VRPDiv: A Divide and Conquer Framework for Large Vehicle Routing Problems

Radu Mariescu-Istodor[†]

Microsoft, Timișoara, Romania
University of Eastern Finland, Joensuu, Finland
radum@cs.uef.fi

Alexandru Cristian

Microsoft, Timișoara, Romania, alcristi@microsoft.com

Mihai Negrea

Microsoft, Timișoara, Romania, mihai.negrea@microsoft.com

Peiwei Cao

Microsoft, Seattle, USA, peiweic@microsoft.com

ABSTRACT

The Vehicle Routing Problem (VRP) is an NP hard problem where we need to optimize itineraries for agents to visit multiple targets. When considering real-world travel (road-network topology, speed limits and traffic), modern VRP *solvers* can only process small instances with a few hundred targets. We propose a framework (VRPDiv) that can scale any *solver* to support larger VRP instances with up to ten thousand targets (10k) by dividing them into smaller clusters. VRPDiv supports the multiple VRP scenarios and contains a pool of clustering algorithms from which it chooses the ideal one depending on properties of the instance. VRPDiv assigns agents based on cluster demand and targets compatibility (i.e. realizable time windows and capacity limitations). We incorporate the framework into the Bing Maps Multi-Itinerary Optimization (MIO)[1] online service. This architecture allows MIO to scale up from solving instances with a few hundred to over 10k targets in under 10 minutes. We evaluate our framework on public datasets and publish a new dataset ourselves, as large enough instances supporting real-world travel were impossible to find. We investigate multiple clustering methods and show that choosing the correct one is critical with differences of up to 60% in quality. We compare with relevant baselines and report a 40% improvement in target allocation and a 9.8% improvement in itinerary durations. We compare with existing scores and report an average delta of 10%, with lower values (<5%) in instances with low workload (few targets per agent), which are acceptable for an online service.

CCS CONCEPTS

Theory of computation → Design and analysis of algorithms → **Massively parallel algorithms**
Computing methodologies → Artificial intelligence → **Multi-agent planning**

KEYWORDS

VRP problem, Divide and Conquer, Multi-Itinerary Optimization, Clustering

1 Introduction

Many real world applications need to generate itineraries for a set of agents (workers) that need to visit a given set of targets (geo-locations) [1]. In practice, these applications include logistics, transportation of

[1] https://www.microsoft.com/en-us/maps/multi-itinerary-optimization

goods and maintenance, to name a few [2]. As illustrated by multiple reviews [3, 4, 5], itinerary optimization makes up an extensively studied field. In the beginning, the goal was to minimize the total travel-distance or travel-time [1] in an attempt to lower the cost of gas or working hours, respectively. Since then, new objectives emerged such as minimizing exposure to sunlight [6], maximizing safety of travel [7] or minimizing CO2 levels [8, 9]. The size of instances that need solving (number of targets and agents) is also increasing by the day, with FedEx, for example, averaging over 6 million deliveries per day[2]. Now, due to the COVID-19 pandemic, couriers struggle more than ever to meet increased demands from hospitals [10], supermarkets and those who self-isolate at home and order food and merchandise online.

The simplest variant of the problem is the classical Traveling Salesperson Problem (TSP) [11] where a single agent needs to visit all locations. Typically the agent must also return to the starting point, however, an open loop formulation is also encountered [12]. TSP is NP hard; meaning that exact algorithms [13, 50] require exponential time and for large instances consisting of many targets they are not useful in practice. For practical reasons when solving large instances, heuristic algorithms are preferred using different strategies such as: Tabu Search, Ant Colony Optimization, Simulated Annealing and Evolutionary methods to name a few [5]. These heuristic solutions provide a quick suboptimal solution that is often of acceptable quality in practice. Some approximation algorithms [48, 49] even guarantee that the quality will be below a certain threshold. The downside is that even though heuristic algorithms do not slow down exponentially as the number of targets increases, the quality of the solution decreases instead due to an increasingly large search space which renders local operators slow [14] or unable to escape local optimum [12].

TSPs do not necessarily refer to real world travel and can appear in other applications like computer wiring, crystallography, robot control, drilling of circuit boards and chronological sequencing [15]. In these scenarios, despite the problem being very difficult in nature, modern heuristic algorithms can now solve large instances with millions of targets within 0.1% of the optimum in hours [15, 16]. The main reason for such high performance is that in these applications the distances can simply be calculated using mathematical formulas (typically the Euclidean distance). When traveling in the real world, however, this is not the case. While it is possible to compute straight distances "as the crow flies" using the Great Circle Distance, these are only suitable when the agent can travel in a straight line between the targets (e.g. airborne drones). Otherwise travel-distances and travel-times depend on the road-network and the traffic in the region.

It is possible to compute travel-distances using shortest path algorithms such as Floyd, Dijkstra or A* [17] on road-networks. Travel-times can also be obtained with these same algorithms by setting a fixed speed, or the individual road speed limits (when available), or predictive traffic information (also when available) to compute the fastest path instead of the shortest one [46, 47]. When generating itineraries, the typical approach is to precompute the values for every pair of targets and store them in a so-called *distance matrix* to be used as a lookup-table during the optimization process. Obtaining large matrices is not trivial and storing them requires quadratic memory, therefore, research on this type of data is more limited and public datasets seem to contain instances with only a few hundred targets at most [18]. As a result, researchers aiming to solve large instances ignore the intricacies of real world travel and resort to mathematical formulas like the Great Circle Distance or, alternatively, project the coordinates in a 2D space (e.g. UTM) where the more efficient Euclidean distance formula can be used. In this way, researchers can focus on the optimization process itself and can attempt to solve larger instances like this one[3] containing 1,904,711 world-wide city locations or this one[4] consisting of 1,437,195 buildings in Finland. The choice to simplify the distance calculations impacts the feasibility of generated itineraries [14]; for example, targets that appear to be nearby may be hours away

[2] https://www.statista.com/statistics/878354/fedex-express-total-average-daily-packages

[3] http://www.math.uwaterloo.ca/tsp/world

[4] https://cs.uef.fi/sipu/santa

as a consequence of natural borders like lakes or rivers [19]. In this paper, we propose a framework that can solve large instances using realistic travel information. We experiment with instances of up to 10,000 targets, which, to our knowledge, are the largest yet attempted using realistic travel.

