Solving Large Scale Arc Routing Problems

Diogo Miguel Ferreira de Oliveira
diogo.miguel.oliveira@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal

December 2021

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

1. Introduction

The Capacitated Arc Routing Problem (CARP) is a combinatorial optimization problem that was first proposed by Golden and Wong in 1981 [5]. It has the objective of finding a set of routes with minimal distance that fulfill the demand in a subset of edges of the problem network, subject to constraints in the capacity of the vehicles. It has a wide range of applications such as waste collection [14], road maintenance [6], and winter gritting [3].

In this thesis, we solve a variant of CARP called the Mixed Capacitated Arc Routing Problem (MCARP), proposed by Belenguer et al. [1]. 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. The MCARP is defined on a mixed network, containing both edges and arcs. Edges can be traversed by a vehicle in both directions, while arcs only allow one direction. As most CARP and MCARP applications span an entire city, we place an emphasis on solving large-scale instances.

2. Background

2.1. Literature Review

The CARP has received considerable attention in the past few decades, with numerous algorithms published. Since the CARP is NP-Hard [5], exact methods are limited to small and medium size problems. As a consequence, heuristics and metaheuristics are often published in the literature. One of the first heuristics for the CARP was proposed by Ulusoy [12]. Instead of building routes one-by-one, this method first builds a giant tour, a route with infinite capacity that services all required edges. 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.

Currently, the state-of-the-art algorithms to solve CARP are metaheuristic algorithms. In 2008, Polacek et al. designed a simple and efficient Variable Neighborhood Search (VNS) [8]. Tang et al. introduced a Memetic Algorithm with Extended Neighborhood Search (MAENS) [10] that uses Split as a large neighborhood move. In 2017, Vidal published Unified Hybrid Genetic Search (UHGS) for the CARP, a Memetic Algorithm (MA) with advanced diversity management and constant time local search move evaluations, that is, independent of problem size, as was the case with previous works. UHGS 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 benchmarks.

With several metaheuristics capable of efficiently solving the classical instances, the focus has shifted to solving large-scale instances. The latest algorithms solve large-scale CARP by using a divide-and-conquer approach. In 2014, Mei et al. [7] 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 merged to form a solution to the main problem. Tang et al. [11] 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 [15] that uses a task rank matrix to find good and poor links to decompose the routes of a solution. RCO-SAHiD and UHGS are currently the metaheuristics with the best results on large-scale benchmarks.

2.2. Mathematical Formulation

The presented formulation is based on the MCARP formulation given by Constantino et. al in [2].

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.

$$\min \sum_{p=1}^{P} \left(\sum_{(i,j)\in A} d_{ij} y_{ij}^p + \sum_{(i,j)\in R} s_{ij} x_{ij}^p + F \sum_{(0,j)\in A} y_{0j}^p \right) \tag{1}$$

subject to

$$\sum_{p=1}^{P} x_{ij}^p = 1 \quad \forall (i,j) \in A_R \tag{2}$$

$$\sum_{p=1}^{P} (x_{ij}^p + x_{ji}^p) = 1 \quad \forall (i,j) \in A_{E_R} \tag{3}$$

$$\sum_{j:(i,j)\in A} y_{ij}^p + \sum_{j:(i,j)\in R} x_{ij}^p = \sum_{j:(j,i)\in A} y_{ji}^p + \sum_{j:(j,i)\in R} x_{ji}^p \quad \forall i \in N, \forall p \tag{4}$$

$$\sum_{(0,j)\in A} y_{0j}^p \leq 1 \quad \forall p \tag{5}$$

$$\sum_{j:(j,i)\in A} f_{ji}^p - \sum_{j:(i,j)\in A} f_{ij}^p = \sum_{j:(j,i)\in R} q_{ij} x_{ji}^p \quad \forall i \in N \setminus \{0\} \forall p \tag{6}$$

$$\sum_{(0,j)\in A} f_{0j}^p = \sum_{j:(j,i)\in R} q_{ij} x_{ji}^p \quad \forall p \tag{7}$$

$$f_{ij}^p \leq W(x_{ij}^p + y_{ij}^p) \quad \forall (i,j) \in R, \forall p \tag{8}$$

$$f_{ij}^p \leq W(y_{ij}^p) \quad \forall (i,j) \in A_D, \forall p \tag{9}$$

$$x_{ij}^p \in \{0,1\} \quad \forall (i,j) \in R, \forall p \tag{10}$$

$$y_{ij}^p \geq 0 \text{ and integer} \quad \forall (i,j) \in A, \forall p \tag{11}$$

$$f_{ij}^p \geq 0 \quad \forall (i,j) \in A, \forall p \tag{12}$$

The first term in the objective function (1) 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)–(3) ensure that every required link is serviced by only one vehicle. Equation (4) imposes the connectivity of routes at each node and equation (5) is needed to adequately charge the fixed cost in the objective function. (6)–(7) are the flow conservation constraints and (8)–(9) are the linking constraints, which together guarantee the connectivity of the routes. Equations (8)–(9) are the capacity constraints. (10)–(12) are the domain constraints.

