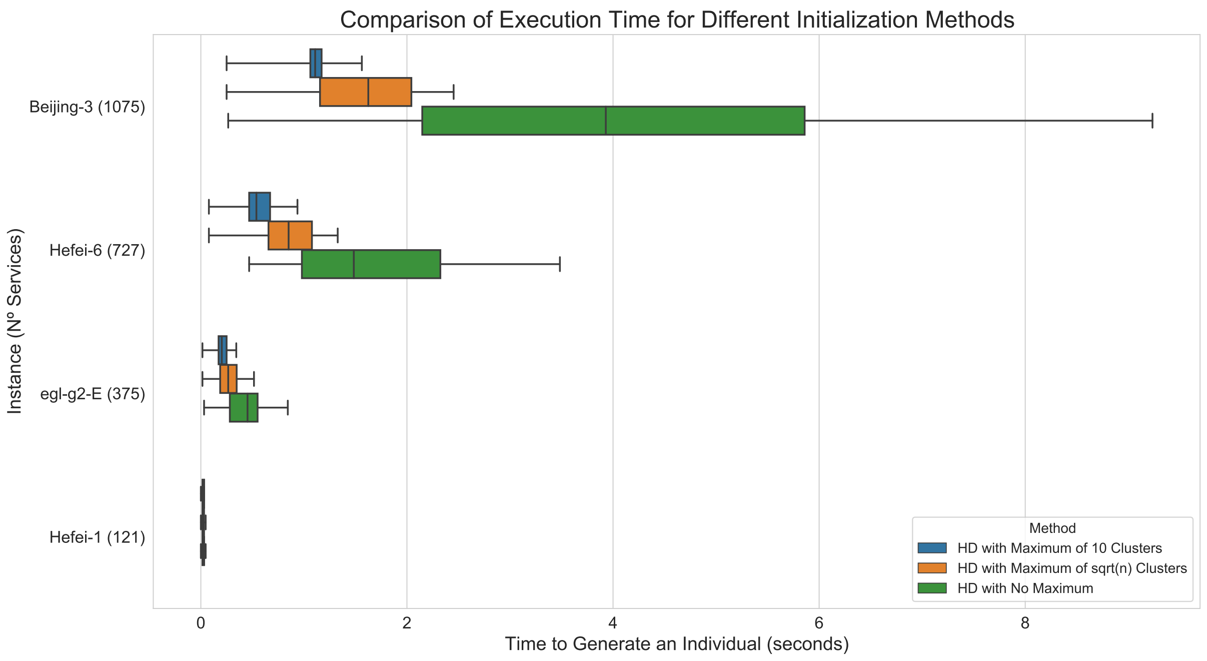


Figure 4.1: Boxplots of the time to generate an individual using different variants of HD.

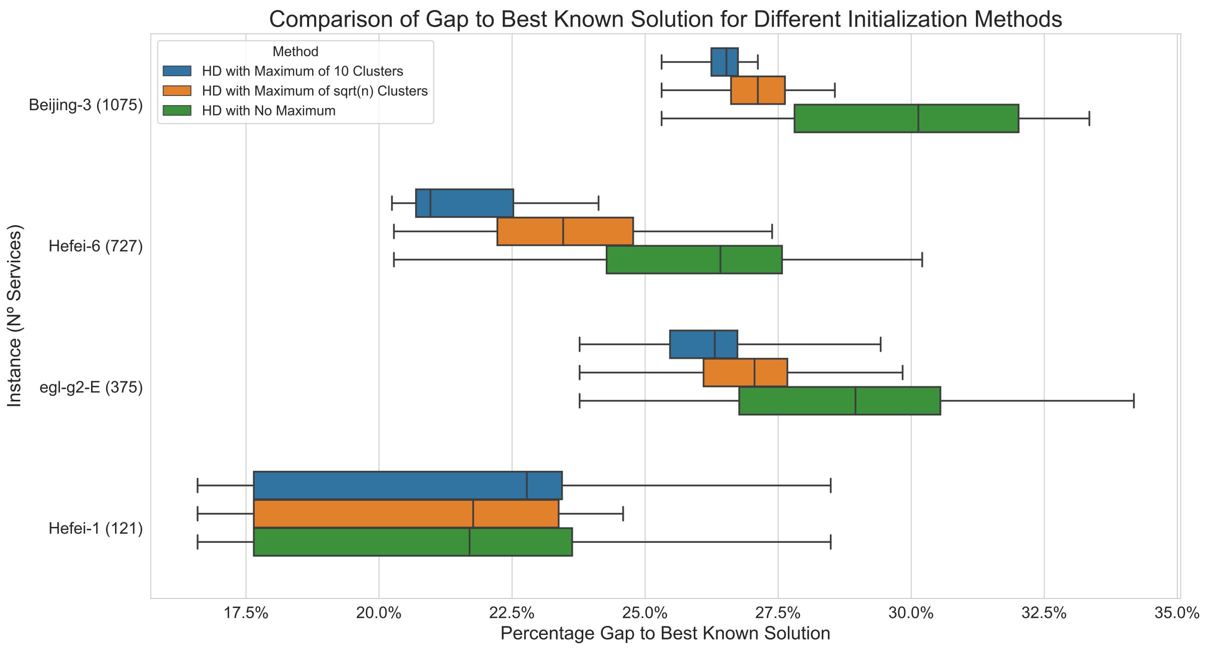


Figure 4.2: Boxplots of the gap to the best known solution using different variants of HD.

Table 4.5: Number of clones generated by each variant of HD.

Instance	Maximum of 10 Clusters	Maximum of $\sqrt{n}$ Clusters	No Maximum
Hefei-1	97	97	93
egl-g2-E	96	53	16
Hefei-6	93	49	13
Beijing-3	87	39	10

While generating the initial population, some individuals will undergo local search. To compare the performance of each variant when combined with local search, the individuals previously generated with HD were improved. For control purposes, a population of random individuals improved by local search was also generated. Figure 4.3 shows the boxplots of the total time to generate an individual and Figure 4.4 shows the solution quality. The boxplots of the solution quality of each HD variant was also included for easier comparison.

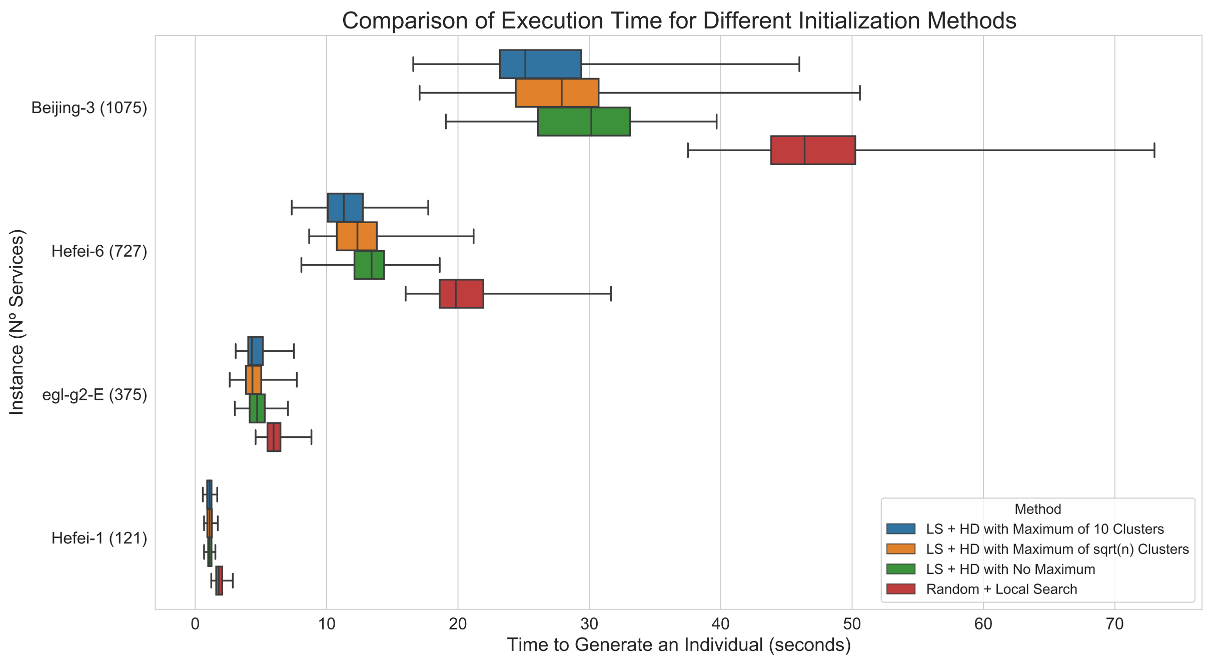


Figure 4.3: Boxplots of the total time to generate an individual using different variants of HD combined with local search.

Imposing a limit reduces the total time to generate an individual and the solutions are better on average. In fact, for the largest instance, solutions generated only with HD are significantly better than random solutions improved by local search, demonstrating the benefits of using HD as an initialization method. When no limit is imposed, the time to generate an individual is larger and the added computational effort does not translate into better solutions on average.

Figure 4.5 shows the boxplots of the diversity contribution of each individual in the population, normalized by the maximum diversity $n-1$. With local search, the populations no longer have clones due to its inherent randomness. The populations generated with HD are not as diverse as the random individuals, especially for smaller instances.

Summarizing, imposing a limit on the number of clusters reduces the computational effort of HD with no significant reduction on solution quality. In MADCoM, we choose the limit $\sqrt{n}$ as it is similar in time and quality to the limit of 10, but generates a more diverse population. Additionally, HD combined with local search generates solutions with higher quality in shorter time than random solutions improved with local search.