When TSPs involve real world travel they usually go by a different name: the Vehicle Routing Problem (VRP) [1]. VRP is a generalization of TSP that considers a multitude of scenarios that appear in real-life and we formally define it as having:

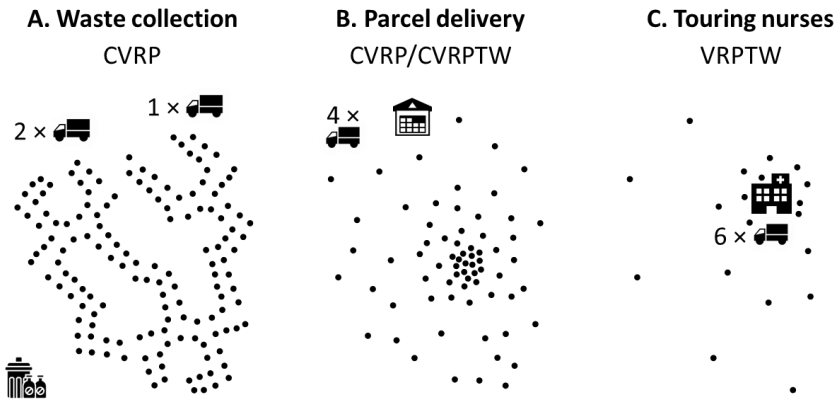
a set of targets $T = \{ t_i, i = 1 \ldots N \}$,
a set of agents $A = \{ a_j, j = 1 \ldots M \}$,
an optional set of depots $D = \{ d_k, k = 0 \ldots P \}$ and the following possible properties (constraints):

	Notations:	
1. Working hours: agents are available in specified time intervals:	$a_j^{\text{sTime}}, a_j^{\text{eTime}}$	$j = 1 \ldots M$
2. Focal points: agents start and end their work at specified locations:	$a_j^{\text{sLoc}}, a_j^{\text{eLoc}}$	$j = 1 \ldots M$
3. Capacities: agents have limited space or volume:	a_j^{capacity}	$j = 1 \ldots M$
3. Locations: the locations of the targets:	t_i^{location}	$i = 1 \ldots N$
4. Time windows: targets can only be visited during specified times:	$t_i^{\text{sTime}}, t_i^{\text{eTime}}$	$i = 1 \ldots N$
5. Dwell times: agents must spend a specified amount of time at each target:	t_i^{dwell}	$i = 1 \ldots N$
6. Quantity: targets have specified pick up or delivery amount:	t_i^{quantity}	$i = 1 \ldots N$

Because time windows can be quite restrictive, it may not always be possible to allocate all targets to the available agents. Therefore, an optimized solution is one that generates itineraries for the given agents that maximize the number of allocated targets while minimizing the travel cost (typically distance or time, depending on the application) and taking into consideration all specified parameters. Depending on which of these parameters are present many real-world scenarios can be defined (see Figure 1).

One common scenario is waste collection (see Figure 1 **A**) also known as CVRP (Capacitated Vehicle Routing Problem). Here, the agents usually start and end at a garage where the vehicles are stored. There could be more such focal points if several companies work in the same region. The depots are places where the waste is deposited and they are usually eccentric: somewhere at the edge or even outside the city. In this scenario, dwell times are very small: minutes or even seconds, sometimes the workers can even collect waste while the truck is moving at a slow pace and in this case dwell times may not be specified at all. The target locations are usually all the buildings in a region because everyone is expected to produce some amount of waste. Even though the number of targets is quite high, the number of agents is not necessarily in proportion, as waste does not need to be collected every day. The targets are usually static and optimization can be done rarely and the resulting itineraries will be useful for a long time. Time windows are usually lacking in this scenario, but capacity is limited to the size of typical garbage trucks. The expected quantity is known per target and varies relative to the size of the residential building.

In Figure 1 **B** we have a delivery scenario which is also referred to as a CVRP or CVRPTW (Capacitated Vehicle Routing Problem with Time Windows). Although similar to **A**, this is different from waste collection in several ways. Second, the target locations change on a daily basis and optimization is needed more often. Second, parcels need to be delivered as soon as possible, so the number of agents is usually higher. Capacity varies more: from small (mailbags carried by people) to large (cargo trucks or even ships). The targets can be individual customers (on a city level) or warehouses (at a country level). Time windows may be present when customers expect home delivery (CVRPTW). Dwell times can vary from long: when refilling a warehouse to short: the time it takes to fill mailboxes or unload one item and receive a confirmation signature. The depots are usually eccentric (a warehouse), but they can be central as well, like a post office.



	A. Waste collection CVRP	B. Parcel delivery CVRP/CVRPTW	C. Touring nurses VRPTW
Optimization needed	Rarely	Daily	Real-time
Capacity / Quantities	Yes	Yes	-
Time windows	-	Possibly	Yes
Depots	Eccentric	Usually eccentric	Central or none existant
Dwell times	Very Short	Short or Long	Medium
Start/End points	Garrage	Warehouse	Hospital
Focal points	One or more	Usually one	Usually One
Workload (targets per agent)	High	Medium	Low

Figure 1: Three VRP scenarios that are common in the real world.

In Figure 1 **C** we have a touring nurses' scenario commonly known as VRPTW (Vehicle Routing Problem with Time Windows). Here capacity is not an issue as nurses have all the necessary equipment always at hand. It is typical that the nurses start at the depot (i.e. hospital) in the morning for supplies, but sometimes this is not a requirement. Dwell times are much longer as patient treatment is usually time consuming, reducing the workload to only a few visits per day. Emergencies can occur at any time which force nurses to make an immediate change of plans and itineraries need to be recomputed on the fly to accommodate all other patients otherwise the consequences can be even fatal. The hospital and focal points are usually central because patients with serious health conditions tend to live near the hospitals and hospital locations, themselves, are often decided so as to minimize the travel-times of patients as this was shown to correlate with better health [51]. A more common scenario similar to the touring nurses is performing maintenance (e.g. electrical work). There, the workload is usually higher but the need for real-time rescheduling is not paramount in that case.

Researchers typically focus on a single scenario like CVRP [20, 21] or VRPTW [22, 23]. Our proposed method is general enough that it can be applied to solve all above scenarios and more: any scenario that can be defined using the aforementioned properties. In doing so, we also provide an online service[5] where customers can customize parameters to fit their exact scenario. Having said that, to keep the scope of our work manageable there are a few scenarios that we do not target in this work like Pickup and Delivery (also known in literature as the problem of finding the optimal sequenced route [44, 45]), where some targets must be visited before others like in the case of food home delivery. These constraints would currently be broken by our clustering strategy and we are still studying how to satisfy both. We also do not support partial

[5] https://www.microsoft.com/en-us/maps/multi-itinerary-optimization

quantities, where agents can visit a target multiple times picking up some items each time; doing so would mean the additional burden of having to solve another NP hard problem, like the Knapsack or Bin Packing problems [17]. We consider this as further work.

2 Proposed Framework

In this paper, we propose VRPDiv, a divide and conquer framework for handling large VRP instances (see Figure 2). VRPDiv first obtains travel-time estimates between the targets, which are useful in all upcoming steps. Then it chooses a clustering objective function based on properties of the given instance. We will later see in the experimental section that some methods perform better than others depending on the properties of the instance. The instance is then divided into independent clusters, which are valid VRP instances that are small enough to be solved efficiently using a common *Solver*. In the experiments, we use the MIO *Solver* [14] because it supports all previously mentioned scenarios, however, our framework will work using any suitable algorithm. We, therefore, make abstraction from it and refer to it as the *Solver* throughout this paper. Finally, it combines the independent solutions to form the complete one.