3. Implementation

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. 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. A flowchart of the algorithm is shown in Figure 1.

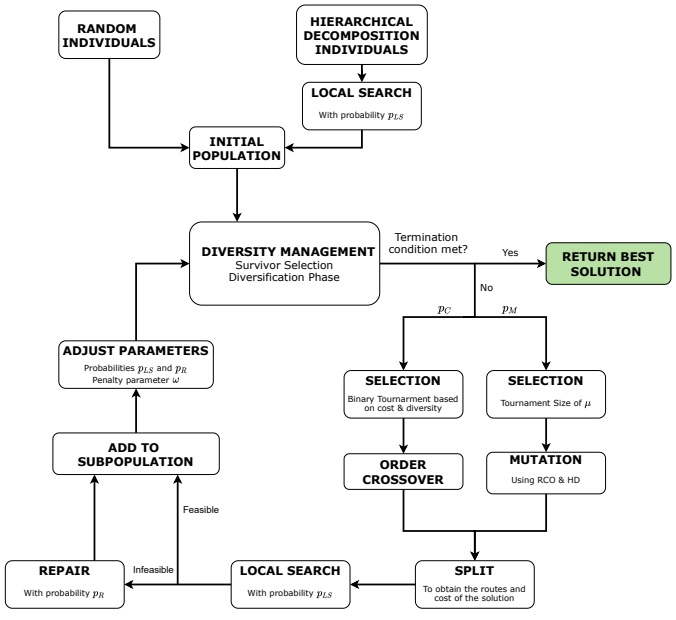

Figure 1: Flowchart of MADCoM.

3.1. 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, but recent algorithms use instead an implicit representation by decomposing the search space of MCARP into four decision subsets [13]:

- 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 and used to build $d(k, l)$, avoiding the need to compute it 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 $\mathcal{K}$ [9]. 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 $\mathcal{K}$ 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.

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 $\rho = (0, \rho(1), \ldots, \rho(|\rho| - 1), 0)$ be the set of mode choices for each service $\sigma(i)$ of route σ. Then, the route cost $C(\sigma)$ is expressed by equation 13.

$$C(\sigma) = F + \sum_{i=1}^{|\sigma|} s_i + \sum_{i=1}^{|\sigma|-1} d(\rho(i), \rho(i+1)) \tag{13}$$

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 and are penalized using equation 14, where $Q(\sigma)$ is the sum of the demands of the required links serviced by route σ and ω is the penalty parameter.

$$C_P(\sigma) = C(\sigma) + \omega \max\{0, Q(\sigma) - Q\} \tag{14}$$

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 $O(1)$.

3.2. Genetic Algorithm

3.2.1. 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.

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 indexes 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.

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 calculated with optimal mode choices. 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.

3.2.2. 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, $C(I) = \sum_{\sigma \in R_I} C_P(\sigma)$.

The fitness $BF(I)$ of an individual I is based on two factors: the cost of the solution $C(I)$ 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 $d_{\mathrm{BP}}(A, B)$ between two individuals A and B is equal to the number of adjacent services in A that are no longer adjacent in B 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. 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 15, where n_{elite} is the number of elite individuals. The diversity contribution and the biased fitness are recalculated for every individual anytime an individual is added or removed from a subpopulation.

$$BF(I) = fit(I) + \left(1 - \frac{n_{elite}}{n_{ind}}\right) dc(I) \tag{15}$$

3.2.3. 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. 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.

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.

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.

3.2.4. 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.

3.2.5. 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 HD, producing quality solutions. These solutions can also undergo local search and be repaired. The remaining individuals 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 HD. Section 4.1 demonstrates this result.

3.2.6. Diversification Phase

One of the main problems with genetic algorithms is the premature convergence of the population. 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.

3.2.7. 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.

3.3. Local Search

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. We consider 4 local search move types:

- 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.