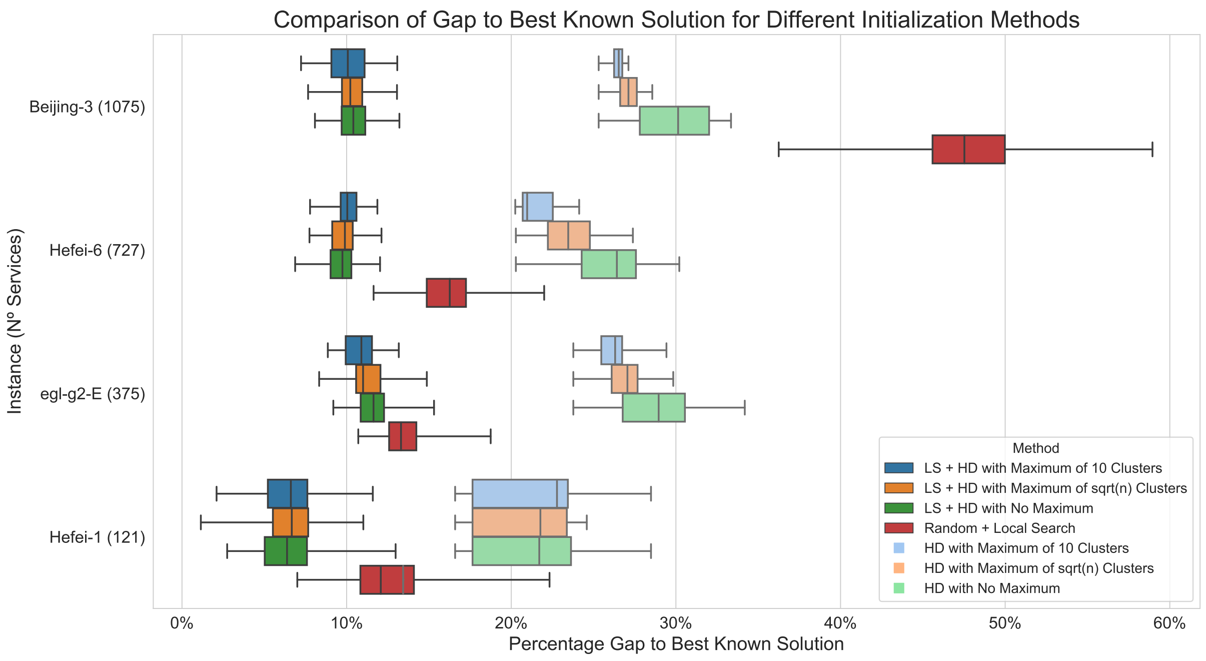


Figure 4.4: Boxplots of the gap to the best known solution using different variants of HD combined with local search.

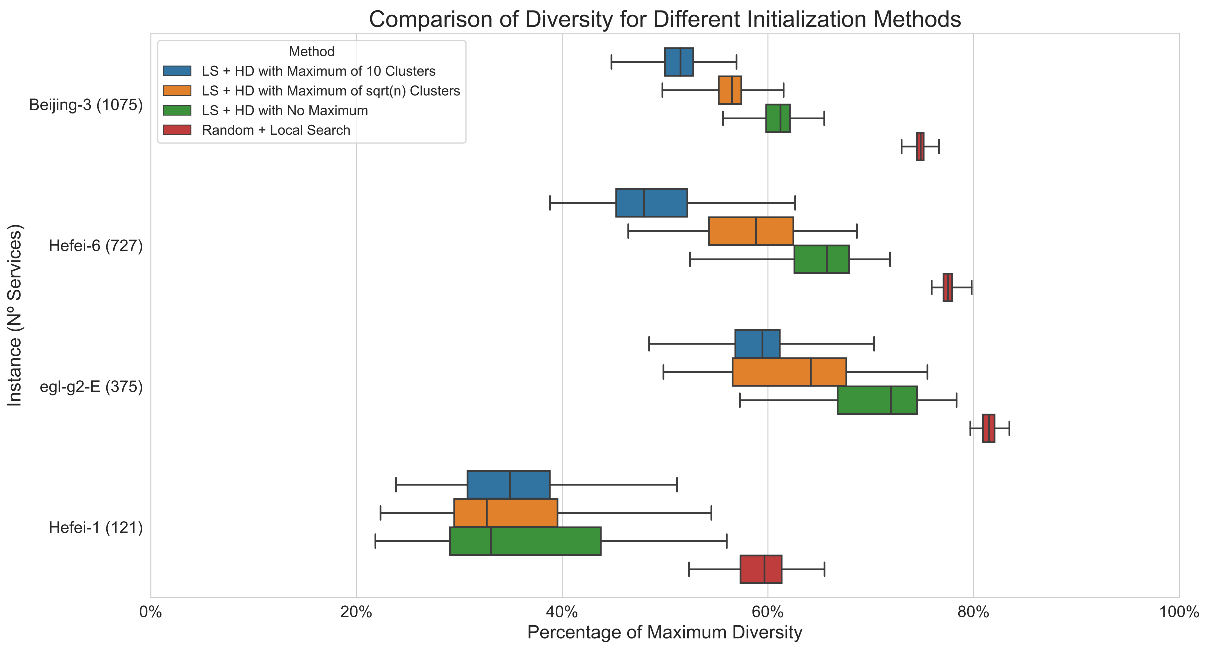


Figure 4.5: Boxplots of the diversity contribution of each individual in the population generated using different variants of HD combined with local search.

4.4 Results on Classical Benchmarks

In this section, we present the results of MADCoM on the classical CARP and MCARP benchmarks in Table 4.1. We performed five runs of 30 minutes on each instance. For comparison, we also present the results of other algorithms in the literature, whose acronyms and references are available in Table 4.6.

Table 4.6: Algorithms for comparison in the classical benchmarks.

Acronym	Name
UHGS	Unified Hybrid Genetic Search [41]
MAENS	Memetic Algorithm with Extended Neighborhood Search [37]
VNS	Variable Neighborhood Search [31]
MABBLP	Memetic Algorithm by Belenguer et. al [11]

Due to the large number of instances, we only present the summarized results in Table 4.6. The results for each instance are available in appendix B. For each algorithm, the column ”Best(%)” displays the average across all instances of the percentage gap to the best known solution of the best solution found in the five runs, and column ”Mean(%)” is the average gap to the best known solution across all instances. For each algorithm, the column ”Best(%)” displays the percentage gap of the best solution in the five runs, calculated according to equation 4.2 and averaged across all instances in the benchmark set. Column ”Mean(%)” is the percentage gap of the mean of the five runs, averaged across all instances in the benchmark set. In column ”Nº BKS”, we display the number of instances in the benchmark for which the algorithm found a solution equal or better than the best known solution.

On the classical CARP benchmarks, MADCoM performs worse than the all algorithms, with a maximum difference in the gap of 0.502 % when compared to the best algorithm. On the MVAL benchmark set, MADCoM outperforms MABBLP, and manages to find new best solutions for 8 instances, 2 of which are optimal solutions as they match the best lower bound in the literature. On the LPR benchmark, which contains larger MCARP instances, MADCoM performs worse than MABBLP.

4.5 Results on Large-Scale Benchmarks

In this section, we present the results of MADCoM on the large-scale benchmarks in Table 4.2. Five runs of 30 minutes were performed on each instance. For comparison, we also present the results of other algorithms in the literature, whose acronyms and references are available in Table 4.8.

The results of MADCoM on the large-scale benchmarks sets (4.2) are available in Tables 4.9-4.12. In each table, n denotes the problem size and BKS the best known solution for each instance. For each comparing algorithm, Best denotes the cost of the best solution found across all runs and Mean the average cost of the best solution found in each run. For MADCoM, we also report the percentage gap to the best known solution of both the best solution found and the mean. A value is in bold if it has the minimal cost among all algorithms, and it is underlined if it is larger than the corresponding value of MADCoM.

Table 4.9 shows the results on the EGL-L benchmark set. MADCoM performs worse than all of the comparing algorithms across all instances. Another observation is the performance of MADCoM is significantly different among instances with the same problem size. The main difference between these instances lies in the number of vehicles. On instances with a larger fleet size, MADCoM performs worse. Since the Split procedure is optimal, this points to a weakness in the local search, in particular

Table 4.7: Summarized results on classical benchmarks.

Benchmark	n	Algorithm	Best(%)	Mean(%)	Nº BKS
GDB	[11,55]	MAENS	0.000%	0.009%	23/23
		UHGS	0.000%	0.000%	23/23
		MADCoM	0.077%	0.123%	22/23
VAL	[34,97]	UHGS	0.013%	0.041%	32/34
		MADCoM	0.051%	0.073%	31/34
BMCV	[28,121]	UHGS	0.003%	0.013%	99/100
		MADCoM	0.035%	0.073%	92/100
EGLESE	[51,190]	VNS	0.174%	0.624%	14/24
		MAENS	0.211%	0.651%	12/24
		UHGS	0.047%	0.139%	19/24
		MADCoM	0.483%	0.641%	11/24
MVAL	[43,138]	MABBLP	0.000%	0.202%	34/34
		MADCoM	-0.238%	-0.185%	34/34
LPR	[50,806]	MABBLP	0.000%	0.090%	15/15
		MADCoM	0.551%	0.703%	5/15

Table 4.8: Algorithms for comparison in the large-scale benchmarks.

Acronym	Name
RDG-MAENS	Route Distance Grouping combined with MAENS
UHGS	Unified Hybrid Genetic Search [41]
RCO-RDG-MAENS	Route Cutting Off Decomposition combined with RDG-MAENS [14]
RCO-SAHiD	Route Cutting Off Decomposition combined with SAHiD [14]
FastCARP	FastCARP [5]
PS	Path-Scanning [17]

the inter-route moves.

Table 4.9: Results on the EGL-L benchmark set.