In the upcoming sections, we first explain VRPDiv in detail and then perform a set of experiments to demonstrate its behavior in practice.

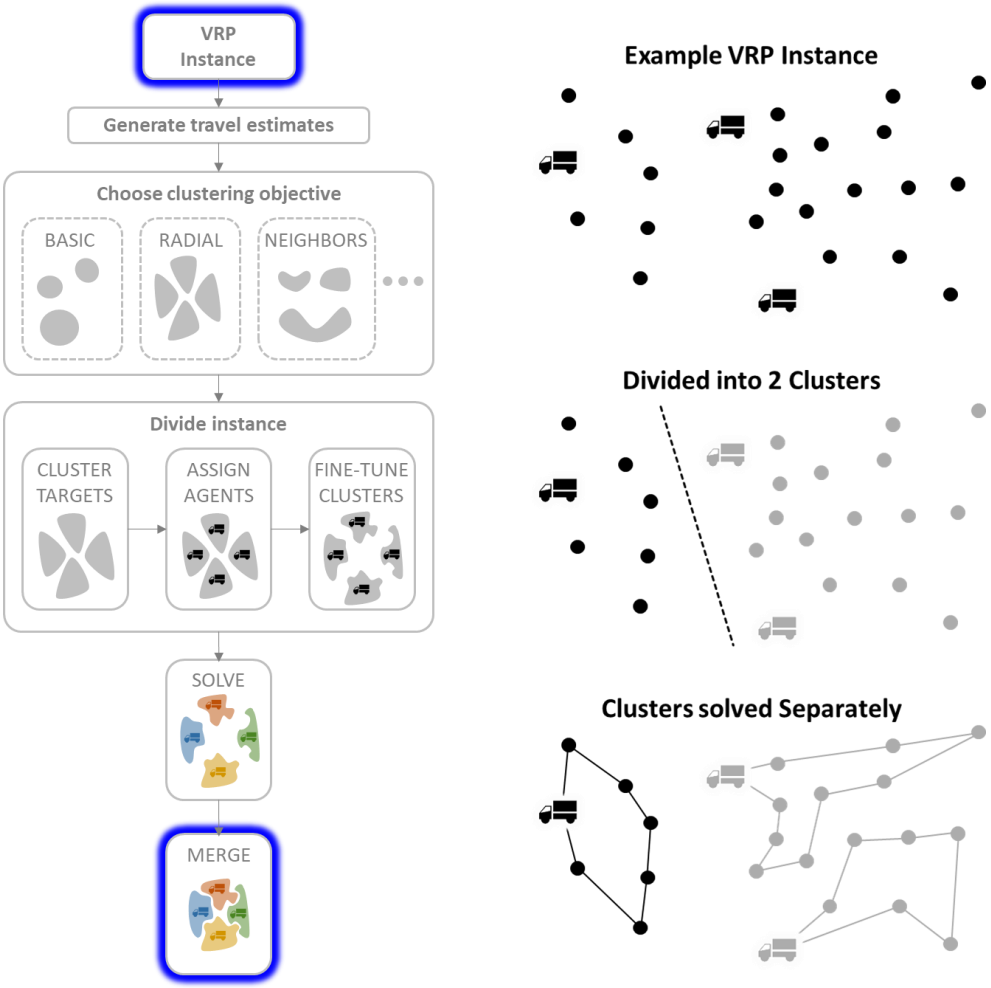


Figure 2: Components of the VRPDiv framework (left) and an example instance divided into clusters (right). For simplicity, the agent start and end locations here are the same and represented by the truck icons.

3 Generating travel estimates

Two physically nearby locations on the map are not necessarily easy to reach in practice when separated by something like a railroad, a highway or a natural barrier such as a river or a lake, which need avoiding in some way. Similarly, two far away locations may be easily reachable if there is good infrastructure like a high-speed road connecting them. Furthermore, this cost (typically distance or time) may not be symmetric as one-way roads often have an impact. Knowing the cost between the agents and the targets is a necessary to solve the problem. To aid in this, Application Programming Interfaces (APIs) such as the Bing Maps Routes API[6], the Google Maps Directions API[7] or open source alternatives like the Open Source Routing Machine (OSRM)[8] can be used to obtain the shortest or fastest route between two locations and consequently, its cost: travel-distance or travel-time. One of these values is typically needed and not the routes themselves. These API requests are customizable, and users can specify, for example, the type of vehicle and whether to use estimated traffic information or not. These are essential to setup properly, otherwise generated itineraries may include portions where the agents cannot pass (physically or legally) or may get stuck in traffic.

If values are needed for a set of locations, services like Bing Maps, for example, offer a Distance Matrix API[9], which retrieves all pairwise values with a single request. However, if the number of locations is too large, using the API as such is very expensive. If we want to process 10k locations, the response size alone for a matrix of 10k × 10k containing both the distance and time (two integer values) is approximately 1.5 GB. This increases to 1 TBs if predictive traffic information is also desired (a full week represented by quarter hour intervals where distance and time is provided for each 15 - minute time interval) [14]. In addition to this, there are also computational concerns. Even though sophisticated methods based on contraction hierarchies [24, 25] and Dijkstra's algorithm [17] are used, a matrix of that size is expected to generate in many hours on state of the art cloud architecture consuming large computing resources, thus becoming prohibitively expensive to purchase. The Bing Maps Distance Matrix API experiences a large number of requests and is, therefore, limited to 200 × 200 locations to the public, however, internally, we can exceed the limit when needed.

With all these complications it is easy to see why the compromise to estimate travel-distances using the Great Circle Distance (GCD) is preferred by researchers and if the travel-time is required instead, the travel-distance is converted into time using a fixed speed. This method, however, is almost always an underestimation which results in many unallocated targets in practice: the agent may not reach a target in time because the travel-distance is longer than that in a straight line. To solve this, GCD distances may be weighted but then the agents will likely reach many targets too early and need to wait. In [14] it has been shown that optimizing using road-network information significantly improves the quality of the result compared to optimizing using GCD.