We use the efficient local search techniques of UHGS to evaluate moves with optimal mode choices. Instead of

computing the cost of a route in the original DAG $\mathcal{K}$, we compute it on a reduced graph. The routes created by a local search move can be expressed in terms of subsequences of the original routes. The shortest paths along these subsequences can be stored, and calculating the cost of a route consists of concatenating each subsequence. To speed up local search even further, we calculate lower bounds of the route cost using the subsequences.

For every service u, chosen in random order, the local search tries to place a service v after u. Lower bounds are 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 by concatenation in $O(1)$. The move that produces the largest improvement of the solution is applied. After a move is applied, the cost for each subsequence are recomputed. Local search stops when an improving move can't be found, reaching a local minimum.

3.4. Large-Scale Heuristics

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 1. 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 1: 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 δ_O by applying HD to VT
    if δ_O = δ_I then
        Double the cutting probabilities of RCO
    else
        Break
```

3.4.1. 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 $\sigma = (0, \sigma_1, \ldots, i, j, \ldots \sigma_{|\sigma|-1}, 0)$ means generating two subsequences $(\sigma_1, \ldots, i)$ and $(j, \ldots \sigma_{|\sigma|-1})$, 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 $c_{ij}^{\text{MIN}} = \min_{k \in M_i} \left\{ \min_{l \in M_j} \{d(k, l)\} \right\}$. 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. By using ranks, Θ_{ij} represents the relative quality of having service j after service i in a route, compared to every other service.

A good link is defined as a link whose task rank is greater 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 [15].

3.4.2. 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 2) 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.

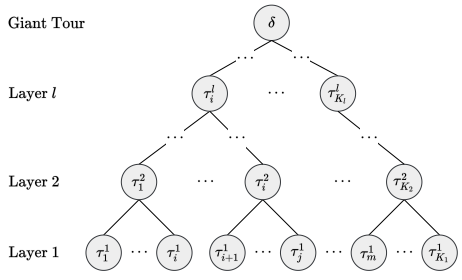

Figure 2: Hierarchical Structure of HD

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 the clustering algorithm k-medoids. 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 is given by $d(\tau_i^l, \tau_j^l) = \frac{1}{2}\left(c_{qr}^{\text{MIN}} + c_{sp}^{\text{MIN}}\right)$.

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.

4. Results

MADCoM was implemented in Python 3.9.7 and executed in a E2ds_v4 virtual machine from Azure running Windows Server 2019, with 16 GB of RAM and a Intel(R) Xeon(R) Platinum 8272CL processor with a frequency of 2.60 GHz. We evaluate the performance of MADCoM, we executed 5 runs of 30 minutes on all instances of the classical benchmarks for CARP and MCARP, as well as the more recent large-scale benchmarks.

On the classical instances, we compare MADCoM to VNS, MAENS, UHGS and the memetic algorithm by Belenguer et al. [1], designated as MABBLP. On the large-scale instances, we compare MADCoM to UHGS, RDG-MAENS, RCO-RDG-MAENS, RCO-SAHiD, as well Fast-CARP [14], a constructive heuristic tailored for large-scale problems, and Path-Scanning (PS) [4], a classical constructive heuristic.

4.1. 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 $[1, \min(\beta|VT|, \sqrt{n})]$.

A population of 100 individuals was generated using each method for four instances of different sizes and each individual underwent local search. 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 $[1, \beta|VT|]$, and random solutions improved by local search. Figure 3 shows the boxplots of the total time to generate an individual and figure 4 shows the solution quality. The boxplots of the solution quality of each HD variant before local search is applied was also included for easier comparison.

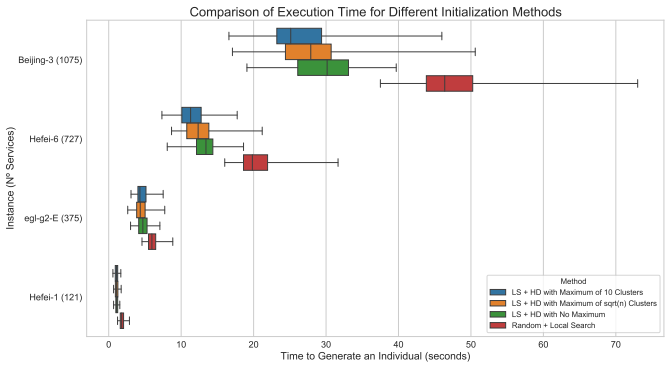

Figure 3: Boxplots of the total time to generate an individual using different variants of HD combined with local search.

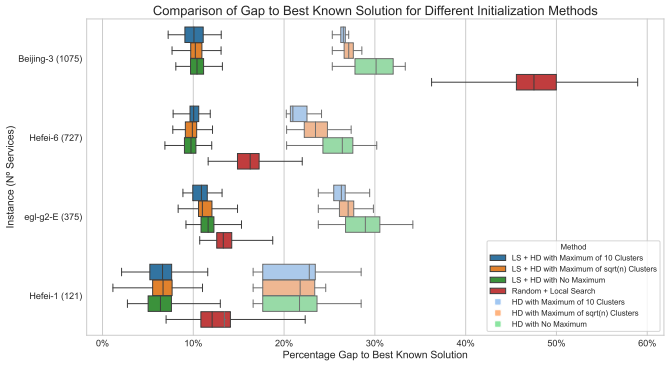

Figure 4: Boxplots of the gap to the best known solution using different variants of HD combined with local search.

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 on average into better solutions.