									MADCoM			
			RDG-MAENS		UHGS		RCO-RDG-MAENS		Best		Mean	
Instance	n	BKS	Best	Mean	Best	Mean	Best	Mean	Cost	Gap	Cost	Gap
egl-g1-A	347	991176	998405	1007368	**992227**	993127	998763	1005870	1010493	1.95 %	1019633	2.87 %
egl-g1-B	347	1109656	1118030	1123369	**1112149**	1116617	1118030	1121529	1135636	2.34 %	1144064	3.10 %
egl-g1-C	347	1230155	1242897	1251029	**1232501**	1236062	1243096	1250070	1268701	3.13 %	1277029	3.81 %
egl-g1-D	347	1361862	1375583	1384902	**1365393**	1370963	1375319	1383355	1409894	3.53 %	1418436	4.15 %
egl-g1-E	347	1501801	1518694	1527631	**1503467**	1511572	1513589	1526503	1559785	3.86 %	1569198	4.49 %
egl-g2-A	375	1086932	1097581	1106082	**1087353**	1090396	1097291	1106843	1115394	2.62 %	1127162	3.70 %
egl-g2-B	375	1196873	1211805	1223706	**1198633**	1202901	1211789	1220454	1239375	3.55 %	1245825	4.09 %
egl-g2-C	375	1330744	1344228	1353819	**1333430**	1336104	1344353	1352802	1377353	3.50 %	1387605	4.27 %
egl-g2-D	375	1468310	1482216	1492745	**1471783**	1476285	1482345	1490704	1528559	4.10 %	1533628	4.45 %
egl-g2-E	375	1602229	1622927	1633192	**1610919**	1616556	1621354	1631378	1675262	4.56 %	1683165	5.05 %

The results on benchmark set Hefei are available in Table 4.10. MADCoM performs worse than

UHGS across all instances and only outperforms RCO-SAHiD on the smallest instance Hefei-1, but finds better solutions than RDG-MAENS in 4 out of 10 instances. Also, MADCoM performs on average better than RDG-MAENS on 7 out of 10 instances. On the Beijing benchmarks set, in Table 4.11, MADCoM performs worse than UHGS and RCO-SAHiD on all instances, but performs better than RDG-MAENS on all instances, both on average and in the best solution found.

Table 4.10: Results on the Hefei benchmark set.

									MADCoM			
			RDG-MAENS		UHGS		RCO-SAHiD		Best		Mean	
Instance	n	BKS	Best	Mean	Best	Mean	Best	Mean	Cost	Gap	Cost	Gap
Hefei-1	121	245596	246221	247341	**245596**	**245596**	246571	247351	246161	0.23 %	246501	0.37 %
Hefei-2	242	433648	436020	441539	**433648**	433807	436031	437631	437431	0.87 %	438778	1.18 %
Hefei-3	364	572545	583050	589152	**572545**	573737	582839	586795	588239	2.74 %	590044	3.06 %
Hefei-4	485	737730	754855	761351	**737730**	740404	750687	753859	764430	3.62 %	767514	4.04 %
Hefei-5	606	941278	980153	991813	**941278**	946574	961376	967045	982810	4.41 %	993578	5.56 %
Hefei-6	727	1068035	1119584	1132063	**1068035**	1072864	1092667	1098915	1119480	4.82 %	1124864	5.32 %
Hefei-7	848	1266931	1329745	1361125	**1266931**	1272880	1299360	1305057	1334139	5.30 %	1337509	5.57 %
Hefei-8	970	1427531	1526453	1550509	**1427531**	1436048	1469819	1478098	1498918	5.00 %	1510409	5.81 %
Hefei-9	1091	1598203	1705381	1749079	**1598203**	1605554	1645841	1656147	1676545	4.90 %	1681245	5.20 %
Hefei-10	1212	1748829	1837767	1923264	**1748829**	1754889	1799158	1810301	1839707	5.20 %	1846897	5.61 %

Table 4.11: Results on the Beijing benchmark set.

									MADCoM			
			RDG-MAENS		UHGS		RCO-SAHiD		Best		Mean	
Instance	n	BKS	Best	Mean	Best	Mean	Best	Mean	Cost	Gap	Cost	Gap
Beijing-1	358	760578	812647	829406	**760578**	**760578**	765538	770199	771892	1.49 %	780293	2.59 %
Beijing-2	717	1129810	1303570	1337954	**1129810**	1132987	1148259	1163978	1190494	5.37 %	1210427	7.14 %
Beijing-3	1075	1534878	1777852	1847922	**1534878**	1542405	1563874	1577027	1628115	6.07 %	1648438	7.40 %
Beijing-4	1434	1836866	2126151	2193399	**1836866**	1847355	1879617	1896581	1963803	6.91 %	1989004	8.28 %
Beijing-5	1792	2199275	2581910	2639458	**2199275**	2210443	2234352	2255386	2347314	6.73 %	2361021	7.35 %
Beijing-6	2151	2561113	2968102	3047295	**2561113**	2571748	2632250	2650420	2737955	6.90 %	2767488	8.06 %
Beijing-7	2509	2851602	3331900	3388263	**2851602**	2871881	2925015	2952809	3066896	7.55 %	3088432	8.31 %
Beijing-8	2868	3136727	3584696	3697025	**3136727**	3150688	3203032	3233296	3360983	7.15 %	3386317	7.96 %
Beijing-9	3226	3462953	3934270	4061793	**3462953**	3485819	3541842	3575671	3719620	7.41 %	3732243	7.78 %
Beijing-10	3584	3765614	4206005	4353966	**3765614**	3785520	3852428	3884308	4047304	7.48 %	4061605	7.86 %

Table 4.12 shows the results of MADCoM on the KW benchmark set. MADCoM finds better solutions for all instances, with improvements ranging from 0.56 % to 6.94 % of the previous best known solution. These instances have previously only been solved by constructive heuristics, which explains the significant improvements on some instances.

4.6 Comparison with Simpler Versions

MADCoM is an algorithm that integrates two main components: the local search of UHGS and large-scale heuristics, RCO and HD. It is important to show that it is the combination of these components that leads to better results and not simply one of them. For this reason, we compare the performance of MADCoM with simpler versions without one of the components. The parameters used for each alternative version are the same as those of MADCoM. Each alternative version is tested on 5 runs of 30 minutes on the instances of Table 4.3.

To analyse the performance of the large scale heuristics, we tested simpler versions without the

Table 4.12: Results on the KW benchmark set.

			FastCARP	PS	MADCoM			
					Best		Mean	
Instance	n	BKS	Best	Best	Cost	Gap	Cost	Gap
F1_g-4	780	768209	768209	843418	**727490**	-5.30 %	731891	-4.73 %
F1_g-6	780	474809	474809	530687	**460267**	-3.06 %	461224	-2.86 %
F1_p-2	728	261743	261743	287581	**249859**	-4.54 %	251383	-3.96 %
F1_p-6	728	162376	162376	182656	**151105**	-6.94 %	152147	-6.30 %
F9_g-4	686	718145	718145	795868	**683088**	-4.88 %	686199	-4.45 %
F9_g-6	686	444369	444369	496164	**430558**	-3.11 %	431320	-2.94 %
S1_g-1	3797	3624502	3624502	3987605	**3582342**	-1.16 %	3594765	-0.82 %
S1_g-6	3797	1470679	1478193	1886632	**1452374**	-1.24 %	1458265	-0.84 %
S2_g-1	3797	3459811	3459811	3822136	**3405300**	-1.58 %	3418296	-1.20 %
S2_g-6	3797	1439140	1478328	1851578	**1431048**	-0.56 %	1441136	0.14 %
S5_g-2	3732	2191441	2197632	2588801	**2157168**	-1.56 %	2165719	-1.17 %
S5_g-6	3732	1413134	1443084	1841206	**1402146**	-0.78 %	1405855	-0.52 %

divide-and-conquer mutation, by setting $p_M = 0$, and without initialization using HD, by setting $f_{HD} = 0$. The effect of using HD as initialization method was analysed in section 4.3, however only in terms of solution quality. It is possible that it only improves the quality of the initial population but it impacts negatively the evolution, especially since the population is less diverse. Table 4.13 shows the gap to the best known solution of the best solution found among the 5 runs and the average of the 5 runs. The best solution found and best mean value for each instance are highlighted in bold.

Table 4.13: Performance of the large-scale heuristics.

			$p_M = 0$ $f_{HD} = 0$		$p_M = 0$ $f_{HD} = 0.20$		$p_M = 0.25$ $f_{HD} = 0$		$p_M = 0.25$ $f_{HD} = 0.20$	
Instance	n	BKS	Best	Mean	Best	Mean	Best	Mean	Best	Mean
egl-s4-C	190	20461	1.89 %	2.24 %	1.63 %	2.13 %	1.67 %	1.99 %	**1.57** %	**1.74** %
egl-g2-E	375	1602229	4.89 %	5.55 %	4.96 %	5.34 %	4.75 %	5.59 %	**4.56** %	**5.05** %
Lpr-c-04	504	169254	1.41 %	1.60 %	0.64 %	0.84 %	0.91 %	1.08 %	**0.61** %	**0.82** %
Hefei-6	727	1068035	5.60 %	7.35 %	6.24 %	6.42 %	5.53 %	6.27 %	**4.82** %	**5.32** %
F1_g-4	780	768209	-4.29 %	-4.07 %	-5.00 %	-4.64 %	-4.69 %	-4.45 %	**-5.30** %	**-4.73** %
Beijing-3	1075	1534878	24.87 %	28.17 %	12.09 %	15.70 %	7.04 %	7.95 %	**6.07** %	**7.40** %

Without the large-scale heuristics, the performance is worse across all instances, and for larger instances it is significantly worse, showing the importance of the large-scale heuristics in MADCoM. Although the initialization is beneficial, especially for larger instances, it can also be observed that between initialization and mutation, the mutation is more valuable. Combining the two shows the best results.

To analyse the performance of local search, we ran MADCoM without local search. The repair operator, which is essentially local search, was replaced by a Split procedure with $Q_{max} = Q$, which always results in a feasible solution. The repair probability was set to $p_R = 0.5$ and the values of the remaining parameters are the same as in MADCoM. We also compare with a genetic algorithm without local search and without the large-scale heuristics and with the previous version with just local search. The results are available in Table 4.14, which has the same table format as Table 4.13.

Table 4.14: Performance of local search.

