The Distance-Constrained Generalized Vehicle Routing Problem

Markus Mattila

School of Science

Bachelor's thesis
Espoo 7.6.2016

Thesis supervisor and advisor:

Prof. Enrico Bartolini

AALTO UNIVERSITY
SCHOOL OF SCIENCE

ABSTRACT OF THE
BACHELOR'S THESIS

Author: Markus Mattila

Title: The Distance-Constrained Generalized Vehicle Routing Problem

Date: 7.6.2016 Language: English Number of pages: 4+31

Degree programme: Engineering Physics and Mathematics

Supervisor and advisor: Prof. Enrico Bartolini

In this thesis, we consider an NP-hard combinatorial optimization problem called the Distance-Constrained Generalized Vehicle Routing Problem (DGVRP). The main characteristics of this problem are unlimited vehicle fleet, limited vehicle capacity, arrangement of customers into clusters, and an upper limit on each route length.

We present a mathematical formulation for the DGVRP, in which we use two-commodity and single-commodity flow formulations to describe the constraints on vehicle capacity and maximum route length. In addition, a heuristic algorithm is implemented for finding solutions to the DGVRP. Main components in the algorithm include a split procedure and solving a set partitioning model.

The algorithm is tested on benchmark GVRP instances (with no distance constraints) as well as a new set of DGVRP instances. The results obtained by the heuristic algorithm for the GVRP are competitive compared to those presented in recent literature. The solutions for the DGVRP instances are at least as good as those given by a commercial solver.

Keywords: Combinatorial optimization, vehicle routing, distance constraints, heuristic algorithm

Preface

I want to thank Professor Enrico Bartolini for his excellent guidance and continued patience.

Otaniemi, 30.8.2016

Markus Mattila

Contents

Introduction

Problem description

The Generalized Vehicle Routing Problem (GVRP) is a combinatorial optimization problem which generalizes the Capacitated Vehicle Routing Problem (CVRP). The CVRP is to design a set of routes for a fleet of vehicles, based at a depot, to visit a set of customers so that each customer is visited exactly once. The combined demand of all customers on a route cannot exceed the vehicle capacity. An optimal solution is one that minimizes the total cost, e.g. the sum of the lengths of all routes. The GVRP, instead, contains clusters that include one or more customers, and exactly one customer per cluster must be visited.

The Distance-Constrained Generalized Vehicle Routing Problem (DGVRP) is a generalization of the GVRP, in which *distance constraints* limit the maximum length of each route. The DGVRP is NP-hard, since it contains the CVRP as a special case when distance constraints are ignored and all clusters are singletons. To our knowledge, this particular problem has not been considered in the literature.

Application examples

Real-life applications of the GVRP and DGVRP are numerous. One such application, as mentioned by Afsar et al. [1] is the delivery of supplies to scenes of natural disasters. Roads and other transportation infrastructure are often damaged in natural disasters, leaving villages dependant on airborne supply delivery. However, not every village needs to be visited, since connections between neighbouring villages may be intact. By representing each set of interconnected villages as a cluster, a GVRP can be used to model the situation. In addition to this example, a more realistic model can then be obtained by adding a constraint limiting the maximum distance the plane can travel without refuelling. By solving this DGVRP, we obtain an optimal set of routes that save fuel and expedite supplying operations.

Another example from an article by Bektaş et al. [2] is route design for trucks carrying pharmaceutical products. It may be inconvenient to drive a big truck from one small pharmacy to another, but it may be possible to drop off all the supplies for one neighbourhood in one store and deliver the products to nearby pharmacies with smaller vehicles. The clusters would now correspond to neighbouring areas in a city. By adding distance constraints, we can extend this application to take into account the maximum duration of a working day.

In addition, many different types of combinatorial optimization problems can be transformed into a GVRP. Some such problems are presented by Baldacci et al. [3], including the travelling salesman problem with profits, several types of arc routing problems, and extensions of the VRP. Adding distance constraints to the GVRP model thus allows to model more realistic variants of all such problems.

Literature review

The CVRP was first presented in 1959 by Dantzig and Ramser [4]. The CVRP has been studied extensively in the literature. For instance, Baldacci et al. [5] present an exact solution algorithm for the CVRP based on a two-commodity flow formulation. Numerous solving methods and extensions for the CVRP are presented in a book by Toth and Vigo [6]. One extension of the CVRP is the Distance-Constrained Capacitated Vehicle Routing Problem (DCVRP). This problem has been studied, e.g., by Laporte et al. [7] in 1985 and more recently in 2006 by De Franceschi et al. [8] who present a refinement heuristic for the DCVRP.

The allocation of customers into clusters has been considered already in the 1960s in the context of the Generalized Traveling Salesman Problem (GTSP). The GTSP is a well-known optimization problem, and the literature on the subject is rich; see, e.g., the exact method for the GTSP by Fischetti et al. [9].

The GVRP was, to our knowledge, first presented in an article by Ghiani and Improta [10] in 2000. The article describes a transformation of the GVRP to a Capacitated Arc Routing Problem (CARP). Baldacci et al. [3] present several applications for the GVRP in their 2010 article, showing that the GVRP can be used to model a number of different combinatorial optimization problems. Bektaş et al. [2] were the first to present a direct solution method for the GVRP in 2011. In the article, four new mathematical formulations for the GVRP are presented, exact methods are presented and applied, a heuristic algorithm is presented, and a data set of test instances is created.

Recently, GVRP variants have gained more attention in the literature. Moccia et al. [11] present the GVRP with Time Windows (GVRPTW) in their 2012 article, which is the first to consider time windows for the GVRP. In the article, a tabu search heuristic is presented. Hà et al. [12] consider the GVRP with unlimited fleet size in their 2013 article. They present a two-commodity flow formulation and develop both an exact and a heuristic method for the GVRP. In 2014, Afsar et al. [1] describe exact and heuristic solution methods for the GVRP with unlimited fleet size.

Thesis outline

This thesis is structured into four sections. In the first one, we describe the DGVRP and present mathematical formulations including the distance constraints. The flow of goods is modelled as a two-commodity flow and time as a single-commodity flow.

In section 2, we describe a metaheuristic algorithm for the DGVRP. An important part of the algorithm is a split procedure, similar to those implemented by Hà et al. [12] and Afsar et al. [1] which determine the optimal way of splitting a giant tour (a route visiting all clusters) into a feasible set of routes that satisfy the capacity constraints. Our algorithm is modified to consider distance constraints, remaining exact in the case that the distance and time matrices are equal. The heuristic algorithm also includes a local search procedure and a set partitioning model that has previously not been utilized in GVRP heuristics. The set partitioning model is based on a pool of feasible routes found by the heuristic algorithm.

Computational results are presented in section 3. We use the benchmark GVRP instances created by Bektaş et al. [2] to evaluate the performance of the heuristic algorithm compared to other studies. We also present a new set of test instances for the DGVRP. Conclusions can be found in section 4.

1 Mathematical Formulations

In this section, we present mathematical formulations for the GVRP and the DGVRP. We begin with defining the necessary problem properties, and continue to present the objective functions and constraints. To model the GVRP, we use a two-commodity flow formulation presented by Baldacci et al. [5]. To model the distance constraints, we use a single commodity flow as described by Bektaş and Lysgaard [13]. With the formulation, exact solutions to the DGVRP can be calculated.

1.1 Problem Description

The GVRP is defined on a directed graph $G = (V,A)$, where $V = \{v_0,v_1,...,v_{n+1}\}$ is the vertex set and $A = \{(i,j)|\, v_i,v_j \in V, i \neq j\}$ is the arc set. All vehicles start and end their route at the depot. For simplicity, the depot is split into two separate vertices v_0 and v_{n+1} so that each route begins at v_0 and ends at v_{n+1}. Vertices $v_1, v_2,..., v_n$ represent the n ordinary customers. Customers are divided into K clusters, which form the cluster set $C = \{C_1,C_2,...,C_K\}$. Each cluster C_k is associated with a demand q_k. Because only one customer per cluster is visited, we can assign to every customer v_i a demand $\tilde{q}_i$ that is equal to the demand of the cluster of its cluster. The sum of demands of the customers (or clusters) visited on a single route cannot exceed the total vehicle capacity Q. Bektaş et al. [2] and Moccia et al. [11] considered the problem with a fixed amount m of vehicles. Similarly to Hà et al. [12] and Afsar et al. [1], we consider instead a scenario where the fleet size is flexible, i.e., we impose no constraint on the number of vehicles in the solution.

Each arc (i,j) has a cost d_{ij} that is equal to the distance between the respective nodes v_i and v_j. The costs are assumed to be symmetric, i.e., $d_{ij} = d_{ji}$ for all vertex pairs. In addition, each arc (i,j) is associated with a time t_{ij}. The total time it takes to any one route to travel from v_0 to v_{n+1} must be less or equal than T, an upper time limit.

1.2 Two-commodity flow formulation of the GVRP

To express the GVRP mathematically, we need a set of decision variables. First, a binary variable x_{ij} indicates whether the arc (i,j) is in use in the solution: if the arc is used, x_{ij} equals 1, and 0 otherwise. Similarly, a binary variable y_i indicates if customer v_i is visited in the solution. The number of routes or vehicles m is also treated as a decision variable.

$$\min \sum_{i=0}^{n+1} \sum_{j=0}^{n+1} d_{ij} x_{ij} \tag{1}$$

$$s.t. \sum_{j=1}^{n} x_{0j} = m \tag{2}$$

$$\sum_{j=1}^{n} x_{jn+1} = m \tag{3}$$

$$\sum_{i \in C_l} y_i = 1 \quad \forall\, l = 1,2,\ldots,K \tag{4}$$

$$\sum_{i=0}^{n} x_{ij} = y_j \quad \forall v_j \in V \setminus \{v_0, v_{n+1}\} \tag{5}$$

$$\sum_{k=1}^{n+1} x_{jk} = y_j \quad \forall v_j \in V \setminus \{v_0, v_{n+1}\} \tag{6}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A \tag{7}$$

$$y_i \in \{0,1\} \quad \forall v_i \in V \setminus \{v_0, v_{n+1}\} \tag{8}$$

$$m \in \mathbb{Z}_+ \tag{9}$$

The objective of the problem is to minimize the total cost of the routes (1). Constraints (2) and (3) state that exactly m vehicles enter and leave the depot. Constraints (4) ensure that exactly one customer per cluster is visited. Constraints (5) and (6) imply that if a customer is visited, there is exactly one arc entering it and one arc leaving it. If a customer is not visited, no arcs arrive at it or depart from it. Constraints (7) - (9) simply state the variable types.

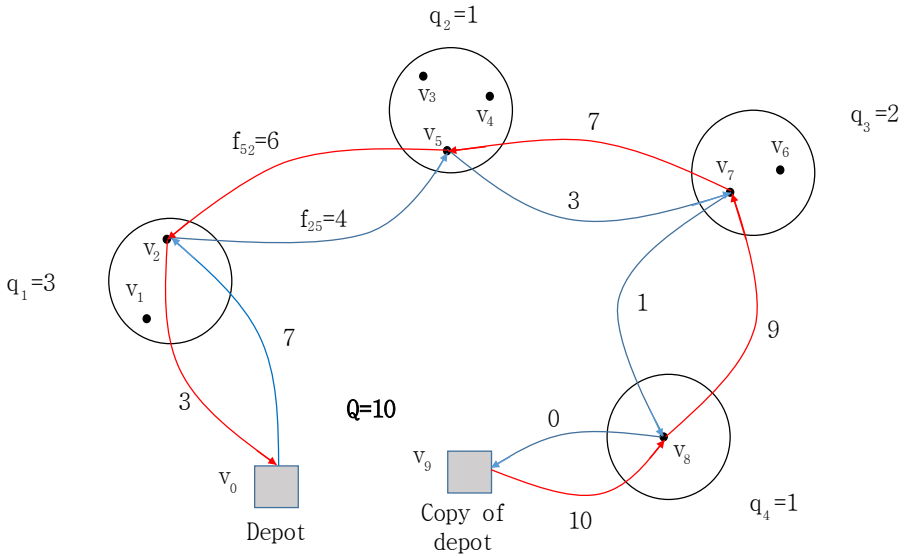

Figure 1: An example of two-commodity flow

To mathematically describe the delivery of goods to customers, we use a two-commodity flow formulation with variables f_{ij}. The formulation ensures that the solution contains no subtours, i.e., directed cycles not passing through the depot. The two-commodity flow formulation was first introduced by Baldacci et al. [5], and adapted to the GVRP by Hà et al [12]. An example of a two-commodity flow

modelling the solution of a GVRP instance with four clusters and one vehicle can be viewed in Figure 1.