One important reason why VRPDiv is a divide and conquer framework is to produce clusters that are small enough to be able to compute such distance matrices (one per cluster). In that way, agents assigned to each cluster will be able to accomplish the itineraries in practice by following navigation instructions. Dividing the instance into clusters, however, should consider the travel information as well and this is a problem because, as previously stated, we cannot obtain these values for large instances. In theory, division could also

[6] https://docs.microsoft.com/en-us/bingmaps/rest-services/routes

[7] https://developers.google.com/maps/documentation/directions/overview

[8] http://project-osrm.org

[9] https://docs.microsoft.com/en-us/bingmaps/rest-services/routes/calculate-a-distance-matrix

be done by clustering using GCD as a distance estimate, however, using a better estimate that considers the road-network produces a more meaningful division as we will see in Section 7.3. We use the overhead graph method [19] to estimate the travel-distance or travel-time. This method creates a graph by choosing *K* nodes at representative locations from the complete set. We obtain these nodes by applying K-Means [26, 27, 28] equipped with GCD on the set of all locations and select the nearest locations (from the set) to the resulting centroids as nodes. We do not use the centroids as such because they may be located in inaccessible locations, like in the middle of a lake. We then call the Distance Matrix API to obtain travel-costs between the *K* nodes and use the result to generate the overhead graph: a complete graph where each link represents the overhead from any one node to another, calculated as:

$$\text{overhead}(\boldsymbol{p}_1, \boldsymbol{p}_2) = \frac{\text{time}(\boldsymbol{p}_1, \boldsymbol{p}_2)}{\text{GCD}(\boldsymbol{p}_1, \boldsymbol{p}_2)} \qquad (1)$$

where $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ are two locations and time ($\boldsymbol{p}_1$, $\boldsymbol{p}_2$) is the true travel-time of the fastest path from $\boldsymbol{p}_1$ to $\boldsymbol{p}_2$ obtained with the Distance Matrix API. Once the graph is available, we can predict the travel-time between two locations ($\boldsymbol{p}_1$ and $\boldsymbol{p}_2$) by computing first the GCD and then scaling it using the overhead of their nearest nodes $NN(\boldsymbol{p}_1)$ and $NN(\boldsymbol{p}_2)$ stored in the overhead graph:

$$\widehat{\text{time}}(\boldsymbol{p}_1, \boldsymbol{p}_2) = \text{GCD}(\boldsymbol{p}_1, \boldsymbol{p}_2) \cdot \text{Graph}[NN(\boldsymbol{p}_1), NN(\boldsymbol{p}_2)] \qquad (2)$$

Using the overhead graph, we could estimate travel-distance [19] or any other property in the same way if the application demands it; however, in this paper we focus on time to avoid redundant writing and because time is more useful when formulating a measure for cluster demand (see Section 5). Figure 3 shows an example of an overhead graph and how the travel-time estimation is made. We note that even though the graph is small, it already encodes meaningful information like the long travel-times around the estuary.

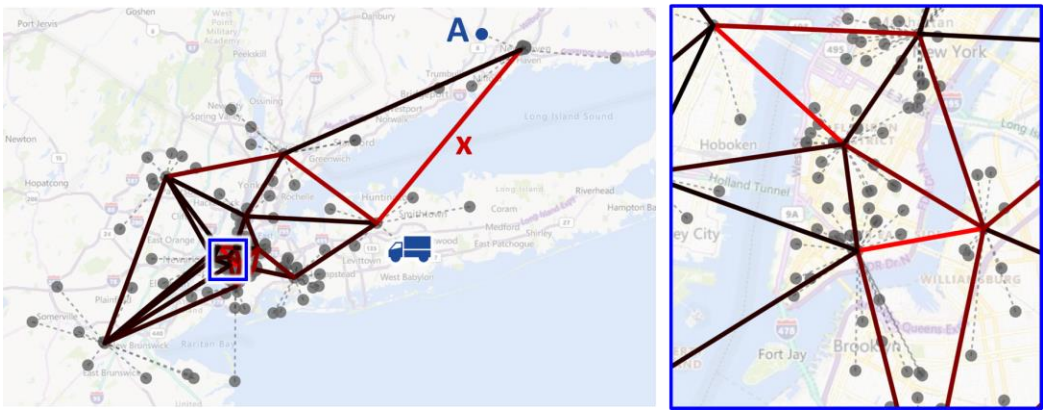


Figure 3: A set of locations in and around New York and a small overhead graph with 16 nodes. We only display few nearest neighbor links to make the figure readable. Each link is colored with increasing amounts of red in proportion to the overhead values. The nearest nodes to each target are shown using dotted lines. The travel cost between the agent location (truck) and target A is estimated by multiplying their GCD with the overhead value X.

We can now estimate the travel-time between any two locations using Equation 2 at the cost of a nearest-neighbor search. However, estimates can be made in constant time between any two locations that are part of the original set because we already have a mapping to their nearest node given by the K-Means *partitioning* step. In all upcoming steps we only use locations from the original set, therefore, all travel-time estimations are computed in O(1) time.

In practice, given that computing distance matrices is expensive, one must carefully design the size of the overhead graph *G* such that it (1) represents well the perturbations in the road network and (2) is practically or efficiently computable. In practice, we recommend experimenting with *G* in the [50, 200] interval.

4 Clustering targets

Before we talk about clustering a4lgorithms, we perform a natural division and validity check based on agent-to-target compatibility. In a valid instance, every target $t_i \in T$ must be compatible with at least one agent $a_j \in A$ and vice-versa. A compatible pair (t_i, a_j) must satisfy these conditions:

$$a_j^{\text{capacity}} \geq t_i^{\text{quantity}} \tag{3}$$

$$a_j^{\text{sTime}} + \widehat{time}\left(a_j^{\text{sLoc}}, t_i^{\text{location}}\right) + t_i^{\text{dwell}} < t_i^{\text{eTime}} \tag{4}$$

$$a_j^{\text{sTime}} + \widehat{time}\left(a_j^{\text{sLoc}}, t_i^{\text{location}}\right) + t_i^{\text{dwell}} + \widehat{time}\left(t_i^{\text{location}} + a_j^{\text{eLoc}}\right) < a_j^{\text{eTime}} \tag{5}$$

$$t_i^{\text{sTime}} + t_i^{\text{dwell}} < a_j^{\text{eTime}} \tag{6}$$

where t_i^{sTime}, t_i^{eTime} and a_j^{sTime}, a_j^{eTime} are the start and end times between which a target can be visited and when an agent can work, respectively; t_i^{dwell} is the duration the agent spends at the target; a_j^{sLoc} and a_j^{eLoc} are the locations where the agent leaves from and ends at; and t_i^{location} is the location of the target. All target quantities must fit in at least one vehicle that can reach there in time (Equation 3). We must note that using here estimated travel-time between locations may result in false positives, however, the travel estimates are expected to be reliable and false positives, if present, will be borderline cases that are likely to cause complications as well.

Next, we divide using compatibility as follows. We generate a compatibility graph, which is a bipartite graph with agents in one set and targets in another and define links between every compatible agent-target pair. Isolated nodes in this graph imply the problem instance is invalid (see Figure 4). In addition to this, disconnected components may form which we detect using depth-first search [17]. If such components exist, it makes no sense to consider solving them together and we separate them already now (a first stage of clustering). Otherwise, the clustering and the Solver will consider incompatible pairs, essentially wasting time. In practice, these situations happen rarely: when a business operates in multiple locations and presents a single VRP instance with combined data; the locations also need to be relatively far from each other.