			Without Heuristics Without Local Search		With Heuristics Without Local Search		Without Heuristics With Local Search		With Heuristics With Local Search	
Instance	n	BKS	Best	Mean	Best	Mean	Best	Mean	Best	Mean
egl-s4-C	190	20461	28.93 %	30.32 %	4.88 %	5.90 %	1.89 %	2.24 %	**1.57** %	**1.74** %
egl-g2-E	375	1602229	77.21 %	79.23 %	11.34 %	12.28 %	4.89 %	5.55 %	**4.56** %	**5.05** %
Lpr-c-04	504	169254	26.48 %	27.39 %	2.85 %	2.95 %	1.41 %	1.60 %	**0.61** %	**0.82** %
Hefei-6	727	1068035	234.09 %	242.61 %	14.75 %	15.36 %	5.60 %	7.35 %	**4.82** %	**5.32** %
F1_g-4	780	768209	72.94 %	73.34 %	5.85 %	6.71 %	-4.29 %	-4.07 %	**-5.30** %	**-4.73** %
Beijing-3	1075	1534878	791.84 %	803.44 %	20.57 %	21.55 %	24.87 %	28.17 %	**6.07** %	**7.40** %

The simpler version with neither heuristics nor local search performs significantly worse, regardless of the problem size. On all but the largest instance, the version with just local search performs better than the version with just heuristics. Looking at figure 4.4, we can observe that for instance Beijing-3, the initial solutions generated using HD are of better than quality than random solutions improved by local search, which could explain this result. Once again, combining local search and the large-scale heuristics leads to the best results.

Chapter 5

Conclusions

5.1 Concluding Remarks

The main objective was accomplished as MADCoM is competitive on solving MCARP. We found new best solutions for 8 MCARP benchmark instances, 2 of which are optimal, and new best solutions on all the instances of the KW benchmark set. On the classical benchmarks, MADCoM performs on average at most 0.502 % than the best algorithm. On the large scale instances, although our algorithm outperforms RDG-MAENS on several instances, further work is required to achieve the best known solutions. The comparisons with state-of-the-art are not straightforward, as they have been implemented in a different programming language, ran on different processors and with varied termination conditions.

We show that the novel mutation operator is beneficial, increasing the performance significantly, especially for larger instances. Coupling it with local search is essential to achieve the best performance, as the mutation operator identifies the promising parts of the solution based on their relative quality and not their absolute quality. Therefore, if the solutions being mutated do not contain useful patterns, the algorithm will be slower at finding quality solutions.

We also demonstrate that Hierarchical Decomposition can be used as an initialization method, generating a diverse population with quality solutions. When combined with local search it outperforms random solutions improved by local search on both time and quality. Furthermore, our technique that limits the maximum number of clusters based on the problem size reduces the computational effort without impacting solution quality.

5.2 Future Work

The performance of MADCoM could be improved in several ways. First of all, further exploration of parameters values is necessary, especially the influence of the parameters of HD and RCO during the evolution. Secondly, it was observed that the local search is not as effective for problems with a larger number of vehicles, indicating a weakness with inter-route moves. A neighborhood move with larger step size, such as ejection chains [54], could help mitigate this. Lastly, MADCoM would benefit from a faster implementation in C++. It would also allow for the direct integration with the source code of UHGS and Route Cutting Off Decomposition and result in a fairer comparison between both methods and MADCoM.

Another next step would be to generalize MADCoM to solve other CARP variants for which a giant tour representation can be used, such as heterogeneous fleets [44] and multiple depots [45]. This generalization can be accomplished without changing the mutation operator or the initialization method, since both use a giant tour representation. It would only require adapting the Split procedure and local search to the specific variant.

Bibliography

[1] United Nations. *The World's Cities in 2018: Data Booklet*. Statistical Papers - United Nations (Ser. A), Population and Vital Statistics Report. UN, Dec. 2018. ISBN 978-92-1-047610-2. doi: 10.18356/c93f4dc6-en. URL https://www.un-ilibrary.org/content/books/9789210476102.

[2] K. C. Seto, B. Güneralp, and L. R. Hutyra. Global forecasts of urban expansion to 2030 and direct impacts on biodiversity and carbon pools. *Proceedings of the National Academy of Sciences*, 109(40):16083–16088, Oct. 2012. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.1211658109. URL https://www.pnas.org/content/109/40/16083. Publisher: National Academy of Sciences Section: Social Sciences.

[3] B. Esmaeilian, B. Wang, K. Lewis, F. Duarte, C. Ratti, and S. Behdad. The future of waste management in smart and sustainable cities: A review and concept paper. *Waste Management*, 81:177–195, Nov. 2018. ISSN 0956-053X. doi: 10.1016/j.wasman.2018.09.047. URL https://www.sciencedirect.com/science/article/pii/S0956053X18305865.

[4] L. Kiilerich and S. Wøhlk. New large-scale data instances for CARP and new variations of CARP. *INFOR: Information Systems and Operational Research*, 56(1):1–32, Jan. 2018. ISSN 0315-5986. doi: 10.1080/03155986.2017.1303960. URL https://doi.org/10.1080/03155986.2017.1303960. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/03155986.2017.1303960.

[5] S. Wøhlk. A fast heuristic for large-scale capacitated arc routing problems. *Journal of the Operational Research Society*, 69, July 2018. doi: 10.1080/01605682.2017.1415648.

[6] S.-H. Huang and P.-C. Lin. Multi-treatment capacitated arc routing of construction machinery in Taiwan's smooth road project. *Automation in Construction*, 21:210–218, Jan. 2012. ISSN 0926-5805. doi: 10.1016/j.autcon.2011.06.005. URL https://www.sciencedirect.com/science/article/pii/S0926580511001178.

[7] R. W. Eglese. Routeing winter gritting vehicles. *Discrete Applied Mathematics*, 48(3):231–244, Feb. 1994. ISSN 0166-218X. doi: 10.1016/0166-218X(92)00003-5. URL https://www.sciencedirect.com/science/article/pii/0166218X92000035.

[8] B. L. Golden and R. T. Wong. Capacitated arc routing problems. *Networks*, 11 (3):305–315, 1981. ISSN 1097-0037. doi: https://doi.org/10.1002/net.3230110308.

URL https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230110308. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230110308.

[9] P. Vansteenwegen, W. Souffriau, and K. Sörensen. Solving the mobile mapping van problem: A hybrid metaheuristic for capacitated arc routing with soft time windows. *Computers and Operations Research*, 37(11):1870–1876, Nov. 2010. ISSN 0305-0548. doi: 10.1016/j.cor.2009.05.006. URL https://doi.org/10.1016/j.cor.2009.05.006.

[10] R. Eglese, B. Golden, and E. Wasil. Chapter 13: Route Optimization for Meter Reading and Salt Spreading. In *Arc Routing*, MOS-SIAM Series on Optimization, pages 303–320. Society for Industrial and Applied Mathematics, Feb. 2015. ISBN 978-1-61197-366-2. doi: 10.1137/1.9781611973679.ch13. URL https://locus.siam.org/doi/abs/10.1137/1.9781611973679.ch13.

[11] J.-M. Belenguer, E. Benavent, P. Lacomme, and C. Prins. Lower and upper bounds for the mixed capacitated arc routing problem. *Computers and Operations Research*, 33(12):3363–3383, Dec. 2006. ISSN 0305-0548. doi: 10.1016/j.cor.2005.02.009. URL https://doi.org/10.1016/j.cor.2005.02.009.

[12] M. Constantino, L. Gouveia, M. Mourão, and A. C. Nunes. The mixed capacitated arc routing problem with non-overlapping routes. *European Journal of Operational Research*, 244(2):445–456, July 2015. ISSN 0377-2217. doi: 10.1016/j.ejor.2015.01.042. URL https://www.sciencedirect.com/science/article/pii/S0377221715000624.

[13] R. Baldacci, M. Battarra, and D. Vigo. Routing a Heterogeneous Fleet of Vehicles. In B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, Operations Research/Computer Science Interfaces, pages 3–27. Springer US, Boston, MA, 2008. ISBN 978-0-387-77778-8. doi: 10.1007/978-0-387-77778-8_1. URL https://doi.org/10.1007/978-0-387-77778-8_1.

[14] Y. Zhang, Y. Mei, S. Huang, X. Zheng, and C. Zhang. A Route Clustering and Search Heuristic for Large-Scale Multidepot-Capacitated Arc Routing Problem. *IEEE Transactions on Cybernetics*, PP: 1–14, Feb. 2021. doi: 10.1109/TCYB.2020.3043265.

[15] F. Chu, N. Labadie, and C. Prins. A Scatter Search for the Periodic Capacitated Arc Routing Problem. *European Journal of Operational Research*, 169:586–605, Feb. 2006. doi: 10.1016/j.ejor.2004.08.017.

[16] G. Fleury, P. Lacomme, C. Prins, and W. Ramdane-Chérif. Improving robustness of solutions to arc routing problems. *Journal of the Operational Research Society*, 56(5):526–538, May 2005. ISSN 0160-5682. doi: 10.1057/palgrave.jors.2601822. URL https://doi.org/10.1057/palgrave.jors.2601822. Publisher: Taylor & Francis _eprint: https://doi.org/10.1057/palgrave.jors.2601822.

[17] B. L. Golden, J. S. Dearmon, and E. K. Baker. Computational experiments with algorithms for a class of routing problems. *Computers & Operations Research*, 10(1):47–59, Jan. 1983. ISSN 0305-0548. doi: 10.1016/0305-0548(83)90026-6. URL http://www.sciencedirect.com/science/article/pii/0305054883900266.

[18] G. Clarke and J. W. Wright. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research*, 12(4):568–581, Aug. 1964. ISSN 0030-364X. doi: 10.1287/opre.12.4.568. URL https://pubsonline.informs.org/doi/abs/10.1287/opre.12.4.568. Publisher: INFORMS.

[19] C. Prins. Chapter 7: The Capacitated Arc Routing Problem: Heuristics. In *Arc Routing*, MOS-SIAM Series on Optimization, pages 131–157. Society for Industrial and Applied Mathematics, Feb. 2015. ISBN 978-1-61197-366-2. doi: 10.1137/1.9781611973679.ch7. URL https://locus.siam.org/doi/abs/10.1137/1.9781611973679.ch7.

[20] G. Ulusoy. The fleet size and mix problem for capacitated arc routing. *European Journal of Operational Research*, 22:329–337, Feb. 1985. doi: 10.1016/0377-2217(85)90252-8.

[21] W.-L. Pearn, A. Assad, and B. L. Golden. Transforming arc routing into node routing problems. *Computers & Operations Research*, 14(4):285–288, Jan. 1987. ISSN 0305-0548. doi: 10.1016/0305-0548(87)90065-7. URL https://www.sciencedirect.com/science/article/pii/0305054887900657.