For each arc (i,j), f_{ij} describes the flow of goods from v_i to v_j, that is, *the amount of goods* in a vehicle as it travels from v_i to v_j. This flow is illustrated with blue arcs in Figure 1. At the same time, f_{ji} describes *the amount of empty space* in a vehicle traversing from v_i to v_j. The empty space is modelled as a backward flow, illustrated with red arcs in the figure. This backward flow between two customers then simply equals $Q - f_{ij}$ if those two customers are consecutively visited on the route, or zero otherwise. As an example, Figure 1 illustrates the flows and backward flows associated with a route visiting the vertex sequence $(v_0,v_2,v_5,v_7,v_8,v_9)$.

The following constraints are added to the formulation to model the flows and backward flows:

$$f_{ij} + f_{ji} = Q(x_{ij} + x_{ji}) \quad \forall (i,j) \in A \tag{10}$$

$$\sum_{j=1}^{n} f_{n+1j} = Qm \tag{11}$$

$$\sum_{j=1}^{n} f_{0j} = \sum_{k=1}^{K} q_k \tag{12}$$

$$\sum_{j=0}^{n+1} f_{ji} - \sum_{j=0}^{n+1} f_{ij} = 2\tilde{q}_i y_i \quad \forall v_i \in V \setminus \{v_0,v_{n+1}\}, i \neq j \tag{13}$$

$$f_{ij} \geq 0 \quad \forall (i,j) \in A \tag{14}$$

Constraints (10) ensure that when v_i and v_j are not visited in sequence, both the flow and the backward flow between these vertices is zero. However, if they are visited in sequence, the sum of goods and empty space must equal the vehicle capacity. Constraint (11) states that the total amount of backward flow outgoing from the destination depot equals the total space in all used vehicles, that is, all vehicles arrive at the depot fully unloaded.

Similarly, (12) ensures that the total amount of goods loaded at the depot equals the amount distributed to the customers. Constraints (13) describe the changes in flows as a customer is visited. Let us examine the situation in Figure 1. Consider v_7, its predecessor v_5 and its successor v_8. The only flows entering v_7 are f_{57} and f_{87}, which represent the actual flow from v_5 and backward flow from v_8. Correspondingly, the flows leaving v_7 are f_{78} and f_{75}. When the amount $\tilde{q}_7 = 2$ is unloaded at v_7, the change in empty space is $f_{87} - f_{75} = 9 - 7 = 2$, which is the same as the change in the amount of goods on the vehicle, i.e., $f_{57} - f_{78} = 3 - 1 = 2$. Thus, the total difference in the sums of (13) equals $2\tilde{q}_7 = 4$. For unvisited customers, there are no flows so the difference of flows is also zero. Constraints (14) ensure the non-negativity of each flow variable.

The two-commodity flow formulation ensures that the vehicle capacity is never exceeded. Because each flow variable value is non-negative and the sum of forward flows and backward flows on each arc must equal Q, a forward flow can never exceed

Q. The formulation also eliminates subtours from the solution. A subtour would need to contain only customers with demand 0 in order to appear in the solution. Otherwise the forward flow along the subtour would have to decrease after each arc traversed. Since all the demands are strictly positive, such a subtour cannot exist.

1.3 Distance constraints

To include distance constraints (or time limits) in the problem, there is still need for one set of flow variables. We use a set of variables h_{ij} which represent the time at which customer v_j is reached after traversing the arc (i,j). This single commodity flow formulation is also used by Bektaş and Lysgaard [13] in the context of the multiple Traveling Salesman Problem (mTSP).

To model distance constraints, the following constraints are added to the formulation:

$$\sum_{j=0}^{n+1} h_{ij} - \sum_{k=0}^{n+1} h_{ki} = \sum_{j=0}^{n+1} t_{ij}x_{ij} \quad \forall\, v_i \in V \setminus \{v_0,v_{n+1}\}, i \neq j \tag{15}$$

$$h_{ij} \leq T x_{ij} \quad \forall\, (i,j) \in A \tag{16}$$

$$h_{0\,i} = t_{0\,i}x_{0\,i} \quad \forall v_i \in V \tag{17}$$

$$h_{ij} \geq 0 \quad \forall\, (i,j) \in A \tag{18}$$

Constraints (15) simply state that the difference between the arrival times at v_j and v_i must be t_{ij} if the arc (i,j) is traversed. Constraints (15) are also sufficient to eliminate subtours. Assuming positive travel times between any pair of customers, time flow variables can only increase along a directed path. If there was a subtour in the solution, equations 15 would not admit a feasible solution. Constraints (16) impose that the upper time limit cannot be exceeded when arriving at the depot, and that the flow value is at most zero on untraversed arcs. Constraints (18) ensure that the flow h_{ij} equals zero in this case. Constraints (17) force the arrival time at customer v_i to be equal to the time needed to travel from the depot to the customer, if v_i is the first customer on a route.

An example of the time flow variables can be viewed in Figure 2. The black arcs measure the time between customers, and the blue curved arcs represent the flow variables h_{ij}. The flow variables h_{01} and h_{06} simply equal the travel times from the depot to the first customers of the two routes. The other flow variables are determined by the formula in constraints (17). Note that if the solution included the arc (2,4), the corresponding flow variable would have value 22. This would make the flow variable h_{47} violate the upper limit constraint (16).

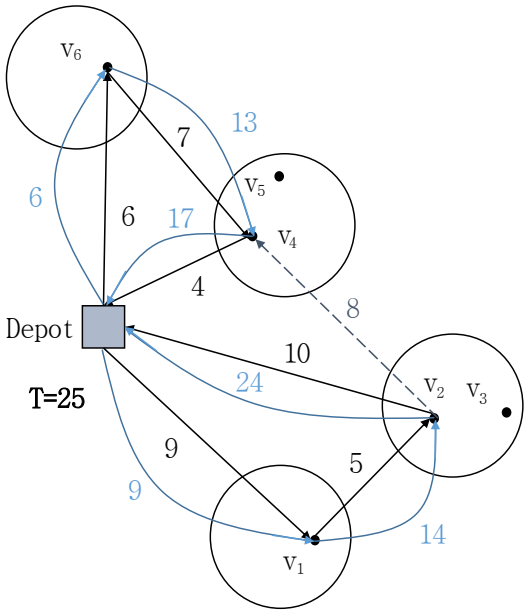

Figure 2: An example of the time flow variables

1.4 Set Partitioning

As a part of our metaheuristic algorithm, we solve a set partitioning problem. Generally, the set partitioning problem is to divide a set into subsets so that each element belongs to exactly one subset. In the GVRP, the problem is to divide the cluster set into routes, so that each cluster belongs to exactly one route and the total combined cost of the routes is minimized. Let R be the set of all feasible routes. Let c_r be the cost of route r (i.e., the sum of costs of the arcs it traverses), and let z_r be a 0-1 variable taking value 1 if and only if the route r is used in the solution. Define also a binary indicator p_{kr} having value 1 if some customer in cluster C_k is visited by route r, and 0 otherwise.

The set partitioning formulation for the GVRP is then the following:

$$\min \sum_{r \in R} c_r z_r \tag{19}$$

$$s.t. \sum_{r \in R} p_{kr} z_r = 1 \quad \forall k = 1,2,\ldots,K \tag{20}$$

$$z_r \in \{0,1\} \tag{21}$$

Similar formulations can be used to model a wide range of VRP variants. To the best of our knowledge, in the context of the GVRP Afsar et al. [1] are the only to develop a solution algorithm based on the set partitioning formulation. The formulation is simple to state, but the difficulty is in constructing the enormous solution space of all feasible routes R. The model could be solved, for example, with column generation techniques, but instead we only use the routes generated by the heuristic algorithm. Thus, we have no guarantee to obtain an optimal solution.

2 Heuristic Algorithm

Solving a GVRP to optimality becomes increasingly more difficult as more customers are included. According to our preliminary experiments, the commercial solver CPLEX (IBM ILOG CPLEX 12.6) is only able to find optimal solutions to GVRP instances with up to 50 customers within a time limit of two hours, using the formulation presented in the previous section. In addition, the computation was performed without considering the time flow variables and thus without considering distance constraints. Including these constraints would delay the computation even further.

Exact solution methods are typically impractical to be used in real-world applications, where the number of customers is often in the hundreds or thousands. Therefore, a different method of route construction is needed. In this section, we present a heuristic algorithm that searches solutions to the problem. The cost of any feasible solution from the heuristic algorithm is an upper limit to the optimal objective value. The best value returned by the heuristic can be compared with the best known solution value or a lower bound in order to evaluate the performance of the algorithm. The algorithm only handles symmetric instances, since the test instances are all defined on an undirected graph.

The structure of the algorithm is mainly the same as used by Hà et al. [12]. In addition to local search techniques, the algorithm includes a large neighbourhood search procedure called the *split algorithm*. The same algorithm is used by Afsar et al. [1]. Our implementation is, however, tailored to include the distance constraints. We have also added a heuristic set partitioning model at the end of the algorithm. To our knowledge, this approach has not been considered in the context of the GVRP.

The pseudo-code structure for the heuristic can be viewed in Algorithm 1. The various functions are later presented in more detail. We start by creating an initial set of routes, and then perform a simple local search on the resulting solution. All the routes are then merged together to form a giant tour that covers all the clusters. This giant tour is mutated to allow for more variation. Then the split procedure divides the giant tour into a set of feasible routes which are improved with another round of local search. Each improved initial solution is mutated and split n_m times (we have chosen this number to be 50). The procedure is repeated n_i =30 times, and the resulting routes are stored in different stages of the algorithm. Finally, an exact set partitioning method is used to combine the collected routes to find the best solution.

Algorithm 1: The metaheuristic algorithm

```
Parameters: n_i, n_m
RoutePool ← ∅
for i ← 1 to n_i do
    Routes ← InitialRoutes(i)
    Routes ←LocalSearch(Routes)
    Add Routes to RoutePool
    GiantTour ← Concat(Routes)
    for j ← 1 to n_m do
        MutatedTour ← Mutate(GiantTour)
        Routes ← Split(MutatedTour)
        Add Routes to RoutePool
        Routes ← LocalSearch(Routes)
        Add Routes to RoutePool
    end
end
Solution ← SetPartitioning(StoredRoutes)
```

2.1 Initial Route Generation

We use two different methods to obtain as diverse initial solutions as possible. These solutions are later improved by other algorithms. The outline of the initial route generation algorithm is presented in Algorithm 2.

The first method is the same as used by Hà et al. [12]. Routes are constructed iteratively using a greedy approach. At each iteration, the vehicle determines the two unvisited clusters closest to the last one and chooses one by random. The visited customer in that cluster is also decided randomly. At each step, it is ensured that there is enough time to visit the chosen cluster and return back to the depot. If the truck is fully loaded or there is no time to visit more clusters, the vehicle returns to the depot and the process starts again, until every cluster has been visited.

One benefit of this algorithm is the generation of diverse solutions. The greedy choices in the generation process aim at including in the heuristic some elements that could be found in an optimal solution, which is an advantage compared to a completely random route construction. On the downside, the resulting routes are close to their feasibility bounds, which limits the possibilities for improvement by local search. The routes are also similar in shape, since the first customer of each route is always close to the depot.

The other initial route construction algorithm is much simpler. It forms exactly K routes; one for each cluster. On any route, only one customer is visited by the vehicle before returning back to the depot. This type of solution is known as a 'daisy'. The advantage of this procedure is that altering the routes in the local search algorithm is easy, since the routes are not close to their feasibility bounds. On the

other hand, the 'daisy' lacks the random suboptimal elements that are included in the first route generation method, so the local search algorithm has a greater effect on these routes. These differences between the two initial route construction methods are the reason why both methods are included in the final algorithm.