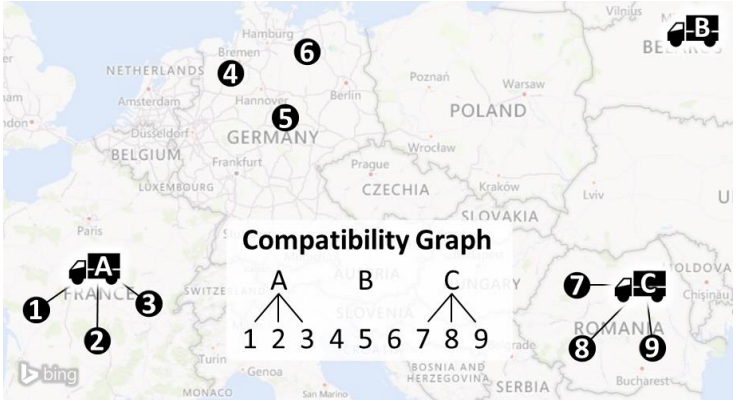


Figure 4: An instance with 3 agents and 9 targets in different countries. The compatibility graph shows two connected components: one in France and one in Romania and four isolated nodes rendering agent B useless and targets 4, 5 and 6 unreachable.

4.1 Clustering algorithms

Clustering is the process of grouping objects so that those within a group are more similar to each other than to objects in other groups. We aim to form groups based on the estimated travel-times between the targets. These clusters will have agents assigned to them to form smaller VRP instances to be solved independently (see Figure 5).

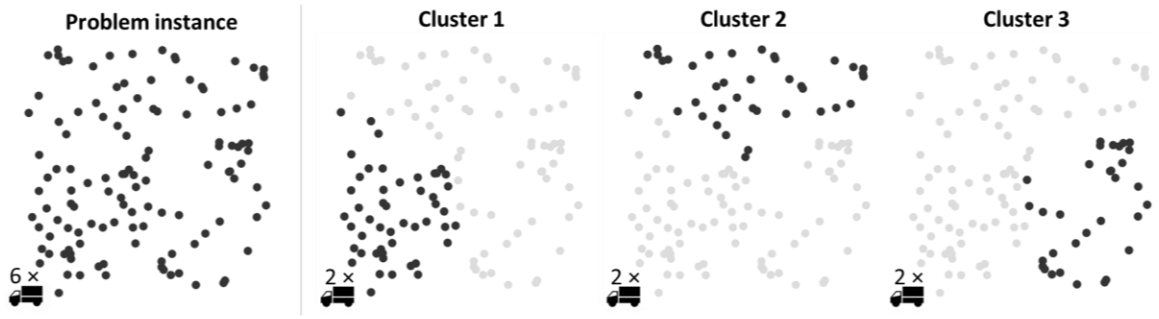


Figure 5: A VRP instance with 120 targets and 6 agents being clustered into three smaller instances where 2 agents are assigned to each. All agents start and end in the same focal point.

Clustering algorithms typically require the number of clusters K beforehand. We, therefore must discuss the possible values for K. In practice, K is upper bounded by the number of agents M (Equation 7). Otherwise, some clusters will lack agents and not constitute viable VRP instances. To obtain a lower bound for K we need to analyze the behavior of the VRP *Solver* we plan to use. Typically, *solvers* are extensively tested and in the case of optimal ones, we know the expected time t they need to process a given instance. The values for t grow exponentially with respect to the number of targets, therefore, only small instances can be solved in a reasonable time. Likewise, for heuristic *solvers* we typically know the expected quality of the solution after optimizing an instance with size S for a specific amount of time t. For practical purposes, it is common to fix the processing time to something reasonable like t = 1 minute [14], which means that we can predict the solution quality using only S and decide the maximum size of our clusters. A cluster larger than this may take too long to process (optimal solvers) or produce a solution of unacceptable quality (heuristic solvers). The run time and the error you are talking about is a worst-case error for exact solvers and maybe an expected error for heuristic solvers. But it is not true that they are forced to produce garbage just because the instance is large.

$$K_{max} = M \qquad (7)$$

$$K_{min} = \min\left(\left\lceil \frac{N}{S} \right\rceil, M\right) \qquad (8)$$

To analyze further, let us assume the simple case where the clusters are balanced [29]. In this case, we can set $K = \lceil N / S \rceil$, where N is the total number of targets. However, if the value becomes greater than M, it will exceed the upper bound and we need to compromise by setting $K = M$ in that case (Equation 8). This would not be ideal as the clusters will contain more than S items, causing the *Solver* to produce lower quality solutions, but the alternative is to have clusters that lack agents, which guarantees unallocated targets, so we consider this lower bound as a soft constraint while aiming at a value between K_{min} and K_{max}.

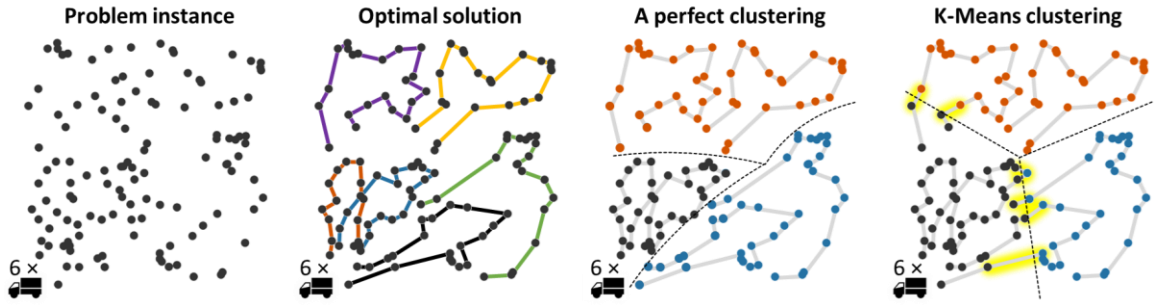


Figure 6: A VRP instance with 120 targets and 6 agents and its optimal solution (the links to the focal point are not drawn). A perfect clustering exists where the paths do not intersect the cluster borders. A typical K-Means clustering interferes with 4 of the 6 optimal paths.

When dividing an instance, we expect a suboptimal division: some targets that belong together may be separated which will interfere with optimal itineraries (see Figure 6). While a perfect clustering is theoretically possible, it is difficult to achieve because clustering is also NP-hard. Having less clusters is preferable because less cluster borders mean a lower probability to make mistakes. We therefore, have a preference towards K_{min}. We next explain several clustering algorithms.

4.1.1 K-Means

K-means algorithm [26, 27, 28] groups objects into K clusters by minimizing the total squared error between every point and its nearest cluster centroid (mean). The algorithm starts with a random set of centroids and iterates two steps until convergence: Partitioning and Centroid update. In the partitioning step, the nearest points to each centroid are grouped together; in the centroid update step the centroids are relocated to be at the center of mass of each partition. This second step naturally optimizes the sum of squared Euclidean distances, which is not ideal in the case of realistic travel. We will address this in section 4.3. Moreover, in K-Means we do not have control over the size of the clusters, which we prefer not to exceed S. Therefore, we next introduce two variants that have control over this size in some way.