[22] R. Baldacci and V. Maniezzo. Exact methods based on node-routing formulations for undirected arc-routing problems. *Networks*, 47(1):52–60, 2006. ISSN 1097-0037. doi: 10.1002/net.20091. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/net.20091. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.20091.

[23] H. Longo, M. P. de Aragão, and E. Uchoa. Solving capacitated arc routing problems using a transformation to the CVRP. *Computers & Operations Research*, 33(6):1823–1837, June 2006. ISSN 0305-0548. doi: 10.1016/j.cor.2004.11.020. URL https://www.sciencedirect.com/science/article/pii/S0305054804003223.

[24] J. M. Belenguer and E. Benavent. A cutting plane algorithm for the capacitated arc routing problem. *Computers & Operations Research*, 30(5):705–728, Apr. 2003. ISSN 0305-0548. doi: 10.1016/S0305-0548(02)00046-1. URL https://www.sciencedirect.com/science/article/pii/S0305054802000461.

[25] C. Bode and S. Irnich. Cut-First Branch-and-Price-Second for the Capacitated Arc-Routing Problem. *Operations Research*, 60(5):1167–1182, Oct. 2012. ISSN 0030-364X. doi: 10.1287/opre.1120.1079. URL https://pubsonline.informs.org/doi/abs/10.1287/opre.1120.1079. Publisher: INFORMS.

[26] J. M. Belenguer, E. Benavent, and S. Irnich. Chapter 9: The Capacitated Arc Routing Problem: Exact Algorithms. In *Arc Routing*, MOS-SIAM Series on Optimization, pages 183–221. Society

for Industrial and Applied Mathematics, Feb. 2015. ISBN 978-1-61197-366-2. doi: 10.1137/1.9781611973679.ch9. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611973679.ch9.

[27] L. Gouveia, M. C. Mourão, and L. S. Pinto. Lower bounds for the mixed capacitated arc routing problem. *Computers & Operations Research*, 37(4):692–699, Apr. 2010. ISSN 0305-0548. doi: 10.1016/j.cor.2009.06.018. URL https://www.sciencedirect.com/science/article/pii/S0305054809001737.

[28] F. W. Glover and G. A. Kochenberger, editors. *Handbook of Metaheuristics*. International Series in Operations Research & Management Science. Springer US, 2003. ISBN 978-0-306-48056-0. doi: 10.1007/b101874. URL https://www.springer.com/gp/book/9780306480560.

[29] R. W. Eglese and L. Y. O. Li. A Tabu Search based Heuristic for Arc Routing with a Capacity Constraint and Time Deadline. In I. H. Osman and J. P. Kelly, editors, *Meta-Heuristics: Theory and Applications*, pages 633–649. Springer US, Boston, MA, 1996. ISBN 978-1-4613-1361-8. doi: 10.1007/978-1-4613-1361-8_38. URL https://doi.org/10.1007/978-1-4613-1361-8_38.

[30] A. Hertz, G. Laporte, and M. Mittaz. A Tabu Search Heuristic for the Capacitated arc Routing Problem. *Operations Research*, 48(1):129–135, Feb. 2000. ISSN 0030-364X. doi: 10.1287/opre.48.1.129.12455. URL https://pubsonline.informs.org/doi/abs/10.1287/opre.48.1.129.12455. Publisher: INFORMS.

[31] M. Polacek, K. Doerner, R. Hartl, and V. Maniezzo. A Variable Neighborhood Search for the Capacitated Arc Routing Problem with Intermediate Facilities. *Journal of Heuristics*, 14, Oct. 2008. doi: 10.1007/s10732-007-9050-2.

[32] J. Brandão and R. Eglese. A deterministic tabu search algorithm for the capacitated arc routing problem. *Computers & Operations Research*, 35(4):1112–1126, Apr. 2008. ISSN 0305-0548. doi: 10.1016/j.cor.2006.07.007. URL http://www.sciencedirect.com/science/article/pii/S0305054806001535.

[33] S. Wøhlk. *Contributions to Arc Routing*. PhD thesis, University of Southern Denmark, 2006.

[34] P. Beullens, L. Muyldermans, D. Cattrysse, and D. Van Oudheusden. A guided local search heuristic for the capacitated arc routing problem. *European Journal of Operational Research*, 147(3):629–643, June 2003. ISSN 0377-2217. doi: 10.1016/S0377-2217(02)00334-X. URL https://www.sciencedirect.com/science/article/pii/S037722170200334X.

[35] L. Santos, J. Coutinho-Rodrigues, and J. Current. An improved ant colony optimization based algorithm for the capacitated arc routing problem. *Transportation Research Part B: Methodological*, 44:246–266, Feb. 2010. doi: 10.1016/j.trb.2009.07.004.

[36] P. Lacomme, C. Prins, and W. Ramdane-Chérif. A Genetic Algorithm for the Capacitated Arc Routing Problem and Its Extensions. In E. J. W. Boers, editor, *Applications of Evolutionary Computing*,

Lecture Notes in Computer Science, pages 473–483, Berlin, Heidelberg, 2001. Springer. ISBN 978-3-540-45365-9. doi: 10.1007/3-540-45365-2_49.

[37] K. Tang, Y. Mei, and X. Yao. Memetic Algorithm With Extended Neighborhood Search for Capacitated Arc Routing Problems. *IEEE Transactions on Evolutionary Computation*, 13(5):1151–1166, Oct. 2009. ISSN 1941-0026. doi: 10.1109/TEVC.2009.2023449. Conference Name: IEEE Transactions on Evolutionary Computation.

[38] Y. Mei, X. Li, and X. Yao. Cooperative Coevolution With Route Distance Grouping for Large-Scale Capacitated Arc Routing Problems. *IEEE Transactions on Evolutionary Computation*, 18(3):435–449, June 2014. ISSN 1941-0026. doi: 10.1109/TEVC.2013.2281503. Conference Name: IEEE Transactions on Evolutionary Computation.

[39] K. Tang, J. Wang, X. Li, and X. Yao. A Scalable Approach to Capacitated Arc Routing Problems Based on Hierarchical Decomposition. *IEEE Transactions on Cybernetics*, PP:1–13, Aug. 2016. doi: 10.1109/TCYB.2016.2590558.

[40] Y. Zhang, Y. Mei, B. Zhang, and K. Jiang. Divide-and-conquer large scale capacitated arc routing problems with route cutting off decomposition. *Information Sciences*, 553:208–224, Apr. 2021. ISSN 0020-0255. doi: 10.1016/j.ins.2020.11.011. URL `https://www.sciencedirect.com/science/article/pii/S0020025520310975`.

[41] T. Vidal. Node, Edge, Arc Routing and Turn Penalties: Multiple Problems—One Neighborhood Extension. *Operations Research*, 65, May 2017. doi: 10.1287/opre.2017.1595.

[42] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 3 edition, July 2009. ISBN 978-0-262-03384-8.

[43] C. Prins, N. Labadie, and M. Reghioui. Tour splitting algorithms for vehicle routing problems. *International Journal of Production Research*, 47:507–535, Jan. 2009. doi: 10.1080/00207540802426599.

[44] C. Prins. Two memetic algorithms for heterogeneous fleet vehicle routing problems. *Engineering Applications of Artificial Intelligence*, 22(6):916–928, Sept. 2009. ISSN 0952-1976. doi: 10.1016/j.engappai.2008.10.006. URL `https://www.sciencedirect.com/science/article/pii/S0952197608001693`.

[45] T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei. A Hybrid Genetic Algorithm for Multidepot and Periodic Vehicle Routing Problems. *Operations Research*, 60:611–624, June 2012. doi: 10.1287/opre.1120.1048.

[46] E. Schubert and P. J. Rousseeuw. Fast and eager k -medoids clustering: O (k) runtime improvement of the PAM, CLARA, and CLARANS algorithms. *Information Systems*, 101:101804, Nov. 2021. ISSN 03064379. doi: 10.1016/j.is.2021.101804. URL `https://linkinghub.elsevier.com/retrieve/pii/S0306437921000557`.

[47] F.-A. Fortin, F.-M. D. Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, 13(70):2171–2175, 2012. ISSN 1533-7928. URL `http://jmlr.org/papers/v13/fortin12a.html`.

[48] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring Network Structure, Dynamics, and Function using NetworkX. In G. Varoquaux, T. Vaught, and J. Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[49] E. Benavent, V. Campos, A. Corberan, and E. Mota. The Capacitated Arc Routing Problem: Lower bounds. *Networks*, 22(7):669–690, 1992. ISSN 1097-0037. doi: 10.1002/net.3230220706. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230220706`. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230220706.

[50] L. Y. O. Li and R. W. Eglese. An Interactive Algorithm for Vehicle Routeing for Winter - Gritting. *The Journal of the Operational Research Society*, 47(2):217–228, 1996. ISSN 0160-5682. doi: 10.2307/2584343. URL `https://www.jstor.org/stable/2584343`. Publisher: Palgrave Macmillan Journals.

[51] C. Huang, Y. Li, and X. Yao. A Survey of Automatic Parameter Tuning Methods for Metaheuristics. *IEEE Transactions on Evolutionary Computation*, 24(2):201–216, Apr. 2020. ISSN 1941-0026. doi: 10.1109/TEVC.2019.2921598. Conference Name: IEEE Transactions on Evolutionary Computation.

[52] M. Birattari. *Tuning Metaheuristics: A Machine Learning Perspective*. Studies in Computational Intelligence. Springer-Verlag, Berlin Heidelberg, 2009. ISBN 978-3-642-00482-7. doi: 10.1007/978-3-642-00483-4. URL `https://www.springer.com/gp/book/9783642004827`.

[53] F. Nogueira. Bayesian Optimization: Open source constrained global optimization tool for Python, 2014. URL `https://github.com/fmfn/BayesianOptimization`.

[54] F. Glover and C. Rego. Ejection chain and filter-and-fan methods in combinatorial optimization. *4OR*, 4(4):263–296, Dec. 2006. ISSN 1614-2411. doi: 10.1007/s10288-006-0029-x. URL `https://doi.org/10.1007/s10288-006-0029-x`.