Algorithm 2: Route generation algorithm

Input: i

Routes $\leftarrow \emptyset$
if i **is even then**
  visited $\leftarrow 0$
  while $visited < K$ **do**
    Route $\leftarrow \emptyset$
    load $\leftarrow 0$
    time $\leftarrow 0$
    node $\leftarrow v_0$
    while $node \neq v_{n+1}$ **do**
      next $\leftarrow$ random node from one of two unvisited clusters closest to node
      N $\leftarrow$ the cluster of next
      if $load + q_N \leq Q$ **and** $time + t_{node,next} + t_{next,v_{n+1}} \leq T$ **then**
        Add arc (node, next) to Route
        Label N as visited
        load $\leftarrow$ load + q_N
        time $\leftarrow$ time + $t_{node,next}$
        node $\leftarrow$ next
        visited $\leftarrow$ visited + 1
      end
      else
        Add arc (node, v_{n+1}) to Route
        node $\leftarrow v_{n+1}$
      end
    end
    Add Route to Routes
  end
end
else
  for $k \leftarrow 1$ **to** K **do**
    In cluster k, choose the customer v_s closest to the depot
    Add arcs (v_0,v_s) and (v_s,v_{n+1}) to the solution
  end
end

Output: Routes

2.2 Local Search

The local search algorithm improves a given solution in order to find a local minimum. In our approach, the local search algorithm executes three basic types of moves: one-point move, two-point move, and 2-opt, in that order. The pseudocode for the local search algorithm is presented in Algorithm 3.

The one-point move relocates one cluster. First, a randomly chosen cluster is removed from the solution and its predecessor and successor are connected to each other. The algorithm then calculates the cost of inserting the cluster back between each pair of clusters on all routes, at the same time determining the optimal customer to visit in the cluster. The cluster is reallocated into the best position, leaving the solution unchanged if no improving reallocations are possible. The one-point move repeats this process until no improvement has been made in the last n_r moves. We use $n_r = \max(100,n)$, where n is the number of customers.

The two-point move is very similar to the one-point move. At first, one cluster is chosen randomly. Instead of reallocating the cluster, however, the algorithm tries to swap the chosen cluster with another one. If an improving swap exists, the one yielding the greatest saving is chosen and executed, otherwise the solution remains the same. The algorithm runs until no improvement has been made in the last n_s moves. In our algorithm, we set $n_s = n_r$.

Finally, the 2-opt algorithm alters the improved routes. The algorithm examines a chain of vertices, deleting arcs that connect it to the rest of the route. The chain is then reversed and connected back to the route in the only feasible way, if the resulting route has smaller cost than the original one. When all vertex chains are considered and no more improvements can be made, the route is 2-optimal. Roughly speaking, a 2-optimal route never crosses itself. The classical 2-opt algorithm is tailored to a problem including clusters as presented by Fischetti et al. [9]. An example of this generalized 2-opt is presented in Figure 3. The solid and dashed black lines represent the route before and after the 2-opt move, respectively. The reversed chain is highlighted with the red dashed line. The customers visited are redetermined not only in the first and last cluster of the reversed chain (clusters C_2 and C_4), but also in all clusters neighboring C_2 and C_4.

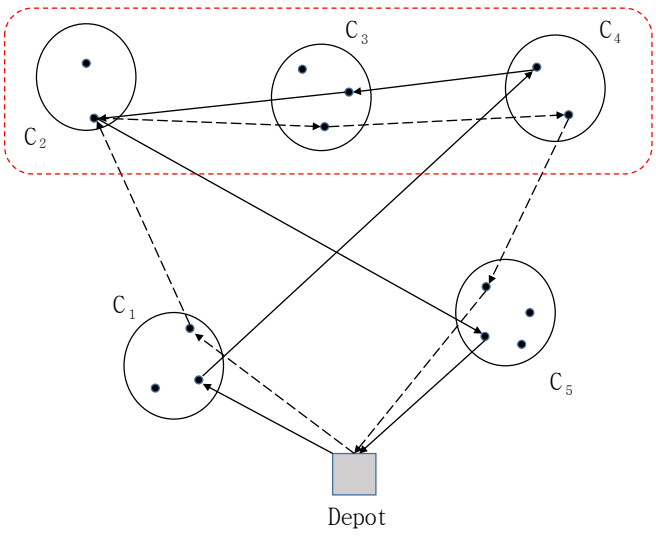

Figure 3: A modified 2-opt move example

Algorithm 3: The local search algorithm

Input: Routes, n_r, n_s

BestInsertionCost $\leftarrow \infty$
counter $\leftarrow$0
while *counter* $< n_r$ **do**
  Saving $\leftarrow$ saving from removing cluster C_k from Routes
  for all *arcs (i,j) in the remaining solution* **do**
    for all *customers* $v_k \in C_k$ **do**
      if *insertion between* v_i *and* v_j *is feasible* **then**
        InsertionCost$\leftarrow$ min$\{d_{ik} + d_{kj} - d_{ij}\}$
        if *InsertionCost<BestInsertionCost* **then**
          BestInsertionCost $\leftarrow$ InsertionCost
          $(v_a, v_b, v_c) \leftarrow (v_i, v_j, v_k)$
        end
      end
    end
  end
  counter $\leftarrow$ counter+1
  if *BestInsertionCost<Saving* **then**
    Insert v_c to Routes between v_a and v_b
    counter $\leftarrow$ 0
  end
end

counter $\leftarrow$0
while *counter* $< n_s$ **do**
  BestResult $\leftarrow \infty$
  Choose a cluster C_k
  for all $C_s \in C, C_s \neq C_k$ **do**
    if *swapping* C_k *and* C_s *is feasible* **then**
      Saving $\leftarrow$ saving from removing both clusters C_k and C_s
      SwapCost $\leftarrow$ minimal cost of inserting C_k in the place of C_s and vice versa
      if *SwapCost-Saving<BestResult* **then**
        BestResult $\leftarrow$ SwapCost-Saving
        $(C_a, C_b) \leftarrow (C_k, C_s)$
      end
    end
  end
  counter $\leftarrow$ counter+1
  if *BestResult<0* **then**
    Execute swap between C_a and C_b
    counter $\leftarrow$ 0
  end
end

Perform the 2-Opt move on Routes
Output: Routes

2.3 Concat & Split

After the local search, we have a set of feasible routes of fairly good quality; they are 2-optimal, and cannot be easily improved by simple local search methods. To attempt improving the corresponding solution, we alter it and create a new set of routes to be further improved by the local search. The concat algorithm concatenates the routes in the current solution to form one giant tour, and the split algorithm divides the resulting giant tour into feasible routes. These algorithms extend those presented by Hà et al. [12] and Afsar et al. [1] for the GVRP. In the case where the distances t_{ij} are directly proportional to arc costs d_{ij}, the split algorithm is exact, meaning that the algorithm is guaranteed to optimally split the giant tour.

The concat algorithm first determines a set of 'endpoints' which correspond to the first and last clusters on each route. For each pair of these endpoints, it then computes the cost saving that would be achieved by disconnecting the endpoints from the depot and connecting them to each other. The endpoints yielding the greatest saving are connected unless they belong to the same route, and removed from the set of endpoints. The orientation of the routes merged this way is adjusted, if necessary. The merging process is repeated until there are only two endpoints left. These two last endpoints correspond to the first and the last clusters of a giant tour visiting every cluster.

Before the resulting giant tour is split, a mutating procedure executes a two-point move twice by making two consecutive cluster swaps. This way, the split yields routes that are not entirely similar to routes obtained by the local search. The same giant tour is mutated and split n_m times. In our algorithm, $n_m = 50$.

The split algorithm uses dynamic programming to split an ordered sequence of clusters into a set of feasible routes. The algorithm is exact if there are no distance constraints or if the distances are directly proportional the costs, i.e., if $t_{ij} = \alpha d_{ij} \, \forall (i,j) \in A, \alpha > 0$. If this is not the case, then it is not guaranteed that the giant tour is split optimally.

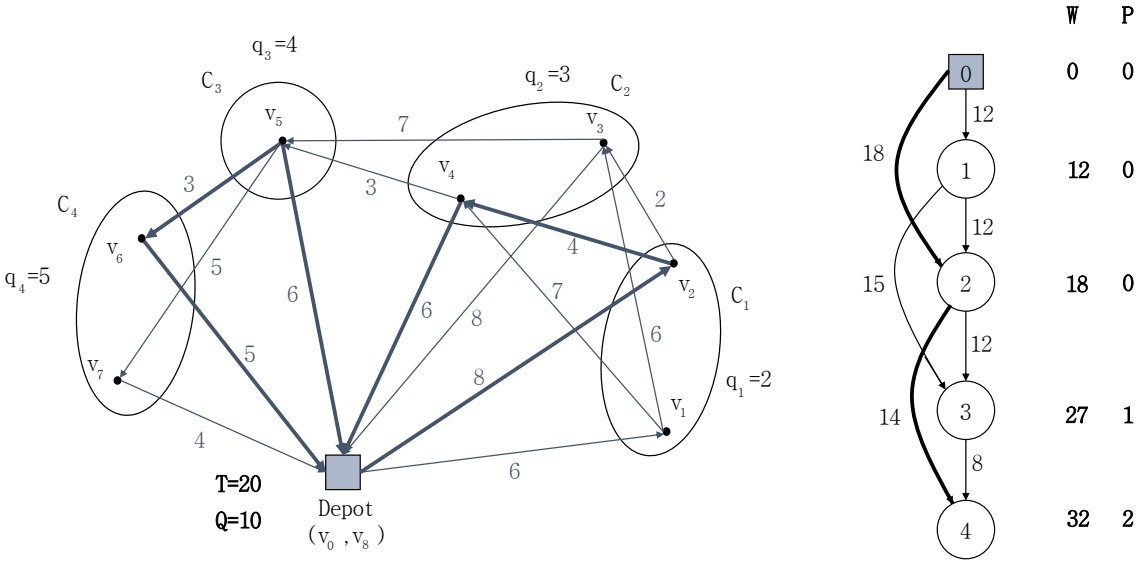

Figure 4: An example of the split algorithm and the auxiliary graph

Algorithm 4: The split algorithm

Input: cluster sequence $(C_0, C_1, ..., C_K)$
W(0) $\leftarrow$ 0
P(0) $\leftarrow$ 0
for $i \leftarrow$ *1* **to** K **do**
  W(i) $\leftarrow \infty$
  Z(C_i,v_s) $\leftarrow \infty$ **for all** $v_s \in V$
end
for $i \leftarrow$ *1* **to** K **do**
  j $\leftarrow$ i
  load $\leftarrow$ 0
  while $j \leq K$ **do**
    load $\leftarrow$ load+q_j
    if *load* $\leq Q$ **then**
      for all v_s **in** C_j **do**
        mincost $\leftarrow \infty$
        length $\leftarrow \infty$
        if *i=j* **then**
          mincost $\leftarrow d_{v_0 v_s} + d_{v_s v_{n+1}}$
          length $\leftarrow t_{v_0 v_s} + t_{v_s v_{n+1}}$
          U(C_i,v_s) $\leftarrow v_0$
        end
        else
          for $v_r \in C_{j-1}$ **s.t.** $L(C_i, v_r) + t_{v_r v_s} + t_{v_s v_{n+1}} - t_{v_r v_{n+1}} \leq T$ **do**
            if $Z(C_i,v_r) + d_{v_r,v_s} + d_{v_s v_{n+1}} - d_{v_r v_{n+1}} <$*mincost* **then**
              mincost $\leftarrow Z(C_i, v_r) + d_{v_r v_s} + d_{v_s v_{n+1}} - d_{v_r v_{n+1}}$
              length $\leftarrow L(C_i, v_r) + t_{v_r v_s} + t_{v_s v_{n+1}} - t_{v_r v_{n+1}}$
              U(C_i,v_s) $\leftarrow v_r$
            end
          end
        end
        Z(C_i,v_s) $\leftarrow$ mincost
        L(C_i,v_s) $\leftarrow$ length
      end
      W(j) $\leftarrow \min\{W(j), W(i-1) + \min_{v_s \in C_j} Z(C_i,v_s)\}$
      P(j) $\leftarrow i-1$ if W(j) is changed
    end
    j $\leftarrow$ j+1
  end
end
Routes $\leftarrow$ `ExtractSolution(`*P, Z, U*`)`
Output: Routes

Assuming to traverse the cluster sequence in the given order, the goal is to determine when to add a detour to the depot so that the routes between any two consecutive depot visits are feasible and have minimum total cost. To accomplish this, we use an auxiliary graph which contains $K + 1$ nodes: node 0 for the depot and K nodes corresponding to the clusters. The nodes are connected with weighted arcs. Each arc (i,j) emanating from node i in the auxiliary graph corresponds to a least-cost route that starts with the cluster $i + 1$ and ends with cluster j before returning to the depot. The arc weight is the cost of that route. The problem is then to find the least-cost path from node 0 to the final node K in the auxiliary graph. The number of arcs on the shortest path indicates the amount of routes in the solution and the visited nodes correspond to the clusters that are the last ones on their routes.