4.1.2 Balanced K-Means

Balanced K-Means [29] is an algorithm that guarantees equal sized clusters (±1). It is essentially K-Means with a modified partitioning step formulated as a pairing problem and solved optimally using the Hungarian algorithm [30]. The algorithm assigns N / K slots to each centroid and forms a complete bipartite graph between every slot (set 1) and every location (set 2) where the edge weight is the physical distance, squared. Then, it does optimal pairing between these two sets (see Figure 7).

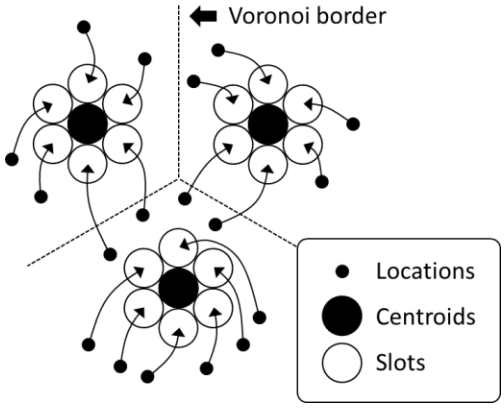


Figure 7: Assigning locations to centroids via cluster slots.

We run this algorithm with $K = K_{min}$ as it guarantees that resulting clusters will have sizes equal to S. There are two drawbacks to this algorithm. First, the structure is somewhat affected by the strict balancing constraint, meaning that naturally small clusters are forced to become larger and vice-versa. Another more significant drawback is the execution time. The Hungarian algorithm is $O(N^3)$ and it governs the overall time complexity of the method. Such complexity means it is too slow in practice for over 1k locations. Nevertheless, this method is the one that guarantees the fewest clusters, which generates the fewest cluster borders reducing the probability of interference with optimal itineraries.

4.1.3 X-Means

X-Means [31] is a divisive algorithm that attempts to find the number of clusters within a specified range. It alternates between two steps:

1. K-Means
2. Cluster Splitting

The first step is conventional K-Means. In the second step, a split is made based on some criterion. The Bayesian or Akaike Information Criterion are investigated in [31], however, for our needs, we modify this to

use the cluster size as the splitting criterion. We begin with $K = K_{min}$ and repeatedly perform the two steps until all cluster sizes are at most S or until $K = K_{max}$. This will usually produce more clusters than the minimum necessary, but the algorithm is fast and can be applied on over 10k locations. One drawback is that the number of clusters tends to be well above k_{min} and can even surpass $2 \cdot k_{min}$. In an attempt to reduce the number of clusters while keeping their size below S, we introduce a balancing step and refer to the new algorithm as X-Means +balancing.

4.1.4 X-Means +balancing

We modify X-Means by introducing a third step: centroid swap where we move one centroid from one location to another. This step is inspired from [32], however, instead of considering successful relocations those that reduce the mean square error; we consider those that reduce the maximum cluster size. We perform systematic relocations from clusters with fewer items to those with too many items. Doing so will cause the larger clusters to divide and the small ones to merge to their neighbors. In the experiments, we will demonstrate the effect this modification has on the number of clusters and the positive impact on the quality of the itineraries.

4.1.5 SLINK

All aforementioned algorithms optimize the total squared error which is the most Basic objective function used in clustering. Single-linkage, however, is an agglomerative clustering method [33], which optimizes the distance to Neighbors. This usually gets less attention because it tends to produce long chains that are often seen as unwanted artifacts in many applications. In our case, however, chains suggest that a path exists where agents can follow while visiting targets along the way which is ideal. The naïve algorithm begins with every target in its own cluster and proceeds to merge the nearest two clusters at each step. The distance between two clusters is the distance between the nearest targets from the two clusters. The process continues until everything is in one cluster and the result is a dendrogram, which is later used to generate any number of clusters. This naïve implementation is $O(N^3)$ and not useful in practice for more than 1k targets. To be able to efficiently process over 10k targets we use the SLINK algorithm [34] which generates the dendrogram more efficiently in $O(N^2)$ time.

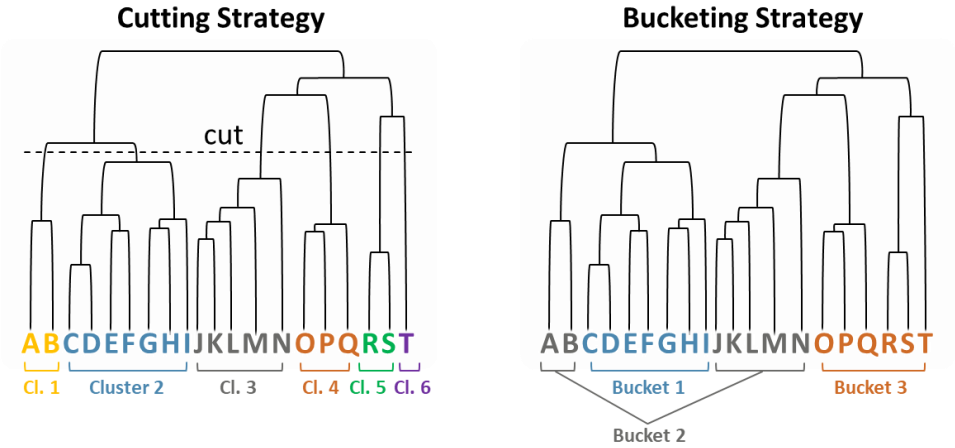


Figure 8: Cutting a dendrogram so that the size of the clusters does not exceed S = 7 (left). The bucketing strategy forms less clusters (right), however, bucket 2 contains two groups that are not near each other and this is not ideal.

The strategy of cutting the dendrogram to produce actual clusters is not trivial as large clusters tend to form rapidly in high density areas like the city center. This fast-growing cluster quickly reaches size S. Cutting at this level typically causes a very large number of clusters, many of which are very small (see Figure 8). The number of clusters typically produced at this step far exceeds the number of agents, which is unacceptable. To avoid this, we use a bucketing strategy to form the clusters. We move from the lowest level towards the top and when merging we verify that the size of the resulting cluster does not exceed S. If it does, the two branches are bucketed and considered separately from now on. Later, two nearby buckets can merge if the

result has less than *S* items. This does not always produce an ideal result, as seen in Figure 8 where items in bucket 2 merge even though bucket 1 is in-between. Such artifacts do not happen very often in practice when *S* is larger and when they do, they often do not impact itineraries in a negative way because the *Solver* will make it so that different agents are sent to separate groups. Noticeable problems only occur when the number of agents is limited, and then they need to traverse the in-between cluster without visiting targets with no apparent explanation.

4.2 Objective functions

Hypothetically, if we know the optimal solution for a given VRP instance, it would be possible to generate a perfect clustering: one that has zero interference with the optimal itineraries of the agents. This would have the potential to produce the optimal result assuming the *Solver* finds the optimal solution within each cluster (see Figure 6). Then the key question to ask is what objective functions interferes the least with the optimal paths of agents, meaning that if it interferes, it allows for minimal reconfiguration that still produces a good result. Many features affect how the optimal result looks like. For example: low capacities force the agents to visit a depot more often, small time windows force the clusters to overlap and so on. Therefore, instead of trying to give a single answer, we propose to choose from a pool of complementary objective functions and to select which one to use depending on the properties of the instance and past experience.