Appendix A

Flowcharts

A.1 MADCoM

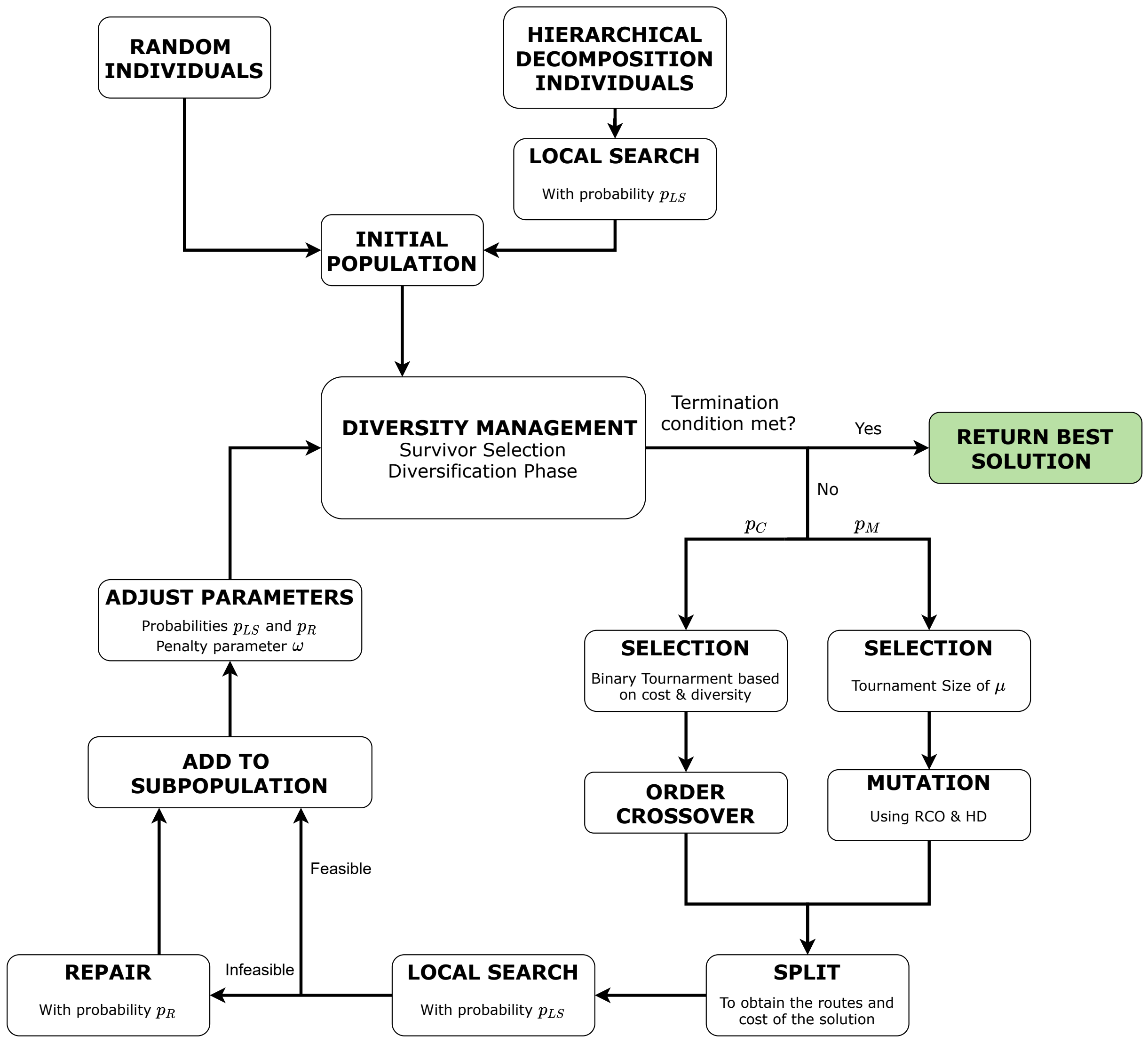


Figure A.1: Flowchart of MADCoM.

Appendix B

Extended Results on Classical Benchmarks

B.1 Table Format

In each table, n denotes the problem size and BKS the best known solution for each instance. The BKS is accompanied by an asterisk if it is a proven optimal solution. In the tables of the benchmark sets GDB and VAL, all instances have been solved to optimality, therefore BKS is replaced with Optimal, which stands for the cost of the optimal solution. In the table of the benchmark set MVAL, we include the lower bounds reported by Gouveia et al. [27], to prove that it is in fact the optimal solution. For each comparing algorithm, Best denotes the cost of the best solution found across all runs and Mean the average cost of the best solution found in each run. For MADCoM, we also report the percentage gap to the best known solution of both the best solution found and the mean. A value is in bold if it is equal to the minimal cost among all algorithms, and it is underlined if it is larger than the corresponding value of MADCoM.

B.2 Comparison Algorithms

The results of MADCoM are compared with several algorithms in the literature. The acronyms and references for each one are available in table B.1

Table B.1: Algorithms for comparison in the classical benchmarks.

Acronym	Name
UHGS	Unified Hybrid Genetic Search [41]
MAENS	Memetic Algorithm with Extended Neighborhood Search [37]
VNS	Variable Neighborhood Search [31]
MABBLC	Memetic Algorithm by Belenguer et. al [11]

B.3 GDB

Table B.2: Results on the GDB benchmark set.

							MADCoM			
			MAENS		UHGS		Best		Mean	
Instance	n	Optimal	Best	Mean	Best	Mean	Cost	Gap	Cost	Gap
gdb1	22	316	**316**	**316**	**316**	**316**	**316**	0.00 %	**316**	0.00 %
gdb2	26	339	**339**	**339**	**339**	**339**	345	1.77 %	345	1.77 %
gdb3	22	275	**275**	**275**	**275**	**275**	**275**	0.00 %	**275**	0.00 %
gdb4	19	287	**287**	**287**	**287**	**287**	**287**	0.00 %	**287**	0.00 %
gdb5	26	377	**377**	**377**	**377**	**377**	**377**	0.00 %	381	1.06 %
gdb6	22	298	**298**	**298**	**298**	**298**	**298**	0.00 %	**298**	0.00 %
gdb7	22	325	**325**	**325**	**325**	**325**	**325**	0.00 %	**325**	0.00 %
gdb8	46	348	**348**	<u>349</u>	**348**	**348**	**348**	0.00 %	**348**	0.00 %
gdb9	51	303	**303**	**303**	**303**	**303**	**303**	0.00 %	**303**	0.00 %
gdb10	25	275	**275**	**275**	**275**	**275**	**275**	0.00 %	**275**	0.00 %
gdb11	45	395	**395**	**395**	**395**	**395**	**395**	0.00 %	**395**	0.00 %
gdb12	23	458	**458**	**458**	**458**	**458**	**458**	0.00 %	**458**	0.00 %
gdb13	28	536	**536**	**536**	**536**	**536**	**536**	0.00 %	**536**	0.00 %
gdb14	21	100	**100**	**100**	**100**	**100**	**100**	0.00 %	**100**	0.00 %
gdb15	21	58	**58**	**58**	**58**	**58**	**58**	0.00 %	**58**	0.00 %
gdb16	28	127	**127**	**127**	**127**	**127**	**127**	0.00 %	**127**	0.00 %
gdb17	28	91	**91**	**91**	**91**	**91**	**91**	0.00 %	**91**	0.00 %
gdb18	36	164	**164**	**164**	**164**	**164**	**164**	0.00 %	**164**	0.00 %
gdb19	11	55	**55**	**55**	**55**	**55**	**55**	0.00 %	**55**	0.00 %
gdb20	22	121	**121**	**121**	**121**	**121**	**121**	0.00 %	**121**	0.00 %
gdb21	33	156	**156**	**156**	**156**	**156**	**156**	0.00 %	**156**	0.00 %
gdb22	44	200	**200**	**200**	**200**	**200**	**200**	0.00 %	**200**	0.00 %
gdb23	55	233	**233**	**233**	**233**	**233**	**233**	0.00 %	**233**	0.00 %

B.4 VAL

Table B.3: Results on the VAL benchmark set.

					MADCoM			
			UHGS		Best		Mean	
Instance	n	Optimal	Best	Mean	Cost	Gap	Cost	Gap
1A	39	173	**173**	**173**	**173**	0.00 %	**173**	0.00 %
1B	39	173	**173**	**173**	**173**	0.00 %	**173**	0.00 %
1C	39	245	**245**	**245**	**245**	0.00 %	**245**	0.00 %
2A	34	227	**227**	**227**	**227**	0.00 %	**227**	0.00 %
2B	34	259	**259**	**259**	**259**	0.00 %	**259**	0.00 %
2C	34	457	**457**	**457**	**457**	0.00 %	**457**	0.00 %
3A	35	81	**81**	**81**	**81**	0.00 %	**81**	0.00 %
3B	35	87	**87**	**87**	**87**	0.00 %	**87**	0.00 %
3C	35	138	**138**	**138**	**138**	0.00 %	**138**	0.00 %
4A	69	400	**400**	**400**	**400**	0.00 %	**400**	0.00 %
4B	69	412	**412**	**412**	**412**	0.00 %	**412**	0.00 %
4C	69	428	**428**	**428**	**428**	0.00 %	**428**	0.00 %
4D	69	528	**528**	530	530	0.38 %	530	0.38 %
5A	65	423	**423**	**423**	**423**	0.00 %	**423**	0.00 %
5B	65	446	**446**	**446**	**446**	0.00 %	**446**	0.00 %
5C	65	474	**474**	**474**	**474**	0.00 %	**474**	0.00 %
5D	65	575	**575**	576	**575**	0.00 %	577	0.35 %
6A	50	223	**223**	**223**	**223**	0.00 %	**223**	0.00 %
6B	50	233	**233**	**233**	**233**	0.00 %	**233**	0.00 %
6C	50	317	**317**	**317**	**317**	0.00 %	**317**	0.00 %
7A	66	279	**279**	**279**	**279**	0.00 %	**279**	0.00 %
7B	66	283	**283**	**283**	**283**	0.00 %	**283**	0.00 %
7C	66	334	**334**	**334**	**334**	0.00 %	**334**	0.00 %
8A	63	386	**386**	**386**	**386**	0.00 %	**386**	0.00 %
8B	63	395	**395**	**395**	**395**	0.00 %	**395**	0.00 %
8C	63	521	**521**	**521**	**521**	0.00 %	**521**	0.00 %
9A	92	323	**323**	**323**	**323**	0.00 %	**323**	0.00 %
9B	92	326	**326**	**326**	**326**	0.00 %	**326**	0.00 %
9C	92	332	**332**	**332**	**332**	0.00 %	**332**	0.00 %
9D	92	388	**389**	391	391	0.77 %	391	0.77 %
10A	97	428	**428**	**428**	**428**	0.00 %	**428**	0.00 %
10B	97	436	**436**	**436**	**436**	0.00 %	437	0.23 %
10C	97	446	**446**	**446**	**446**	0.00 %	**446**	0.00 %
10D	97	525	**526**	**526**	528	0.57 %	529	0.76 %

B.5 BMCV

Table B.4: Results on the C instances of the BMCV benchmark set.