To determine the shortest path, each node i in the auxiliary graph is associated with a label $W(i)$ corresponding to the cost of the shortest path from node 0 to node i. Thus, the label $W(i)$ is equal to the minimum cost of traversing all clusters up to the i:th one and returning to the depot. $W(K)$ is then the cost of the final solution. In addition, each node i in the auxiliary graph has a label $P(i)$ that indicates the predecessor node of i in the shortest path from 0 to i. For a route ending with the i:th cluster, the label $P(i)$ then stores the cluster that was the last one on the previous route. To determine the labels, we use Bellman's algorithm. The auxiliary graph is not constructed explicitly but each pair of labels $W(i)$ and $P(i)$ is updated whenever an improved path from node 0 to node i is found.

A small example instance of four clusters and the corresponding auxiliary graph are presented in Figure 4. In this example, the giant tour is the cluster sequence (C_1,C_2,C_3,C_4). The cost and distance values between vertices are equal and they are indicated by the numbers next to the arcs. Bold arrows in the DGVRP graph represent arcs that are included in the final optimal solution. The auxiliary graph contains arcs, nodes and labels as described above. The bold arrows in the auxiliary graph denote the shortest path from node 0 to node K, corresponding to the optimal split in the DGVRP graph.

The outline of the split algorithm can be found in Algorithm 4. For simplicity, we assume that the cluster sequence is organized so that C_1 is the first cluster to be visited, C_2 is the second etc. The algorithm examines each cluster C_i in turn, extending the route starting from C_i until the vehicle capacity or the maximum route length is exceeded. After this, routes with C_{i+1} as the first cluster are considered. The process continues, until the routes containing only cluster C_K have been examined.

The algorithm stores labels $Z(C_i,v_s)$, $L(C_i,v_s)$ and $U(C_i,v_s)$ for each cluster-customer pair (C_i,v_s) such that v_s can be reached on a route starting from C_i. $Z(C_i,v_s)$ stores the minimum cost of such a route. $L(C_i,v_s)$ stores the length of the least-cost route while $U(C_i,v_s)$ stores the predecessor of v_s on that route.

As the first step, we set $j = i$. For each customer v_s belonging to C_j, $Z(C_i,v_s)$ and $L(C_i,v_s)$ are set equal to the cost and distance of a single-customer route and $U(C_i,v_s)$ is set equal to v_0. After this, j is set equal to $j + 1$. When $j \neq i$, the cluster C_j is not the first one on the route. In this case, all connections between clusters C_{j-1} and C_j are considered. The labels $Z(C_i,v_r)$ have already been defined for all

$v_r \in C_{j-1}$. To obtain the labels $Z(C_i,v_s)$ for $v_s \in C_j$, we calculate the minimum additional cost of inserting v_s between v_r and the depot, added to the value of the label $Z(C_i,v_r)$. The length of this route is stored in $L(C_i,v_s)$ and the predecessor customer in C_{j-1} is stored in $U(C_i,v_s)$.

When all the labels have been set for customers v_s in cluster C_j, the algorithm moves to updating the label $W(j)$. Label $W(i-1)$ corresponds to the minimum cost of traversing the cluster sequence that precedes the current route. If one of the labels $Z(C_i,v_s)$ added to the label $W(i-1)$ is smaller than the current value of $W(j)$, the label is updated. In this case, we also update the label $P(j)$ that stores the last cluster on the previous route. C_i is the first cluster on the current route, so $P(j)$ is set to $i-1$.

Consider the example in Figure 4. The algorithm starts by examining the nodes v_1 and v_2 belonging to the first cluster. The label $Z(C_1,v_1)$ gets the value $d_{01} + d_{18} = 6 + 6 = 12$. Similarly, $Z(C_1,v_2) = 8 + 8 = 16$. The corresponding predecessor labels are $U(C_1,v_1) = U(C_1,v_2) = v_0$. The length labels have identical values to the cost labels in this example and are not considered furthermore. The label $W(1)$ is set to the smaller value of the cost labels (added to $W(0) = 0$), which is 12. The route starts at cluster C_1, so $P(1)$ is updated to $1 - 1 = 0$.

The algorithm then proceeds by examining nodes v_3 and v_4 in cluster C_2. Inserting customer v_3 between v_1 and the depot, an additional cost of $d_{13}+d_{38}-d_{18} = 6+8-6 = 8$ is accumulated. Adding this to the label $Z(C_1,v_1) = 12$ yields a result of 20. If the insertion is made between customer v_2 and the depot, a similar calculation gives the result 18. The latter value is smaller, so we set $Z(C_1,v_3) = 18$ and $U(C_1,v_3) = v_2$. For customer v_4, the label values are $Z(C_1,v_4) = 18$ and $U(C_1,v_3) = v_2$. The label $W(2)$ is updated to equal the minimum of these values: $W(2) = 18$. The route started at the first cluster C_1, so $P(2) = 0$.

Proceeding to cluster C_3, there is no feasible way of inserting customer v_5 on the route; both possibilities would result in a route longer than the upper limit T. No more labels are updated. Instead, the process starts again with C_2 as the first cluster. Now the label $W(1)$ cannot change. When updating each label $W(i)$, the value $W(2-1) = 12$ must be added to the cost labels. If an improvement is made for node i, $P(i)$ should be set to 1.

The algorithm runs until C_4 has been considered as the starting cluster. The label $W(4) = 32$ corresponds to the cost of the optimal solution. $P(4) = 2$ and $P(2) = 0$, so the solution consists of two routes that start with clusters C_1 and C_3. Using the predecessor labels, the whole solution can be extracted with Algorithm 5.

After the split, the routes in the solution are stored. Local search is then performed to improve the solution quality, and the resulting routes are stored. After this, the original giant tour is mutated and split again.

Algorithm 5: Algorithm for extracting the DGVRP solution by the split algorithm

Input: P, Z, U

Routes $\leftarrow \emptyset$
endcluster $\leftarrow$ K
while *endcluster* $\neq 0$ **do**
    Route $\leftarrow \emptyset$
    startcluster $\leftarrow$ P(endcluster)+1
    node $\leftarrow v_s$ with $\min_{v_s \in endcluster}$ Z(startcluster, v_s)
    Add the arc between *node* and v_{n+1} to Route
    while *node* $\neq v_0$ **do**
        prev $\leftarrow$ U(startcluster, *node*)
        Add the arc between *prev* and *node* to Route
        node $\leftarrow$ *prev*
    end
    Add Route to Routes
    endcluster $\leftarrow$ P(endcluster)
end

Output: Routes

2.4 Set Partitioning Heuristic

Instead of keeping the best achieved solution in memory, we store all routes obtained by local search and the split algorithm in a common solution pool. When the heuristic has performed the desired number of iterations, we typically have a few thousands of routes stored. The problem of choosing the optimal combination can be modelled as a set partitioning problem as presented in section 1.4. The problem is to find a combination of existing routes such that each cluster is visited exactly once. This problem can be modelled as a mixed-integer problem and then solved by a mixed integer programming solver such as CPLEX.

3 Results

In this section, we evaluate computationally our heuristic algorithm on a set of GVRP benchmark instances and a new set of DGVRP instances. The algorithm is coded using C/C++ and it is run on a 2.67 GHz Intel Core2 Quad Q9400 processor.

3.1 GVRP Instances

To test the heuristic algorithm, we apply it to instances created by Bektaş et al. [2]. These instances are derived from 79 existing CVRP instances. There are two GVRP versions of each CVRP instance, containing $\lceil n/\theta \rceil$ clusters where the parameter θ equals either 2 or 3. We call the resulting two instance sets T2 and T3, depending on the value of θ. The instances are further divided into subsets A, B, P, M and G. Subsets A, B and P contain over 20 instances each. The number of customers in these instances varies from 16 to 101. Subset M consists of four large instances with number of customers ranging from 101 up to 200. Finally, subset G contains one instance with 262 customers. The complete instance set is fairly heterogeneous. Some instances have the depot located at the center, and others at the margin. In some instances the clusters are dense and clearly separated from each other, and in other instances the clusters may be sparse and overlapping.

The algorithm was executed five times for each instance. The best and the average results are presented in Table 1 for instance set T2 and in Table 2 for the set T3. The first four columns in the tables describe the instance properties: 'Instance' reports the instance name, 'n' the number of customers, 'K' the number of clusters and 'Q' the vehicle capacity. The column 'm' reports the number of routes in our best solution and column 'T_{tot}' the length of the longest route in that solution. The best known upper bound is reported in column 'Ub^*'. These bounds are taken from either Hà et al. [12], Bektaş et al. [2] or Moccia et al. [11]. The bound is bolded if it is known to be the optimal solution cost. Column 'GAP^1' gives the average percentage distance from Ub^* of the best upper bound obtained over the five runs without using the set partitioning heuristic. Column 'GAP^2', instead, reports the percentage distance from Ub^* when the set partitioning heuristic is used. The best percentage gap over the five runs is reported in column 'GAP^*'. Columns 't^1' and 't^2' report the average computing time before and after solving the set partitioning model.

The heuristic algorithm is able to find the optimal objective value for all instances for which an optimal solution is known, except for one of the larger instances (M-n121-k7-C41). In total, only for six of the 158 instances the heuristic could not find the best known upper bound, and only one of these instances belongs to the smaller instance sets A, B and P.

Table 1: Heuristic algorithm results for the instances of set T2 with $K = \lceil n/2 \rceil$