Our framework currently supports four objective functions (see Figure 9). The first is the Basic one found in literature: to minimize the total squared error to cluster centroids which is what K-Means and its variants (X-Means, X-Means +balancing and Balanced K-Means) achieve in practice. When using this Basic objective the clusters are compact and targets are easily reachable within the cluster. One drawback is that targets around the borders may be separated even if really close to each other. Another drawback is that the depot locations and the focal points are not considered meaning that some clusters may form far away causing some agents to travel long distances without visiting targets along the way. A second objective is to optimize the distance to the Neighbors, which the SLINK algorithm accomplishes naturally. This forms long strains which are good because agents can visit many targets along the way. Moreover, items in different clusters are now guaranteed to be relatively far from each other. Clusters may still form far away from the focal points and their sizes are influenced by the density making them disproportionate at times and agent assignation more difficult.

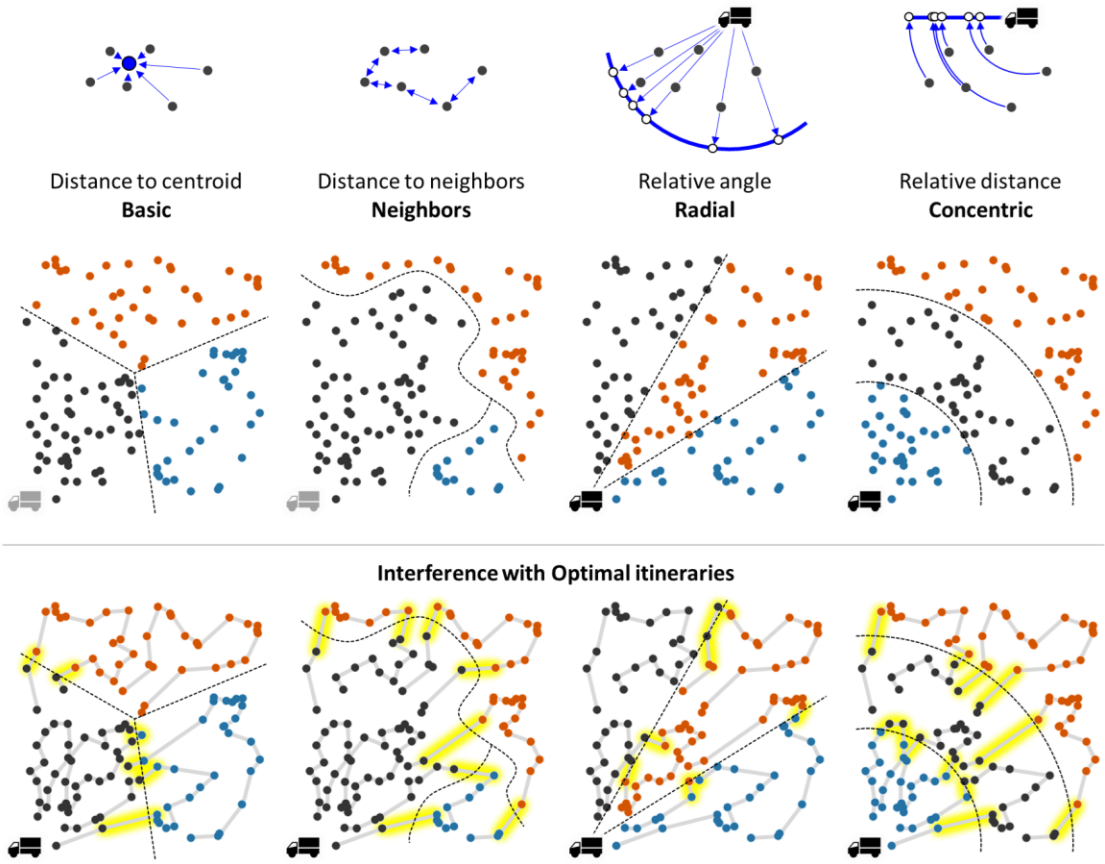


Figure 9: Clusterings using different objective functions and the interference with optimal itineraries.

We propose two new objectives Radial and Concentric, which take into consideration special locations like the focal points and the depots (if present). These objectives minimize the relative angles and relative distances to these special locations. We refer to the first objective as Radial because it tends to create long radially shaped clusters. This enables every agent to visit targets all the way to the outskirts and back; the disadvantage is that the rays become thin as the number of clusters increases forcing partial solutions to resemble straight lines, which reduces the usefulness of the *Solver*. We refer to the second objective as Concentric because it tends to form concentric bands. These can be useful when far away targets are connected through good infrastructure like a highway around the city. The downside, however, is that when the number of clusters increases, the bands become thin, forcing partial solutions to resemble circles, which reduces the usefulness of the Solver.

To optimize these two objectives, we can use any K-Means variant after projecting the target locations relative to the special locations (focal points or depots). When a single such location exists, the projections are simple one-dimensional vectors consisting of just one angle value (Radial) or one distance value (Concentric). When clustering angles, we must use the angle difference as the distance function to properly handle values around 0 and 2π, otherwise, an unnatural border will appear in-between. We can also project with respect to multiple locations F by generating F-dimensional vectors consisting of F relative angles (Radial) or F relative distances (Concentric) to each of the F locations. The effect of multiple focal points is shown in Figure 10. In the case of Radial, we can see clusters shaped to accommodate travel between the focal points. These are useful when agent start and end locations differ. The Concentric objective produces clusters which are at similar distances from each focal point. Both these objectives generate thin clusters in regions with higher density. These are not ideal as they pretty much impose how the agents must travel, rendering the *Solver* essentially useless. We found that combining Radial and Concentric into a Hybrid

objective helps to reduce thin clusters while preserving the individual benefits of the two objectives (see Figure 10). We obtain this Hybrid objective by concatenating the relative angle and distance values to form a 2*F*-dimensional vector and normalize the distance in the scale [0, π] to be comparable with angles using the same formula.