					MADCoM			
			UHGS		Best		Mean	
Instance	n	BKS	Best	Mean	Cost	Gap	Cost	Gap
C01	79	*4150	**4150**	**4150**	**4150**	0.00 %	4160	0.24 %
C02	53	*3135	**3135**	**3135**	**3135**	0.00 %	**3135**	0.00 %
C03	51	*2575	**2575**	**2575**	**2575**	0.00 %	**2575**	0.00 %
C04	72	*3510	**3510**	**3510**	**3510**	0.00 %	**3510**	0.00 %
C05	65	*5365	**5365**	**5365**	**5365**	0.00 %	5369	0.07 %
C06	51	*2535	**2535**	**2535**	**2535**	0.00 %	**2535**	0.00 %
C07	52	*4075	**4075**	**4075**	**4075**	0.00 %	**4075**	0.00 %
C08	63	*4090	**4090**	**4090**	**4090**	0.00 %	**4090**	0.00 %
C09	97	5260	**5260**	**5260**	**5260**	0.00 %	5274	0.27 %
C10	55	*4700	**4700**	**4700**	**4700**	0.00 %	4737	0.79 %
C11	94	4630	**4630**	4636	4640	0.22 %	4645	0.32 %
C12	72	*4240	**4240**	**4240**	**4240**	0.00 %	**4240**	0.00 %
C13	52	*2955	**2955**	**2955**	**2955**	0.00 %	**2955**	0.00 %
C14	57	*4030	**4030**	**4030**	**4030**	0.00 %	**4030**	0.00 %
C15	107	4940	**4940**	**4940**	4965	0.51 %	4986	0.93 %
C16	32	*1475	**1475**	**1475**	**1475**	0.00 %	1478	0.20 %
C17	42	*3555	**3555**	**3555**	**3555**	0.00 %	**3555**	0.00 %
C18	121	5605	**5620**	5626	5640	0.62 %	5649	0.79 %
C19	61	*3115	**3115**	**3115**	**3115**	0.00 %	3119	0.13 %
C20	53	*2120	**2120**	**2120**	**2120**	0.00 %	**2120**	0.00 %
C21	76	*3970	**3970**	**3970**	**3970**	0.00 %	**3970**	0.00 %
C22	43	*2245	**2245**	**2245**	**2245**	0.00 %	**2245**	0.00 %
C23	92	4085	**4085**	**4085**	**4085**	0.00 %	4093	0.20 %
C24	84	*3400	**3400**	**3400**	**3400**	0.00 %	3402	0.06 %
C25	38	*2310	**2310**	**2310**	**2310**	0.00 %	**2310**	0.00 %

Table B.5: Results on the D instances of the BMCV benchmark set.

					MADCoM			
			UHGS		Best		Mean	
Instance	n	BKS	Best	Mean	Cost	Gap	Cost	Gap
D01	79	*3215	**3215**	3218	3230	0.47 %	3233	0.56 %
D02	53	*2520	**2520**	**2520**	**2520**	0.00 %	**2520**	0.00 %
D03	51	*2065	**2065**	**2065**	**2065**	0.00 %	**2065**	0.00 %
D04	72	*2785	**2785**	**2785**	**2785**	0.00 %	**2785**	0.00 %
D05	65	*3935	**3935**	**3935**	**3935**	0.00 %	**3935**	0.00 %
D06	51	*2125	**2125**	**2125**	**2125**	0.00 %	**2125**	0.00 %
D07	52	*3115	**3115**	**3115**	**3115**	0.00 %	**3115**	0.00 %
D08	63	*3045	**3045**	**3045**	**3045**	0.00 %	**3045**	0.00 %
D09	97	*4120	**4120**	**4120**	**4120**	0.00 %	**4120**	0.00 %
D10	55	*3340	**3340**	**3340**	**3340**	0.00 %	**3340**	0.00 %
D11	94	*3745	**3745**	**3745**	**3745**	0.00 %	3754	0.24 %
D12	72	*3310	**3310**	**3310**	**3310**	0.00 %	**3310**	0.00 %
D13	52	*2535	**2535**	**2535**	**2535**	0.00 %	2536	0.04 %
D14	57	*3280	**3280**	**3280**	**3280**	0.00 %	**3280**	0.00 %
D15	107	*3990	**3990**	**3990**	**3990**	0.00 %	3996	0.15 %
D16	32	*1060	**1060**	**1060**	**1060**	0.00 %	**1060**	0.00 %
D17	42	*2620	**2620**	**2620**	**2620**	0.00 %	**2620**	0.00 %
D18	121	*4165	**4165**	**4165**	**4165**	0.00 %	4167	0.05 %
D19	61	*2400	**2400**	**2400**	**2400**	0.00 %	**2400**	0.00 %
D20	53	*1870	**1870**	**1870**	**1870**	0.00 %	**1870**	0.00 %
D21	76	3050	**3050**	**3050**	**3050**	0.00 %	3055	0.16 %
D22	43	*1865	**1865**	**1865**	**1865**	0.00 %	**1865**	0.00 %
D23	92	3130	**3130**	**3130**	**3130**	0.00 %	**3130**	0.00 %
D24	84	*2710	**2710**	**2710**	**2710**	0.00 %	**2710**	0.00 %
D25	38	*1815	**1815**	**1815**	**1815**	0.00 %	**1815**	0.00 %

Table B.6: Results on the E instances of the BMCV benchmark set.

					MADCoM			
			UHGS		Best		Mean	
Instance	n	BKS	Best	Mean	Cost	Gap	Cost	Gap
E01	85	4910	**4910**	**4910**	**4910**	0.00 %	**4910**	0.00 %
E02	58	*3990	**3990**	**3990**	**3990**	0.00 %	**3990**	0.00 %
E03	47	*2015	**2015**	**2015**	2025	0.50 %	2025	0.50 %
E04	77	*4155	**4155**	**4155**	**4155**	0.00 %	**4155**	0.00 %
E05	61	*4585	**4585**	**4585**	**4585**	0.00 %	4674	1.94 %
E06	43	*2055	**2055**	**2055**	**2055**	0.00 %	**2055**	0.00 %
E07	50	*4155	**4155**	**4155**	**4155**	0.00 %	**4155**	0.00 %
E08	59	*4710	**4710**	**4710**	**4710**	0.00 %	**4710**	0.00 %
E09	103	5810	**5810**	**5810**	**5810**	0.00 %	5858	0.83 %
E10	49	*3605	**3605**	**3605**	**3605**	0.00 %	**3605**	0.00 %
E11	94	*4650	**4650**	4655	4670	0.43 %	4670	0.43 %
E12	67	*4180	**4180**	4190	4190	0.24 %	4194	0.33 %
E13	52	*3345	**3345**	**3345**	**3345**	0.00 %	**3345**	0.00 %
E14	55	*4115	**4115**	**4115**	**4115**	0.00 %	**4115**	0.00 %
E15	107	*4205	**4205**	4219	4225	0.48 %	4225	0.48 %
E16	54	*3775	**3775**	**3775**	**3775**	0.00 %	3783	0.21 %
E17	36	*2740	**2740**	**2740**	**2740**	0.00 %	**2740**	0.00 %
E18	88	3835	**3835**	**3835**	**3835**	0.00 %	**3835**	0.00 %
E19	66	*3235	**3235**	**3235**	**3235**	0.00 %	**3235**	0.00 %
E20	63	*2825	**2825**	**2825**	**2825**	0.00 %	**2825**	0.00 %
E21	72	*3730	**3730**	**3730**	**3730**	0.00 %	3732	0.05 %
E22	44	*2470	**2470**	**2470**	**2470**	0.00 %	**2470**	0.00 %
E23	89	3710	**3710**	3713	**3710**	0.00 %	3733	0.62 %
E24	86	*4020	**4020**	**4020**	**4020**	0.00 %	**4020**	0.00 %
E25	28	*1615	**1615**	**1615**	**1615**	0.00 %	**1615**	0.00 %

Table B.7: Results on the F instances of the BMCV benchmark set.