Instance	n	K	Q	m	T_{tot}	Ub^*	GAP^1	GAP^2	GAP^*	t^1	t^2
A-n32-k5-C16	32	16	100	3	274	**508**	0.00	0.00	0.00	8.387	8.538
A-n33-k5-C17	33	17	100	3	192	**451**	0.00	0.00	0.00	9.331	9.487
A-n33-k6-C17	33	17	100	3	207	**465**	0.00	0.00	0.00	9.150	9.301
A-n34-k5-C17	34	17	100	3	225	**489**	0.00	0.00	0.00	9.378	9.521
A-n36-k5-C18	36	18	100	3	288	**502**	0.00	0.00	0.00	9.962	10.12
A-n37-k5-C19	37	19	100	3	236	**432**	0.00	0.00	0.00	10.06	10.19
A-n37-k6-C19	37	19	100	3	229	**584**	0.00	0.00	0.00	9.449	9.614
A-n38-k5-C19	38	19	100	3	205	**476**	0.00	0.00	0.00	9.896	10.35
A-n39-k5-C20	39	20	100	3	237	**557**	0.00	0.00	0.00	10.22	10.61
A-n39-k6-C20	39	20	100	3	234	**544**	0.00	0.00	0.00	10.26	10.45
A-n44-k6-C22	44	22	100	3	246	**608**	0.00	0.00	0.00	12.18	12.47
A-n45-k6-C23	45	23	100	4	202	**613**	0.00	0.00	0.00	12.95	13.19
A-n45-k7-C23	45	23	100	4	234	**674**	0.18	0.18	0.00	12.46	14.23
A-n46-k7-C23	46	23	100	4	216	**593**	0.00	0.00	0.00	12.80	12.99
A-n48-k7-C24	48	24	100	4	240	**667**	0.00	0.00	0.00	13.84	14.51
A-n53-k7-C27	53	27	100	4	197	**603**	0.00	0.00	0.00	16.70	16.98
A-n54-k7-C27	54	27	100	4	224	**690**	0.03	0.00	0.00	16.86	17.62
A-n55-k9-C28	55	28	100	5	200	**699**	0.65	0.00	0.00	17.33	17.75
A-n60-k9-C30	60	30	100	5	228	**769**	1.20	0.00	0.00	20.15	20.60
A-n61-k9-C31	61	31	100	5	165	**638**	0.00	0.00	0.00	21.61	22.28
A-n62-k8-C31	62	31	100	4	251	**740**	0.45	0.00	0.00	22.46	22.96
A-n63-k9-C32	63	32	100	5	267	**912**	0.71	0.00	0.00	23.69	24.13
A-n63-k10-C32	63	32	100	5	200	**801**	0.49	0.00	0.00	22.82	23.28
A-n64-k9-C32	64	32	100	5	252	**763**	0.34	0.00	0.00	23.24	23.53
A-n65-k9-C33	65	33	100	5	195	**682**	1.41	0.00	0.00	24.26	24.88
A-n69-k9-C35	69	35	100	5	224	**680**	0.00	0.00	0.00	26.87	27.96
A-n80-k10-C40	80	40	100	5	292	997	0.87	0.00	0.00	37.67	38.98
Average							0.234	0.007	0.000	16.07	16.54
B-n31-k5-C16	31	16	100	3	246	**441**	0.00	0.00	0.00	7.723	8.372
B-n34-k5-C17	34	17	100	3	215	**472**	0.00	0.00	0.00	8.712	8.892
B-n35-k5-C18	35	18	100	3	250	**626**	0.00	0.00	0.00	8.967	9.109
B-n38-k6-C19	38	19	100	3	225	**451**	0.00	0.00	0.00	9.921	10.65
B-n39-k5-C20	39	20	100	3	216	**357**	0.00	0.00	0.00	10.34	10.49
B-n41-k6-C21	41	21	100	3	192	**481**	0.00	0.00	0.00	11.17	11.34
B-n43-k6-C22	43	22	100	3	183	**483**	0.12	0.00	0.00	11.54	11.94
B-n44-k7-C22	44	22	100	4	208	**540**	0.00	0.00	0.00	11.87	12.20
B-n45-k5-C23	45	23	100	3	224	**497**	0.00	0.00	0.00	12.76	12.90
B-n45-k6-C23	45	23	100	4	146	**478**	0.00	0.00	0.00	12.55	13.36
B-n50-k7-C25	50	25	100	4	145	**449**	0.00	0.00	0.00	14.47	14.61
B-n50-k8-C25	50	25	100	5	323	**916**	0.82	0.13	0.00	14.39	19.43
B-n51-k7-C26	51	26	100	4	229	**651**	0.00	0.00	0.00	15.17	15.38
B-n52-k7-C26	52	26	100	4	160	**450**	0.00	0.00	0.00	15.51	15.70
B-n56-k7-C28	56	28	100	4	199	**486**	0.00	0.00	0.00	17.87	18.39
B-n57-k7-C29	57	29	100	4	204	**751**	0.00	0.00	0.00	18.92	19.98
B-n57-k9-C29	57	29	100	5	274	**942**	0.46	0.00	0.00	18.21	23.87
B-n63-k10-C32	63	32	100	5	226	**816**	0.00	0.00	0.00	23.01	23.38
B-n64-k9-C32	64	32	100	5	186	**509**	0.00	0.00	0.00	23.24	23.63
B-n66-k9-C33	66	33	100	5	268	**808**	1.29	0.00	0.00	24.56	25.27
B-n67-k10-C34	67	34	100	5	244	**673**	0.88	0.12	0.00	25.13	25.99
B-n68-k9-C34	68	34	100	5	196	**704**	0.98	0.00	0.00	25.56	26.06
B-n78-k10-C39	78	39	100	5	268	**803**	1.11	0.07	0.00	33.58	34.14
Average							0.246	0.014	0.000	16.31	17.18
P-n16-k8-C8	16	8	35	5	72	**239**	0.00	0.00	0.00	5.113	5.162
P-n19-k2-C10	19	10	160	2	119	**147**	0.00	0.00	0.00	5.139	5.194
P-n20-k2-C10	20	10	160	2	130	**154**	0.00	0.00	0.00	5.315	5.372
P-n21-k2-C11	21	11	160	2	140	**160**	0.00	0.00	0.00	5.443	5.659
P-n22-k2-C11	22	11	160	2	142	**162**	0.00	0.00	0.00	5.644	5.741
P-n22-k8-C11	22	11	3000	5	115	**314**	0.00	0.00	0.00	5.889	5.954
P-n23-k8-C12	23	12	40	5	88	**312**	0.00	0.00	0.00	6.082	6.157
P-n40-k5-C20	40	20	140	3	104	**294**	0.00	0.00	0.00	10.54	10.72
P-n45-k5-C23	45	23	150	3	143	**337**	0.00	0.00	0.00	12.42	12.62
P-n50-k7-C25	50	25	150	4	102	**353**	0.00	0.00	0.00	14.62	14.96
P-n50-k8-C25	50	25	120	5	111	**372**	0.00	0.00	0.00	14.38	14.60
P-n50-k10-C25	50	25	100	5	109	**410**	0.05	0.00	0.00	14.20	14.47
P-n51-k10-C26	51	26	80	6	97	**427**	0.00	0.00	0.00	14.75	14.94
P-n55-k7-C28	55	28	170	4	100	**361**	0.00	0.00	0.00	17.43	17.88
P-n55-k8-C28	55	28	160	4	100	**361**	0.00	0.00	0.00	17.32	17.64
P-n55-k10-C28	55	28	115	5	100	**415**	0.19	0.00	0.00	16.94	17.30
P-n55-k15-C28	55	28	70	9	90	**551**	0.00	0.00	0.00	16.37	16.59
P-n60-k10-C30	60	30	120	5	112	443	0.63	0.45	0.45	19.39	19.95
P-n60-k15-C30	60	30	80	8	107	**565**	0.00	0.00	0.00	18.48	18.72
P-n65-k10-C33	65	33	130	5	121	**487**	0.00	0.00	0.00	23.62	24.13
P-n70-k10-C35	70	35	135	5	107	**485**	0.00	0.00	0.00	27.32	27.79
P-n76-k4-C38	76	38	350	2	222	**383**	0.10	0.00	0.00	34.67	35.21
P-n76-k5-C38	76	38	280	3	154	**405**	0.00	0.00	0.00	34.18	34.83
P-n101-k4-C51	101	51	400	2	245	**455**	0.13	0.00	0.00	70.85	71.30
Average							0.046	0.019	0.019	17.34	17.62
G-n262-k25-C131	262	131	500	13	415	3249	4.43	1.68	1.10	1192.0	1204.4
M-n101-k10-C51	101	51	200	5	144	**542**	0.11	0.00	0.00	66.13	66.71
M-n121-k7-C61	121	61	200	4	224	719	2.78	1.15	0.28	109.3	110.9
M-n151-k12-C76	151	76	200	6	130	659	2.37	0.39	0.00	217.4	219.4
M-n200-k16-C100	200	100	200	9	114	789	5.11	0.88	0.25	495.4	499.3
Average							2.960	0.820	0.326	416.0	420.1

Table 2: Heuristic algorithm results for the instances of set T3 with $K = \lceil n/3 \rceil$

Instance	n	K	Q	m	T_{tot}	Ub^*	GAP^1	GAP^2	GAP^*	t^1	t^2
A-n32-k5-C11	32	11	100	2	256	**386**	0.00	0.00	0.00	7.359	7.449
A-n33-k5-C11	33	11	100	2	259	**315**	0.00	0.00	0.00	7.026	7.091
A-n33-k6-C11	33	11	100	2	260	**370**	0.00	0.00	0.00	7.031	7.135
A-n34-k5-C12	34	12	100	2	231	**419**	0.00	0.00	0.00	7.504	7.620
A-n36-k5-C12	36	12	100	2	313	**396**	0.00	0.00	0.00	7.577	7.655
A-n37-k5-C13	37	13	100	2	274	**347**	0.00	0.00	0.00	8.092	8.162
A-n37-k6-C13	37	13	100	2	268	**431**	0.00	0.00	0.00	7.883	7.997
A-n38-k5-C13	38	13	100	2	235	**367**	0.00	0.00	0.00	8.197	8.275
A-n39-k5-C13	39	13	100	2	256	**364**	0.00	0.00	0.00	8.323	8.415
A-n39-k6-C13	39	13	100	2	265	**403**	0.00	0.00	0.00	8.281	8.392
A-n44-k6-C15	44	15	100	3	290	**491**	0.00	0.00	0.00	9.986	10.01
A-n45-k6-C15	45	15	100	3	231	**474**	0.00	0.00	0.00	10.14	10.25
A-n45-k7-C15	45	15	100	3	218	**475**	0.00	0.00	0.00	10.18	10.30
A-n46-k7-C16	46	16	100	3	234	**462**	0.00	0.00	0.00	10.98	11.16
A-n48-k7-C16	48	16	100	3	244	**451**	0.00	0.00	0.00	11.77	11.91
A-n53-k7-C18	53	18	100	3	240	**440**	0.00	0.00	0.00	13.71	13.84
A-n54-k7-C18	54	18	100	3	235	**482**	0.00	0.00	0.00	13.58	13.73
A-n55-k9-C19	55	19	100	3	183	**473**	0.00	0.00	0.00	14.57	14.73
A-n60-k9-C20	60	20	100	3	250	**595**	0.00	0.00	0.00	16.14	16.35
A-n61-k9-C21	61	21	100	4	171	**473**	0.00	0.00	0.00	16.51	16.72
A-n62-k8-C21	62	21	100	3	280	**596**	0.36	0.00	0.00	17.23	17.54
A-n63-k9-C21	63	21	100	3	269	**642**	0.00	0.00	0.00	17.23	17.48
A-n63-k10-C21	63	21	100	4	225	**593**	0.34	0.07	0.00	17.20	17.42
A-n64-k9-C22	64	22	100	3	270	**536**	0.00	0.00	0.00	18.15	18.41
A-n65-k9-C22	65	22	100	3	201	**500**	0.00	0.00	0.00	18.32	18.48
A-n69-k9-C23	69	23	100	3	203	**520**	0.04	0.00	0.00	20.17	20.51
A-n80-k10-C27	80	27	100	4	275	**710**	0.00	0.00	0.00	27.56	28.55
Average							0.027	0.003	0.000	12.62	12.80
B-n31-k5-C11	31	11	100	2	248	**356**	0.00	0.00	0.00	6.636	6.718
B-n34-k5-C12	34	12	100	2	234	**369**	0.00	0.00	0.00	7.276	7.378
B-n35-k5-C12	35	12	100	2	270	**501**	0.00	0.00	0.00	7.377	7.477
B-n38-k6-C13	38	13	100	2	246	**370**	0.00	0.00	0.00	8.090	8.172
B-n39-k5-C13	39	13	100	2	210	**280**	0.00	0.00	0.00	8.013	8.080
B-n41-k6-C14	41	14	100	2	228	**407**	0.00	0.00	0.00	8.718	8.815
B-n43-k6-C15	43	15	100	2	181	**343**	0.00	0.00	0.00	9.472	9.718
B-n44-k7-C15	44	15	100	3	172	**395**	0.00	0.00	0.00	9.714	9.842
B-n45-k5-C15	45	15	100	2	256	**410**	0.00	0.00	0.00	9.660	9.741
B-n45-k6-C15	45	15	100	2	181	**336**	0.00	0.00	0.00	9.869	9.954
B-n50-k7-C17	50	17	100	3	186	**393**	0.00	0.00	0.00	11.48	11.61
B-n50-k8-C17	50	17	100	3	279	**598**	0.00	0.00	0.00	11.38	11.51
B-n51-k7-C17	51	17	100	3	204	**511**	0.00	0.00	0.00	11.66	11.77
B-n52-k7-C18	52	18	100	3	216	**359**	0.00	0.00	0.00	12.15	12.41
B-n56-k7-C19	56	19	100	3	272	**356**	0.00	0.00	0.00	13.82	13.96
B-n57-k7-C19	57	19	100	3	245	**558**	0.00	0.00	0.00	14.20	14.43
B-n57-k9-C19	57	19	100	3	270	**681**	0.03	0.03	0.00	14.01	14.66
B-n63-k10-C21	63	21	100	3	233	**599**	0.00	0.00	0.00	16.58	16.71
B-n64-k9-C22	64	22	100	4	184	**452**	0.00	0.00	0.00	17.70	17.88
B-n66-k9-C22	66	22	100	3	282	**609**	0.03	0.00	0.00	18.13	18.46
B-n67-k10-C23	67	23	100	4	256	**558**	0.00	0.00	0.00	19.64	19.86
B-n68-k9-C23	68	23	100	3	208	**523**	0.27	0.00	0.00	19.43	19.65
B-n78-k10-C26	78	26	100	4	204	**606**	0.00	0.00	0.00	25.29	25.51
Average							0.014	0.001	0.000	12.62	12.80
P-n16-k8-C6	16	6	35	4	65	**170**	0.00	0.00	0.00	4.980	5.023
P-n19-k2-C7	19	7	160	1	111	**111**	0.00	0.00	0.00	4.774	4.795
P-n20-k2-C7	20	7	160	1	117	**117**	0.00	0.00	0.00	4.842	4.861
P-n21-k2-C7	21	7	160	1	117	**117**	0.00	0.00	0.00	4.942	4.962
P-n22-k2-C8	22	8	160	1	111	**111**	0.00	0.00	0.00	5.013	5.036
P-n22-k8-C8	22	8	3000	4	98	**249**	0.00	0.00	0.00	5.585	5.632
P-n23-k8-C8	23	8	40	3	92	**174**	0.00	0.00	0.00	5.554	5.598
P-n40-k5-C14	40	14	140	2	111	**213**	0.00	0.00	0.00	8.500	8.675
P-n45-k5-C15	45	15	150	2	141	**238**	0.00	0.00	0.00	9.769	9.890
P-n50-k7-C17	50	17	150	3	108	**261**	0.00	0.00	0.00	11.44	11.59
P-n50-k8-C17	50	17	120	3	108	**262**	0.00	0.00	0.00	11.34	11.44
P-n50-k10-C17	50	17	100	4	107	**292**	0.00	0.00	0.00	11.64	11.74
P-n51-k10-C17	51	17	80	4	95	**309**	0.00	0.00	0.00	11.70	11.85
P-n55-k7-C19	55	19	170	3	110	**271**	0.00	0.00	0.00	13.19	13.42
P-n55-k8-C19	55	19	160	3	109	**274**	0.00	0.00	0.00	13.48	13.63
P-n55-k10-C19	55	19	115	4	100	**301**	0.00	0.00	0.00	13.35	13.52
P-n55-k15-C19	55	19	70	6	90	**378**	0.00	0.00	0.00	12.98	13.10
P-n60-k10-C20	60	20	120	4	115	**325**	0.00	0.00	0.00	15.12	15.29
P-n60-k15-C20	60	20	80	6	94	**374**	0.00	0.00	0.00	14.70	14.98
P-n65-k10-C22	65	22	130	4	125	**372**	0.00	0.00	0.00	18.17	18.50
P-n70-k10-C24	70	24	135	4	117	**385**	0.00	0.00	0.00	21.01	21.22
P-n76-k4-C26	76	26	350	2	172	**309**	0.00	0.00	0.00	25.07	25.27
P-n76-k5-C26	76	26	280	2	172	**309**	0.00	0.00	0.00	25.28	25.46
P-n101-k4-C34	101	34	400	2	262	**370**	0.00	0.00	0.00	47.19	47.44
Average							0.000	0.000	0.000	13.32	13.46
G-n262-k25-C88	262	88	500	9	386	2463	3.05	1.15	0.93	820.7	830.7
M-n101-k10-C34	101	34	200	4	143	**458**	0.00	0.00	0.00	45.80	46.37
M-n121-k7-C41	121	41	200	3	242	**527**	3.44	1.20	0.75	79.41	79.52
M-n151-k12-C51	151	51	200	4	134	483	0.33	0.00	0.00	155.3	155.8
M-n200-k16-C67	200	67	200	6	119	605	0.85	0.00	0.00	352.9	354.8
Average							1.534	0.470	0.336	290.8	293.4