Another important factor is the diversity of the population. By imposing a limit on the number of clusters, the number of outcomes of HD is also reduced. Figure 5 shows the boxplots of the diversity contribution of each individual in the population, normalized by the maximum diversity $n - 1$. The populations generated with HD are not as diverse as the random individuals, especially for smaller instances, but are still suficiently diverse.

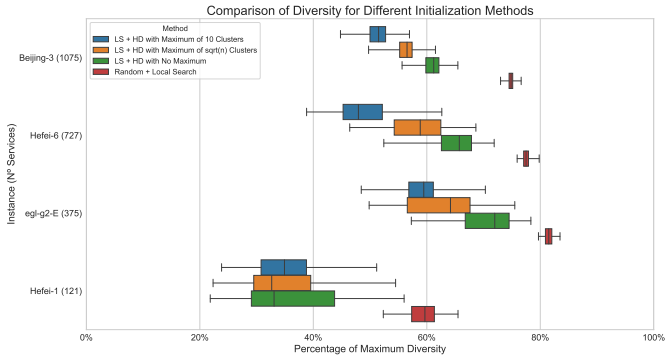

Figure 5: Boxplots of the diversity contribution of each individual in the population generated using different variants of HD combined with local search.

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 less time than random solutions improved with local search.

4.2. Results on Classical Instances

Due to the large number of instances, we only present the summarized results for the classical CARP benchmarks (GDB, VAL, BMCV and EGLESE) and the classical MCARP benchmarks (MVAL and LPR). 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 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.

4.3. Results on Large-Scale Instances

The results of MADCoM on the large-scale benchmarks sets are available in tables 2-5. 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 1: Summarized results on standard 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 2 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 the inter-route moves.

Table 2: Results on the EGL-L benchmark set.

			RDG-MAENS		UHGS		RCO-RDG-MAENS		**MADCoM**			
									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 3. 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, 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 3: 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: 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	4296005	4353966	**3765614**	3785520	3852428	3884308	4047304	7.48 %	4061605	7.86 %

Table 5 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.

Table 5: Results on the KW benchmark set.

					MADCoM			
			FastCARP	PS	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 %

4.4. 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 simpler version are the same as those of MADCoM. Each simpler version is tested on 5 runs of 30 minutes on the instances of different sizes.

To analyse the performance of the large scale heuristics, we tested simpler versions without the divide-and-conquer mutation, by setting $p_M = 0$, and without initialization using HD, by setting $f_{HD} = 0$. Table 6 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 6: 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 7, which has the same table format as Table 6.

Table 7: 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 the worst, as expected. The version without 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, 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.

5. Conclusions

5.1. Concluding Remarks

The main objective was accomplished as MADCoM is competitive on solving MCARP. We found new best so-

lutions 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 with 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.

Furthermore, 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 could help mitigate this.

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 and multiple depots. 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.

References

[1] 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.

[2] 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.

[3] R. W. Eglese. Routeing winter gritting vehicles. *Discrete Applied Mathematics*, 48(3):231–244, Feb. 1994.

[4] 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.

[5] B. L. Golden and R. T. Wong. Capacitated arc routing problems. *Networks*, 11(3):305–315, 1981.

[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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] G. Ulusoy. The fleet size and mix problem for capacitated arc routing. *European Journal of Operational Research*, 22:329–337, Feb. 1985.

[13] T. Vidal. Node, Edge, Arc Routing and Turn Penalties: Multiple Problems—One Neighborhood Extension. *Operations Research*, 65, May 2017.

[14] S. Wøhlk. A fast heuristic for large-scale capacitated arc routing problems. *Journal of the Operational Research Society*, 69, July 2018.

[15] 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.