					MADCoM			
			UHGS		Best		Mean	
Instance	n	BKS	Best	Mean	Cost	Gap	Cost	Gap
F01	85	*4040	**4040**	**4040**	**4040**	0.00 %	**4040**	0.00 %
F02	58	*3300	**3300**	**3300**	**3300**	0.00 %	**3300**	0.00 %
F03	47	*1665	**1665**	**1665**	**1665**	0.00 %	**1665**	0.00 %
F04	77	*3485	**3485**	**3485**	**3485**	0.00 %	3490	0.14 %
F05	61	*3605	**3605**	**3605**	**3605**	0.00 %	**3605**	0.00 %
F06	43	*1875	**1875**	**1875**	**1875**	0.00 %	**1875**	0.00 %
F07	50	*3335	**3335**	**3335**	**3335**	0.00 %	**3335**	0.00 %
F08	59	*3705	**3705**	**3705**	**3705**	0.00 %	**3705**	0.00 %
F09	103	*4730	**4730**	**4730**	**4730**	0.00 %	**4730**	0.00 %
F10	49	*2925	**2925**	**2925**	**2925**	0.00 %	**2925**	0.00 %
F11	94	*3835	**3835**	**3835**	**3835**	0.00 %	**3835**	0.00 %
F12	67	*3395	**3395**	**3395**	**3395**	0.00 %	3408	0.38 %
F13	52	*2855	**2855**	**2855**	**2855**	0.00 %	**2855**	0.00 %
F14	55	*3330	**3330**	**3330**	**3330**	0.00 %	**3330**	0.00 %
F15	107	*3560	**3560**	**3560**	**3560**	0.00 %	**3560**	0.00 %
F16	54	*2725	**2725**	**2725**	**2725**	0.00 %	**2725**	0.00 %
F17	36	*2055	**2055**	**2055**	**2055**	0.00 %	**2055**	0.00 %
F18	88	3075	**3075**	**3075**	**3075**	0.00 %	**3075**	0.00 %
F19	66	2525	**2525**	**2525**	**2525**	0.00 %	**2525**	0.00 %
F20	63	*2445	**2445**	**2445**	**2445**	0.00 %	2447	0.08 %
F21	72	*2930	**2930**	**2930**	**2930**	0.00 %	**2930**	0.00 %
F22	44	*2075	**2075**	**2075**	**2075**	0.00 %	**2075**	0.00 %
F23	89	3005	**3005**	**3005**	**3005**	0.00 %	3008	0.10 %
F24	86	*3210	**3210**	**3210**	**3210**	0.00 %	3212	0.06 %
F25	28	*1390	**1390**	**1390**	**1390**	0.00 %	**1390**	0.00 %

B.6 EGLESE

Table B.8: Results on the EGLESE benchmark set.

									MADCoM			
			VNS		MAENS		UHGS		Best		Mean	
Instance	n	BKS	Best	Mean	Best	Mean	Best	Mean	Cost	Gap	Cost	Gap
egl-e1-A	51	*3548	**3548**	**3548**	**3548**	**3548**	**3548**	**3548**	**3548**	0.00 %	**3548**	0.00 %
egl-e1-B	51	*4498	**4498**	4522	**4498**	4516	**4498**	**4498**	**4498**	0.00 %	**4498**	0.00 %
egl-e1-C	51	*5595	**5595**	5608	**5595**	5602	**5595**	**5595**	**5595**	0.00 %	**5595**	0.00 %
egl-e2-A	72	*5018	**5018**	5024	**5018**	**5018**	**5018**	**5018**	**5018**	0.00 %	**5018**	0.00 %
egl-e2-B	72	6317	**6317**	6335	**6317**	6341	**6317**	6321	**6317**	0.00 %	6318	0.02 %
egl-e2-C	72	*8335	**8335**	8356	**8335**	8356	**8335**	**8335**	**8335**	0.00 %	**8335**	0.00 %
egl-e3-A	87	*5898	**5898**	**5898**	**5898**	5899	**5898**	**5898**	**5898**	0.00 %	**5898**	0.00 %
egl-e3-B	87	7775	**7775**	7806	**7775**	7803	**7775**	7776	7777	0.03 %	7779	0.05 %
egl-e3-C	87	10292	**10292**	10322	**10292**	10322	**10292**	**10292**	**10292**	0.00 %	10316	0.23 %
egl-e4-A	98	6444	6446	6459	6456	6475	**6444**	**6444**	6458	0.22 %	6462	0.28 %
egl-e4-B	98	8961	8996	9016	8998	9023	**8961**	8985	9012	0.57 %	9035	0.83 %
egl-e4-C	98	11529	11618	11750	11561	11646	**11529**	11563	11606	0.67 %	11627	0.85 %
egl-s1-A	75	*5018	**5018**	**5018**	**5018**	5040	**5018**	**5018**	**5018**	0.00 %	**5018**	0.00 %
egl-s1-B	75	*6388	**6388**	**6388**	**6388**	6433	**6388**	**6388**	**6388**	0.00 %	**6388**	0.00 %
egl-s1-C	75	*8518	**8518**	**8518**	**8518**	**8518**	**8518**	**8518**	**8518**	0.00 %	**8518**	0.00 %
egl-s2-A	147	9875	9895	9998	9895	9959	**9875**	9886	9927	0.53 %	9965	0.91 %
egl-s2-B	147	13057	13100	13176	13147	13232	**13081**	13102	13252	1.49 %	13301	1.87 %
egl-s2-C	147	*16425	**16425**	16552	16430	16510	**16425**	16440	16585	0.97 %	16600	1.07 %
egl-s3-A	159	10201	**10221**	10291	10257	10313	**10221**	10240	10297	0.94 %	10355	1.51 %
egl-s3-B	159	13682	**13682**	13829	13749	13877	**13682**	13694	13757	0.55 %	13836	1.13 %
egl-s3-C	159	*17188	17259	17328	17207	17306	**17188**	17191	17338	0.87 %	17374	1.08 %
egl-s4-A	190	12216	12292	12440	12341	12419	**12273**	12288	12390	1.42 %	12434	1.78 %
egl-s4-B	190	16214	16321	16410	16337	16441	**16230**	16284	16500	1.76 %	16543	2.03 %
egl-s4-C	190	20461	20582	20732	20538	20767	**20500**	20591	20782	1.57 %	20817	1.74 %

B.7 MVAL

Table B.9: Results on the MVAL benchmark set.

						MADCoM			
				MABBLC		Best		Mean	
Instance	n	LB	BKS	Best	Mean	Cost	Gap	Cost	Gap
mval1A	55	230	*230	**230**	**230**	**230**	0.00 %	**230**	0.00 %
mval1B	51	261	*261	**261**	**261**	**261**	0.00 %	**261**	0.00 %
mval1C	53	309	315	315	315	**309**	-1.90 %	310	-1.59 %
mval2A	44	324	*324	**324**	**324**	**324**	0.00 %	**324**	0.00 %
mval2B	52	395	*395	**395**	**395**	**395**	0.00 %	**395**	0.00 %
mval2C	49	521	526	**526**	**526**	**526**	0.00 %	**526**	0.00 %
mval3A	48	115	*115	**115**	**115**	**115**	0.00 %	**115**	0.00 %
mval3B	45	142	*142	**142**	**142**	**142**	0.00 %	**142**	0.00 %
mval3C	43	166	*166	**166**	**166**	**166**	0.00 %	**166**	0.00 %
mval4A	95	580	*580	**580**	**580**	**580**	0.00 %	**580**	0.00 %
mval4B	102	650	*650	**650**	**650**	**650**	0.00 %	**650**	0.00 %
mval4C	103	630	*630	**630**	631	**630**	0.00 %	**630**	0.00 %
mval4D	104	746	770	770	776	**750**	-2.60 %	752	-2.34 %
mval5A	96	597	*597	**597**	**597**	**597**	0.00 %	**597**	0.00 %
mval5B	91	613	*613	**613**	615	**613**	0.00 %	**613**	0.00 %
mval5C	98	697	*697	**697**	**697**	**697**	0.00 %	**697**	0.00 %
mval5D	92	719	739	739	757	**729**	-1.35 %	731	-1.08 %
mval6A	69	326	*326	**326**	**326**	**326**	0.00 %	**326**	0.00 %
mval6B	66	317	*317	**317**	**317**	**317**	0.00 %	**317**	0.00 %
mval6C	68	365	371	371	375	**370**	-0.27 %	371	0.00 %
mval7A	86	364	*364	**364**	**364**	**364**	0.00 %	**364**	0.00 %
mval7B	91	412	*412	**412**	**412**	**412**	0.00 %	**412**	0.00 %
mval7C	90	424	426	**426**	428	**426**	0.00 %	**426**	0.00 %
mval8A	96	581	*581	**581**	**581**	**581**	0.00 %	**581**	0.00 %
mval8B	91	531	*531	**531**	**531**	**531**	0.00 %	**531**	0.00 %
mval8C	83	617	638	638	638	**632**	-0.94 %	**632**	-0.94 %
mval9A	132	458	*458	**458**	**458**	**458**	0.00 %	**458**	0.00 %
mval9B	120	453	*453	**453**	**453**	**453**	0.00 %	**453**	0.00 %
mval9C	125	428	429	429	434	**428**	-0.23 %	429	0.00 %
mval9D	131	514	520	520	520	**519**	-0.19 %	**519**	-0.19 %
mval10A	138	634	*634	**634**	**634**	**634**	0.00 %	**634**	0.00 %
mval10B	134	661	*661	**661**	662	**661**	0.00 %	**661**	0.00 %
mval10C	136	623	*623	**623**	624	**623**	0.00 %	624	0.16 %
mval10D	129	643	649	649	650	**645**	-0.62 %	647	-0.31 %

B.8 LPR

Table B.10: Results on the LPR benchmark set.

					MADCoM			
			MABBLC		Best		Mean	
Instance	n	BKS	Best	Mean	Cost	Gap	Cost	Gap
Lpr-a-01	52	*13484	**13484**	**13484**	**13484**	0.00 %	**13484**	0.00 %
Lpr-a-02	104	*28052	**28052**	**28052**	**28052**	0.00 %	28063	0.04 %
Lpr-a-03	304	76155	**76155**	**76155**	76500	0.45 %	76614	0.60 %
Lpr-a-04	503	127352	**127352**	127930	128532	0.93 %	128929	1.24 %
Lpr-a-05	806	205499	**205499**	206086	208643	1.53 %	209138	1.77 %
Lpr-b-01	50	*14835	**14835**	**14835**	**14835**	0.00 %	**14835**	0.00 %
Lpr-b-02	101	*28654	**28654**	**28654**	**28654**	0.00 %	**28654**	0.00 %
Lpr-b-03	305	77878	**77878**	**77878**	78511	0.81 %	78594	0.92 %
Lpr-b-04	501	127454	**127454**	**127454**	128578	0.88 %	129368	1.50 %
Lpr-b-05	801	211771	**211771**	212279	214954	1.50 %	215798	1.90 %
Lpr-c-01	50	*18639	**18639**	**18639**	**18639**	0.00 %	**18639**	0.00 %
Lpr-c-02	100	*36339	**36339**	**36339**	36342	0.01 %	36363	0.07 %
Lpr-c-03	302	111632	**111632**	**111632**	112242	0.55 %	112323	0.62 %
Lpr-c-04	504	169254	**169254**	169487	170286	0.61 %	170638	0.82 %
Lpr-c-05	803	259937	**259937**	260538	262448	0.97 %	262732	1.08 %