Solving the set partitioning model improves the quality of solutions considerably. The difference can be as large as a few percentage units with large instances. After the set partitioning heuristic, all the solutions are within 2% of the best known upper bounds. The instances in the set T3 seem to be more easy for the heuristic, since only for one instance of subsets A, B and P the algorithm cannot find an optimal solution even before the set partitioning phase. With the set T2, this number is six instead. The set partitioning phase takes only a small fraction of time compared to the rest of the algorithm for both small and large instances.

The computing time on the instances of set T3 is consistently smaller than that required for the instances of the set T2. The difference becomes slightly greater as instance size increases. Overall, the computing time for the instances with up to one hundred customers is about half a minute, and the largest examined instance is solved within a reasonable time of 20 minutes.

The summary of the results for the small and medium instances A, B and P are reported in Table 3. Our results are compared with the heuristic algorithm results of Bektaş et al. [2], Moccia et al. [11], Afsar et al. [1] and Hà et al. [12]. The heuristic algorithm of Bektaş et al. [2] uses a large neighborhood search similar to that of Ropke and Pisinger [14] in a simulated annealing framework. Moccia et al. [11] use an incremental taboo search heuristic while Hà et al. [12] use an evolutionary local search heuristic similar to ours. Finally, the results of Afsar et al. [1] are obtained by two different versions of an iterated local search algorithm.

In Table 3, the column 'θ' reports the average cluster size in the instances. The column 'Succ' reports for each algorithm the number of instances within each subset for which the heuristic found the best known upper bound. For each subset, the best results are bolded. The column '$\bar{t}$' reports for each algorithm the average computing time of instances within a subset. In addition, for each algorithm we report the processor speed in GHz and the number of times the algorithm was executed to obtain the results. For Afsar et al. [1], 'Succ' reports the number of instances where at least one of their heuristics found the best known upper bound and $\bar{t}$ is calculated using the computing time of the faster algorithm finding the best solution.

Bektaş et al. [2] and Moccia et al. [11] consider the problem with a fixed number of vehicles while the other algorithms assume an unlimited fleet size. Because the problems are different, the results may not be comparable. However, there are in total only three instances in subsets A, B and P for which the optimal upper bound is affected by this additional constraint. These instances are left out of the comparison which is why the total number of instances is less for columns corresponding to Bektaş et al. [2] and Moccia et al. [11].

Over our five runs of the algorithm, there was only one instance (P-n60-k10-C30) in the subsets A, B and P for which we could not find the best known upper bound. Comparing the results with the others in terms of the number of best-known upper bounds found, our algorithm is able to find more than any other except Hà et al. [12] who found for all the instances their best known upper bounds. It must be noted, however, that none of the five individual runs of our algorithm produced equally good results.

The average computing times reported for the different algorithms in tables 3

and 4 cannot be directly compared due to the different computers used and since the specific processor models used by some of the authors are unknown. Using the information available, we try to estimate the relative CPU speed by using the CPU2006 bencmarks reported by the Standard Performance Evaluation Corporation (SPEC 2006) [15]. It seems that our 2.67 GHz Intel Core2 Quad Q9000 processor and the 3 GHz Intel Core2 Duo of Afsar et al. [1] are the fastest. The 2.4 GHz Intel Xeon of Hà et al. [12] seems to be about 20% slower than these two. The other processors appear to be similarly fast and about half as fast as the fastest processors. Because the processor types are not known, these relations are only the best estimate we can make and are not guaranteed to be correct.

Bektaş et al. have the fastest heuristic with an average computing time of about half a second. The algorithm of Afsar et al. [1] takes less than a second in average computing time but is performed with a faster computer. The computing times of Moccia et al. [11] are similar to ours with the average being about 15 seconds; taking the difference in processor speed into account, our algorithm is the slower one of the two. Finally, the algorithm of Hà et al. [12] is the slowest with an average computing time of about half a minute. The algorithm of Hà et al. [12] and the one presented in this paper yield good results but are also computationally the most extensive.

For the large instances of subsets M and G, a more detailed comparison is presented in Table 4. These instances were not considered by Afsar et al. [1]. Column 'θ' reports the average cluster size, while column 'Lb' reports the best known lower bound for each of the instances. If the instance has been solved to optimality, the lower bound is bolded. The remaining columns Ub, m and $\bar{t}$ report for each algorithm the best upper bound found, the number of vehicles used and the average computing time in CPU seconds. The best known upper bound values are bolded in the comparison section. Note that the results of Bektaş et al. [2] and Moccia et al. [11] are calculated assuming a fixed number of vehicles, so the best known result for the instance M-n200-k16-C100 is not feasible for their algorithm. With large instances, our algorithm does not match the results of the others; it found 5 best known upper bounds, while the other algorithms found at least seven. In addition, the computing time for our algorithm increases with the problem size much faster than for the other algorithms.

We are interested in comparing our results and computation times with those of Hà et al. [12], since their algorithm structure is very similar to ours. Our local search algorithm is much less sophisticated than theirs and our algorithm performs ≈30 times less split operations. Considering these facts, our algorithm seems to perform well on small to medium instances. On the larger instances of subsets M and G there were four instances where the algorithm of Hà et al. [12] obtained a better result than ours, and for the largest instance, G-n262-k25-C131, our algorithm found a better solution. Our algorithm is typically about twice as fast as the algorithm of Hà et al. [12], which is only faster with the smallest instances ($n \approx 20$) and the largest one. However, the difference in computing time is small compared to the difference in the number of iterations, which suggests that our implementation is not as efficient as possible.

Table 3: Summary and comparison of the algorithm performance on small and medium instances

		Thesis		Bektaş et al.		Moccia et al.		Hà et al.		Afsar et al.	
Subset	θ	Succ	$\bar{t}$	Succ	$\bar{t}$	Succ	$\bar{t}$	Succ	$\bar{t}$	Succ	$\bar{t}$
A	2	**27/27**	16.54	**25/25**	0.48	**25/25**	16.74	**27/27**	26.54	**27/27**	0.74
B	2	**23/23**	17.18	**23/23**	0.50	21/23	17.07	**23/23**	28.17	20/23	0.88
P	2	23/24	17.62	22/24	0.53	20/24	20.12	**24/24**	35.18	22/24	0.92
A	3	**27/27**	12.80	25/26	0.32	25/26	11.34	**27/27**	23.55	**27/27**	0.39
B	3	**23/23**	12.80	22/23	0.32	**23/23**	11.56	**23/23**	24.16	**23/23**	0.45
P	3	**24/24**	13.46	21/24	0.34	23/24	10.45	**24/24**	41.59	23/24	0.45
CPU speed (GHz) & Runs		2.67	5	2.4	1	1.83	1	2.4	1	3.0	2
Processor type		Intel Core2 Quad Q9400		AMD Opteron 250		Intel Core Duo		Intel Xeon		Intel Core2 Duo	

Table 4: Heuristic algorithm result comparison for big instances

			Thesis			Bektaş et al.			Moccia et al.			Hà et al.		
Instance	θ	Lb	Ub	m	$\bar{t}$	Ub	m	$\bar{t}$	Ub	m	$\bar{t}$	Ub	m	$\bar{t}$
M-n101-k10-C51	2	**542.00**	**542**	5	66.7	**542**	5	1.50	**542**	5	57.8	**542**	5	124.2
M-n121-k7-C61	2	705.84	721	4	110.9	**719**	4	2.15	720	4	98.3	**719**	4	234.3
M-n151-k12-C76	2	634.65	**659**	6	219.4	**659**	6	3.24	**659**	6	113.1	**659**	6	306.0
M-n200-k16-C100	2	752.14	791	9	499.3	791	8	5.34	805	8	158.7	**789**	9	454.0
G-n262-k25-C131	2	2945.02	3285	13	1204.4	**3249**	12	6.22	3319	12	193.6	3303	13	822.8
M-n101-k10-C34	3	**458.00**	**458**	4	46.4	**458**	4	0.86	**458**	4	36.8	**458**	4	152.3
M-n121-k7-C41	3	**527.00**	531	3	79.5	**527**	3	1.19	**527**	3	63.3	**527**	3	238.2
M-n151-k12-C51	3	474.32	**483**	4	155.8	**483**	4	1.89	**483**	4	85.5	**483**	4	434.3
M-n200-k16-C67	3	572.81	**605**	6	354.8	**605**	6	3.03	**605**	6	108.4	**605**	6	602.0
G-n262-k25-C88	3	2239.50	2486	9	830.7	2476	9	4.92	**2463**	9	134.3	2477	9	861.7
CPU speed (GHz) & Runs			2.67	5		2.4	1		1.83	1		2.4	1	
Processor			Intel Core2 Quad Q9400			AMD Opteron 250			Intel Core Duo			Intel Xeon		

3.2 Distance-constrained GVRP instances

Knowing that the heuristic algorithm performs well with the GVRP instances, we proceed to test its performance on the distance-constrained GVRP. Since no benchmark instances are available we create a set of test instances by modifying the GVRP instances of Bektaş et al. [2]. We choose a subset of 14 GVRP instances as diverse as possible which contains instances from subsets A, B, P and M with both values of θ and with size varying from 20 to 151 customers.

We have transformed each one of these instances into a DGVRP instance by taking the maximum length of a route in the best solution to the GVRP instance, and by setting the distance limit T to be smaller than this value. This ensures that the GVRP solution is infeasible. Therefore, the heuristic algorithm will find a new solution if the problem itself remains feasible after adding the distance constraint. We repeat the process three times for each instance to obtain increasingly tight distance constraints. Each time, we choose the new distance limit to be equal to the length of the longest route in the best solution of the instance obtained in the previous step minus 5 units. A smaller difference might lead to only small changes in the new solution, whereas a bigger decrease would make many instances infeasible. Overall, we obtain this way 3 DGVRP instances for each original GVRP instance.

Table 5: Test instances with limited distance

Instance	θ	Ub^*	t^*	m^*	T	m	Ub^H	GAP	t^H	Ub^{CX}	Lb^{CX}	t^{CX}
					125	2	**164**	0.00	6.67	**164**	**164**	3.20
P-n20-k2-C10	2	154	5.87	2	117	2	**166**	0.00	6.66	**166**	**166**	1.35
					99	2	**171**	0.00	6.76	**171**	**171**	4.72
					220	3	**515**	0.00	9.79	**515**	**515**	743
A-n34-k5-C17	2	489	8.91	3	199	3	**535**	0.00	9.91	**535**	**535**	834
					193	3	**549**	0.00	9.46	**549**	**549**	1599
					220	3	**520**	3.65	11.17	**520**	501	7200
B-n38-k6-C19	2	451	10.40	3	197	4	**579**	0.00	11.10	**579**	**579**	3680
					192	4	**663**	0.00	10.71	**663**	**663**	2880
					235	4	**671**	10.58	15.29	**671**	600	7200
A-n48-k7-C24	2	667	15.53	4	217	4	**686**	11.08	15.91	**686**	610	7200
					209	4	**714**	11.90	15.92	715	629	7200
					95	5	**403**	7.69	18.36	**403**	372	7200
P-n55-k8-C28	2	361	19.13	4	89	6	**451**	9.53	18.20	**451**	408	7200
					84	6	**451**	1.33	17.60	**451**	445	7200
					263	5	**816**	11.76	26.97	861	720	7200
B-n66-k9-C33	2	808	26.89	5	223	5	**855**	12.98	26.32	862	744	7200
					210	5	**907**	17.09	25.30	915	752	7200
					139	5	**543**	3.50	64.31	573	524	7200
M-n101-k10-C51	2	542	66.14	5	133	5	**562**	6.41	62.81	590	526	7200
					128	5	**571**	6.48	62.14	624	534	7200
					87	3	**187**	0.00	6.29	**187**	**187**	19.68
P-n23-k8-C8	3	174	6.14	3	80	3	**192**	0.00	6.24	**192**	**192**	26.41
					73	-	infeas.	-	-	infeas.	-	-
					263	3	**432**	7.64	8.66	**432**	399	7200
A-n37-k6-C13	3	431	8.52	2	242	2	**447**	0.00	8.71	**447**	**447**	2580
					230	3	**469**	0.00	8.80	**469**	**469**	1320
					176	3	**380**	0.00	10.53	**380**	**380**	394.1
B-n43-k6-C15	3	343	10.28	2	170	3	**420**	0.00	10.61	**420**	**420**	313.6
					163	3	**458**	0.00	10.50	**458**	**458**	120.4
					136	2	**239**	0.00	10.69	**239**	**239**	146.8
P-n45-k5-C15	3	238	10.34	2	125	2	**241**	0.00	10.91	**241**	**241**	100.0
					119	3	**257**	0.00	11.17	**257**	**257**	424.2
					211	3	**388**	0.00	13.69	**388**	**388**	2460
B-n52-k7-C18	3	359	13.22	3	182	3	**392**	0.00	13.71	**392**	**392**	1731
					148	4	**504**	0.00	13.60	**504**	**504**	1471
					264	3	**644**	14.44	17.93	**644**	551	7200
A-n63-k9-C21	3	642	17.28	3	253	3	**647**	13.76	18.31	**647**	558	7200
					237	4	**713**	9.26	18.31	727	647	7200
					129	5	**531**	18.46	143.7	597	433	7200
M-n151-k12-C51	3	483	159.5	4	122	5	**533**	16.51	143.9	584	445	7200
					116	5	**566**	16.78	143.7	590	471	7200

Using the heuristic algorithm, we only obtain upper bounds for the instances. We do not have any information about how far these bounds are from the optimal cost. Therefore, the DGVRP instances were also solved using CPLEX. For each instance, CPLEX was given a time limit of 2 hours. Only the smallest ones were solved to optimality, but the exact algorithm at least provides a lower bound in addition to the upper bound.

The results are reported in Table 5. For each one of the original GVRP instances, there are three DGVRP instances with different values of T. The first columns 'Instance' and 'θ' report the name of the original GVRP instance and its average cluster size. Column 'Ub^*' reports the best known upper bound of the original GVRP instance, which is optimal in all cases except for the instance M-n151-k12-C51. The computing time for solving the GVRP instance is reported in column 't^*', and the number of routes in the GVRP solution is reported in column m^*. The remaining columns contain results for the corresponding DGVRP instances solved by the heuristic. Columns 'T', 'm', 'Ub^H', 'GAP' and 't^H' report the maximum tour length, number of routes, best upper bound found by the heuristic, optimality gap percentage with respect to the lower bound and computing time of the heuristic. The last three columns 'Ub^{CX}', 'Lb^{CX}' and 't^{CX}' report the upper and lower bounds provided by CPLEX and the total computing time of CPLEX.

In all of the 41 feasible instances, the heuristic algorithm finds upper bounds that are at least as good as those provided by CPLEX. 21 of these upper bounds are proved to be optimal, since CPLEX can solve these instances within the 2-hour time limit. In the cases where the optimal solution is not found, the percentage gap between the upper and lower bound is typically very large. Even with medium-size instances ($n \approx 50$) the gap can reach 10%, and with the big instances the gap can be almost 20%.

There is little difference between the computing times of the heuristic for the GVRP and DGVRP instances. Tightening the maximum route length also seems to have little effect on the computing time of the heuristic. Depending on the instance, the computing time can slightly decrease or increase as the maximum route length is being restricted. In the cases where CPLEX succeeded in solving the DGVRP instances to optimality, Table 5 shows that the computation time is usually shorter as the value of T becomes smaller.

An example of three DGVRP instances derived from a GVRP one is presented in Figure 5. The top-left figure corresponds to the routes in the optimal solution for the GVRP instance B-n52-k7-C18. The three other figures show the solutions for the three distance-constrained DGVRP instances that are derived from it. These solutions are optimal, as is shown in Table 5. The limit on the routes' duration is reported as T, and the optimal solution value as z. We can see how the total length of the routes increases as the length of the longest route is more tightly limited. In the bottom-right solution, an additional route must be added so that all clusters can be visited while satisfying the distance constraint.

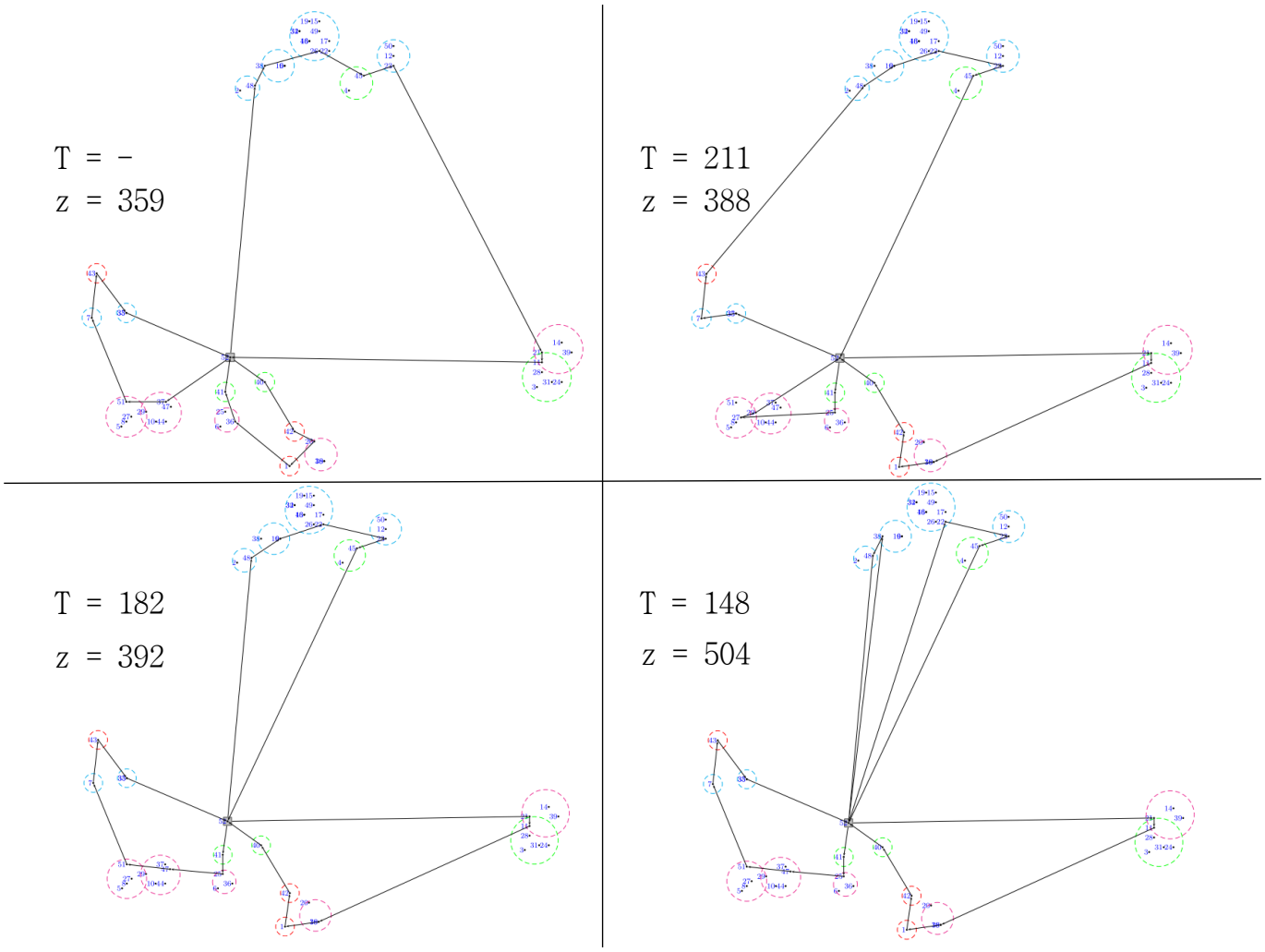

Figure 5: An illustration of a GVRP instance and three corresponding DGVRP instances and their optimal solutions

4 Conclusions

We have presented a mathematical formulation for the distance-constrained GVRP and a heuristic algorithm. The algorithm includes a new set partitioning heuristic, which appears to be effective regarding both computing time and solution quality. We have computationally tested our algorithm both on a set of GVRP instances created by Bektaş et al. [2] and on a newly generated set of DGVRP instances. The results for the GVRP instances show that the upper bounds found by our algorithm match the best known upper bounds on all of the 158 tested instances but six. A new set of 41 test instances for the DGVRP is derived from 14 of the GVRP benchmark instances. These instances were solved using both the heuristic algorithm and the commercial solver CPLEX. The upper bounds provided by the heuristic were equal or better than those found by CPLEX in all cases.

Ideas for future research include improving the lower bounds for DGVRP instances, in order to properly test the performance of the heuristic algorithm in the precence of distance constraints. Our algorithm only considers the case where the distance and time matrices are equal. Modifying the algorithm to allow different distance and time values between two customers would result in a more generic model to use in real-world applications. There are many problems that can be transformed into a GVRP, and the performance of the heuristic algorithm could be tested on these problems as well. By adding distance constraints, it is possible to model an even more wide variety of problems.

References

[1] Afsar H. M., Prins C., Santos A. C., 2014. Exact and heuristic algorithms for solving the generalized vehicle routing problem with flexible fleet size. *International Transactions in Operational Research 21, 153–175.*

[2] Bektaş T., Erdoğan G., Røpke S., 2011. Formulations and branch-and-cut algorithms for the generalized vehicle routing problem. *Transportation Science 45(3), 299-316*

[3] Baldacci R., Bartolini E., Laporte G., 2010. Some applications of the generalized vehicle routing problem. *Journal of the Operational Research Society 61, 1072-1077*

[4] Dantzig G. B., Ramser J. H., 1959. The truck dispatching problem. *Management Science 6 (1), 80-91.*

[5] Baldacci R., Hadjiconstantinou E., Mingozzi A., 2004. An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations Research 52(5), 723-738.*

[6] Toth P., Vigo D., 2014. Vehicle Routing: Problems, methods, and applications, Second edition. *MOS-SIAM Series on Optimization.*

[7] Laporte G., Nobert Y., Desrochers M., 1985. Optimal Routing under capacity and distance restrictions. *Operations Research 33, 1050-1073.*

[8] De Franceschi R., Fischetti M., Toth P., 2006. A new ILP-based refinement heuristic for vehicle routing problems. *Mathematical Programming, Series B 105, 471-499.*

[9] Fischetti M., Salazar-González J., Toth P., 1997. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research 45(3), 378-394.*

[10] Ghiani G., Improta G., 2000. An efficient transformation of the generalized vehicle routing problem. *European Journal of Operational Research 122(1), 11–17.*

[11] Moccia L., Cordeau J., Laporte G., 2012. An incremental tabu search heuristic for the generalized vehicle routing problem with time windows. *Journal of the Operational Research Society 63, 232-244.*

[12] Hà M. H., Bostel N., Langevin A., Rousseau L., 2013. An exact algorithm and a metaheuristic for the generalized vehicle routing problem with flexible fleet size. *Computers & Operations Research 43, 9-19.*

[13] Bektaş T., Lysgaard J., 2015. Optimal vehicle routing with lower and upper bounds on route durations. *Wiley Online Library.*

[14] Ropke S., Pisinger D., 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science 40(4), 455-472.*

[15] SPEC (2011) SPEC CPU2006 Results. Standard Performance Evaluation Corporation. Accessed November 25, 2011. http://www.spec.org/cpu2006/results/index.html

A Thesis summary in Finnish / Yhteenveto

Kandidaatintyössä ”The Distance-Constrained Generalized Vehicle Routing Problem” tutkittiin etäisyysrajoitettua yleistettyä ajoneuvoreititysongelmaa. Kyseessä on kombinatorisen optimoinnin piiriin kuuluva matemaattinen ongelma, joka on pitkälle yleistetty. Siksi moni vastaavanlainen ongelma voidaan ilmaista tutkitun ongelman muodossa. Työssä esitettiin ongelman matemaattinen muotoilu sekä heuristinen algoritmi, joka etsii kokeilemalla ongelmaan mahdollisimman hyviä käypiä ratkaisuja kohtuullisessa ajassa. Algoritmia testattiin valmiiden testitehtävien avulla jättämällä etäisyysrajoitus huomiotta. Rajoitettua tapausta varten luotiin sen sijaan uusi sarja tehtäviä.

Yleistetty ajoneuvoreititysongelma on yleistys kauppamatkustajan ongelmasta, joka puolestaan on kuuluisimpia ja tutkituimpia optimoinnin alan sovelluksia. Ongelmassa kauppamatkustajan tulee suunnitella mahdollisimman lyhyt myyntireitti siten, että hän käy jokaisen asiakkaan luona tasan kerran ja palaa lopuksi lähtöpisteeseensä. Helposti ymmärrettävä tehtävä on kuitenkin laskennallisesti vaikea. Kun asiakkaiden määrää nostetaan useisiin tuhansiin, mahdollisten reittien määrä muuttuu niin suureksi, että parhaan vaihtoehdon laskeminen on työlästä tehokkaallekin tietokoneelle.

Yleistetyssä ajoneuvoreititysongelmassa on kolme merkittävää eroa perinteiseen kauppamatkustajan ongelmaan verrattuna. Ensimmäiseksi kauppiaalla on käytössään kuorma-autolaivue, jolle hänen täytyy suunnitella reitit. Reittejä voidaan siis suunnitella enemmän kuin yksi. Toiseksi kullakin asiakkaalla on oma kysyntä kauppiaan tuotteelle. Kuorma-autojen tilavuus on rajallinen, joten kukin auto voi palvella vain sellaista asiakasjoukkoa, jonka yhteenlaskettu kysyntä ei ylitä sen kapasiteettia. Kolmanneksi asiakkaat on tässä ongelmassa ryhmitelty suuremmiksi joukoiksi. Näissä ryppäissä voi olla yksi tai useampi asiakas, ja jokainen asiakas kuuluu vain yhteen ryppääseen. Samaan joukkoon kuuluvat asiakkaat voivat välittää kauppiaan tuotetta toisilleen. Siispä kussakin ryppäässä vieraillaan vain yhden asiakkaan luona.

Kun reittien pituudelle asetetaan yläraja, saadaan lopulta etäisyysrajoitettu ajoneuvoreititysongelma, jota kandidaatintyössä tarkasteltiin. Kyseistä ongelmaa ei ollut entuudestaan tutkittu, mutta sillä on lukuisia käytännön sovelluksia. Kuorma-autolaivueen esimerkissä etäisyysrajoitus voi mallintaa vaikkapa työpäivän maksimikestoa, jolloin saatava malli on realistisempi. Toinen esimerkkisovellus liittyy tarvikkeiden jakamiseen luonnonkatastrofien jälkeen. Maanjäristyksen seurauksena tieverkko voi olla vahingoittunut, jolloin alueella sijaitsevat kylät muodostavat toisistaan eristyneitä ryppäitä. Jokaiselle ryppäälle tulee toimittaa lentokoneilla avustuskuorma, mutta kunkin reitin pituutta rajoittaa polttoaineen riittävyys.

Kandidaatintyössä tarkasteltu monimutkainen ongelma on työläs ratkaistavaksi. Parhaan mahdollisen ratkaisun etsintään kului tietokoneella yli kaksi tuntia jo 40–50 asiakkaan tapauksessa. Käytännöllisemmäksi katsottiin hyvien ratkaisujen etsiminen kokeilemalla eli heuristisesti. Työn pääpainoksi valittiinkin heuristisen algoritmin kuvailu ja toteutus.

Algoritmiin sisältyi monta pientä funktiota. Ensin aloitusalgoritmi loi alkuratkaisun, jota muokkaamalla parannuksia etsittiin. Reittien luomiseksi käytettiin

kahta erilaista tapaa: joko jokaista rypästä varten luotiin oma reittinsä, tai yhdestä ryppäästä siirryttiin arpomalla jompaankumpaan sen lähimmistä naapureista.

Lähietsintäfunktion tehtävänä oli puolestaan parantaa olemassa olevia reittejä tekemällä yksinkertaisia liikkeitä. Liikkeisiin kuului muun muassa ryppään sijoittaminen uuteen paikkaan ratkaisussa sekä kahden ryppään paikan vaihtaminen. Lisäksi lähietsinnässä varmistettiin, etteivät ratkaisureitit koskaan ylitä itseään.

Koko heuristisen algoritmin tärkeimpiä osia oli jakajafunktio. Funktiolle syötettiin jättiläisreitti, joka saatiin liittämällä senhetkisen ratkaisun kaikki reitit yhteen. Jättiläisreitissä siis kuljetaan kaikki ryppäät läpi palaamatta kertaakaan lähtöpisteeseen. Dynaamista optimointia käyttäen tällainen reitti pystyttiin jakamaan optimaalisella tavalla kapasiteetti- ja etäisyysrajoitukset täyttäviksi reiteiksi. Jakaja-algoritmi on tehokas, mutta koko ongelman parhaan mahdollisen ratkaisun löytäminen vaatisi kaikkien rypäsjärjestysten tutkimista.

Heuristinen algoritmi perustui toistoon, jossa yhtä reittiä parannettiin vuorollaan ja lyhin löydetty reitti pidettiin muistissa. Ensin aloitusratkaisua parannettiin lähietsintäfunktiolla; tämän jälkeen reitit yhdistettiin jättiläisreitiksi, jota muokattiin satunnaisesti. Samaa jättiläisreittiä muokattiin ja jaettiin useaan otteeseen. Jakajafunktion palauttamaa ratkaisua yritettiin vielä parantaa lähietsintäfunktiolla, jonka jälkeen prosessi aloitettiin alusta luomalla uusi aloitusreitti.

Uutena piirteenä heuristisessa algoritmissa oli aivan lopuksi suoritettava ositusvaihe. Hyvälaatuisia ratkaisuja tallennettiin koko ratkaisuprosessin ajan säilöön aina lähietsintä- ja jakajafunktioiden tekemien muokkausten jälkeen. Ositusvaiheessa näistä reiteistä koottiin kaupallisen sovelluksen avulla paras ratkaisu, jossa jokaisessa ryppäässä vierailtiin tasan yhden kerran.

Heuristisen algoritmin suorituksen laatua mitattiin käyttämällä testitehtäviä, jotka on luotu yleistettyä ajoneuvoreititysongelmaa varten. Näitä tehtäviä oli 158, ja niiden koko vaihteli 16 asiakkaasta yli 260:een. Tehtävät on suunniteltu monimuotoisiksi esimerkiksi lähtöpisteen suhteellisen sijainnin ja ryppäiden läpimitan suhteen. Myös tarvittavien reittien määrä vaihteli kahdesta yli kymmeneen.

Algoritmi ajettiin läpi viidesti, ja parasta ratkaisua vertailtiin muissa tutkimuksissa saavutettuihin arvoihin. Sataan asiakkaaseen asti heuristinen algoritmi pystyi löytämään parhaan tunnetun ratkaisun kaikissa paitsi yhdessä tapauksessa. Samoja testitehtäviä oli käytetty neljässä muussa tutkimuksessa, ja näistä vain yhdessä oli saatu parempi tulos. Ositusvaiheen lisääminen algoritmiin paransi tuloksia merkittävästi, eikä vaihe vaatinut juurikaan lisää aikaa muuhun algoritmiin verrattuna. Tyypillinen laskenta-aika pienille ja keskikokoisille tehtäville oli kymmenestä sekunnista minuuttiin, mikä oli muihin toteutuksiin verrattuna hitaimmasta päästä. Yli sadan asiakkaan tehtävissä tulokset eivät puolestaan aivan yltäneet samalle tasolle kuin muiden tutkimusten tulokset. Samoin laskenta-aika oli suhteessa vielä hitaampi suurten tehtävien kohdalla. Kaikista suurimpienkin tehtävien laskentaan kului vielä kuitenkin kohtuullinen 20 minuutin aika.

Etäisyysrajoitettua tapausta varten luotiin oma sarja testitehtäviä muokkaamalla valmiiden tehtävien osajoukkoa. Näitä tehtäviä valittiin yhteensä neljätoista kaikista kokoluokista. Kutakin alkuperäistä tehtävää vastasi kolme uutta rajoitettua tehtävää, joissa etäisyysrajoitusta muutettiin aina tiukemmaksi. Jotta heuristisen

algoritmin tuloksia voitaisiin arvioida, tehtävät myös ratkaistiin optimiinsa kaupallisen sovelluksen avulla. Kaikissa tapauksissa sovellus ei löytänyt kahden tunnin aikarajan sisällä optimivastausta, mutta palautti sen sijaan parhaalle tulokselle ylä- ja alarajat. Algoritmi löysi kaikille etäisyysrajoitetuille tehtäville ratkaisun, joka oli vähintään yhtä hyvä kuin kaupallisen sovelluksen antama optimitulos tai yläraja. Etäisyysrajoitteiden lisääminen ei vaikuttanut merkittävästi algoritmin tarvitsemaan laskenta-aikaan.

Kandidaatintyössä pystyttiin luomaan toimiva ja kohtalaisen kilpailukykyinen heuristinen algoritmi etäisyysrajoitetun yleistetyn ajoneuvoreititysongelman ratkaisemiseksi. Erityisesti uuden ositusvaiheen tuominen osaksi algoritmia paransi tuloksia huomattavasti. Algoritmin tarkempi arviointi etäisyysrajoitusten läsnä ollessa vaatisi kuitenkin vielä tarkempaa tutkimusta muun muassa parempien alarajojen määrittämiseksi.

Toisena kehitysmahdollisuutena on jakajafunktio: nyt funktio vaati olettamuksen, että kahden pisteen välinen etäisyys on aina yhtä suuri kuin pisteiden välin kulkemiseen kuluva aika. Mikäli tämän oletuksen saisi poistettua, algoritmilla olisi vielä laajemmat sovellusmahdollisuudet; esimerkiksi kuorma-autojen reittejä suunnitellessa kahden kaupungin välinen etäisyys voi olla suurikin, mutta suora moottoritieyhteys mahdollistaisi vierailun kummassakin työpäivän aikana.

Laajat mallinnus- ja sovellusmahdollisuudet tekevätkin aiemmin tutkimattomasta etäisyysrajoitetusta yleistetystä ajoneuvoreititysongelmasta mielenkiintoisen jatkotutkimuksia ajatellen.