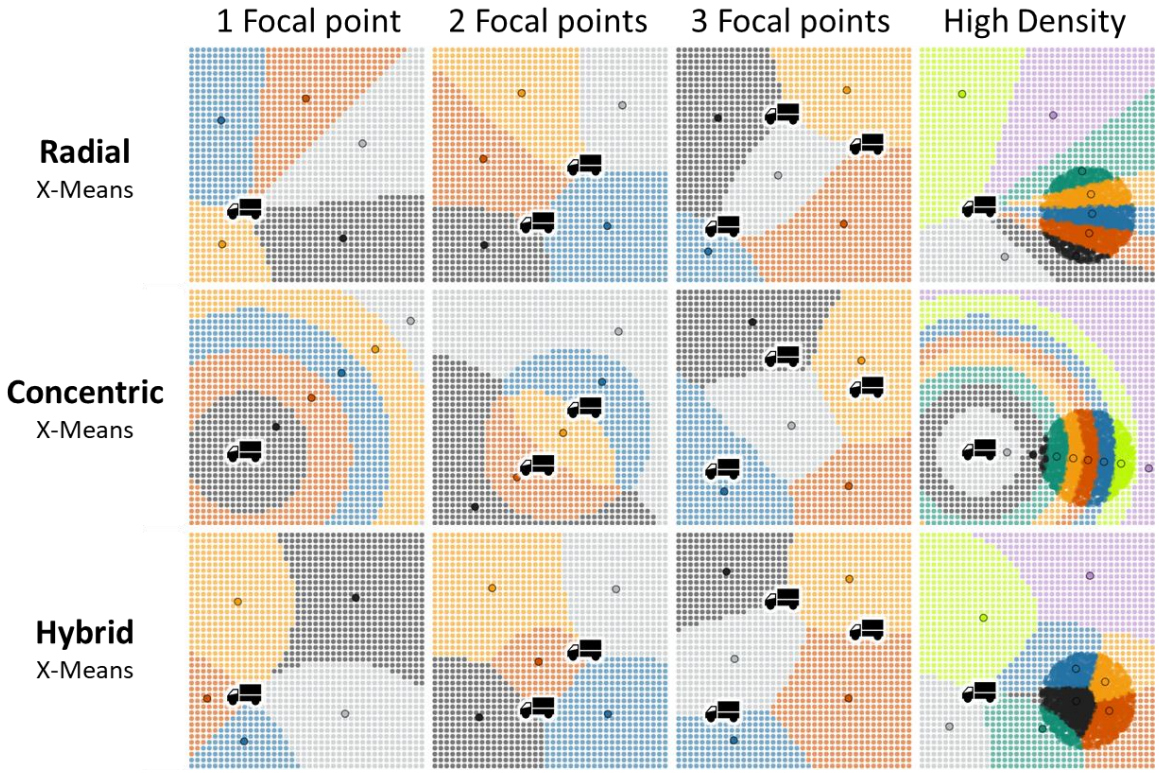


Figure 10. Clustering 1600 uniformly distributed targets with respect to one or more focal points. X-Means algorithm was used with a maximum cluster size set to *S* = 500. The column on the right shows the effect when increasing target density in one region.

4.3 Clustering using estimated travel-times

To our knowledge, there are two techniques to use the estimated travel-times in these clustering algorithms. The first is to modify the algorithms as shown in [35]. This modification is easiest for SLINK, were we simply use the estimated travel-times as the distance function when merging, with the note that the values are usually not symmetric $\widehat{time}(A,B) \neq \widehat{time}(B,A)$ and we use the average value taken in both ways.
For the K-Means variants the modification is only straightforward in the partitioning step where the distances can be substituted with the average estimated travel-time as explained above (see Figure 11). In this way the road-network is considered. The centroid update step, however, causes complications because this geometric mean can appear in unreachable locations (like the middle of a lake). Because of this reason, K-Means is typically replaced with K-Medoids [35], where the next representatives are selected as the targets with the smallest estimated travel-time to all other targets in the cluster. This selection process, however, is too slow (O(N^2)) to apply for each cluster at every iteration and we resort instead to choose the nearest target to the geometric mean (using GCD). This guarantees reachability because we select a location from the set of valid targets and it also means that the upcoming partitioning step can be computed efficiently because travel-times can be estimated in O(1) time when locations are already mapped to the nearest nodes of the overhead graph (as discussed in Section 3).

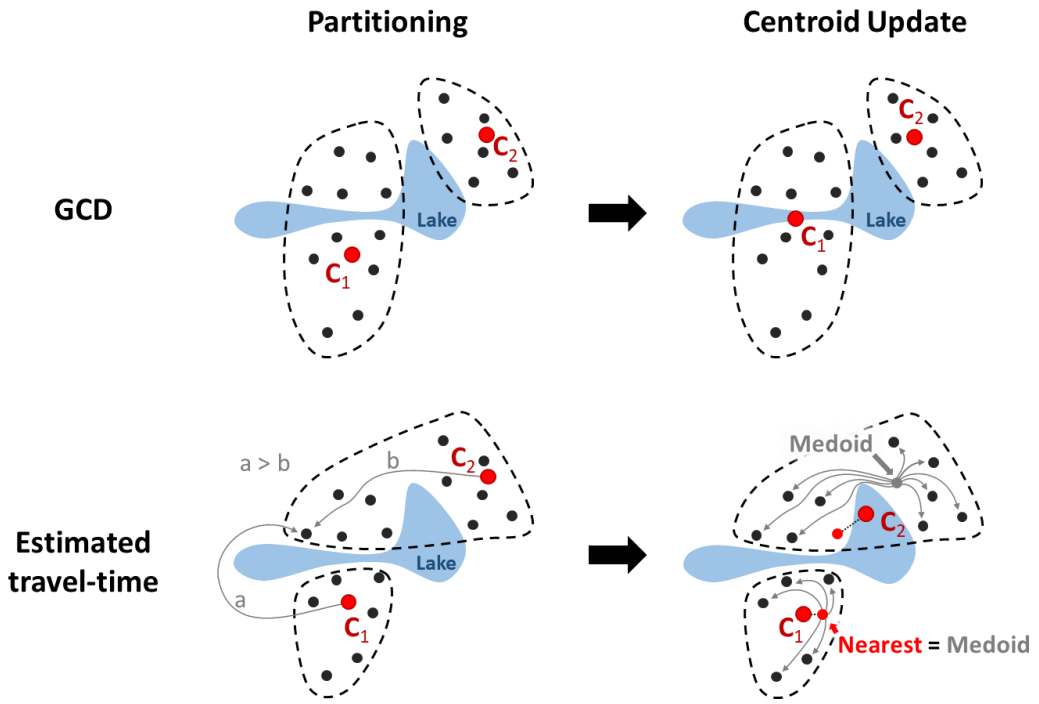


Figure 11. Effect of K-Means partitioning and centroid update steps when using GCD and estimated travel-time. The difference between selecting the medoid and the nearest target to the mean is also highlighted.

We will refer to this technique of incorporating the estimated travel-times into the clustering as MOD because the algorithms are modified to achieve the goal. Using MOD affects all K-Means variants, and all objectives except for Radial, which is geometric in nature and based on the angles which do not change when using estimated travel-times. Hybrid is partially affected because the distance component does change, but the angle component does not.

The second way to incorporate the estimated travel-times is to first project the locations into an abstract space where the Euclidean distance approximates the travel-times. Multi-dimensional scaling (MDS) [36] was used for similar purposes in the past [37]. It is a method of generating physical locations from the pairwise relationships between the targets. Using the average (both-way) estimated travel-times between the targets we can obtain a new configuration for the targets as seen in Figure 12. We notice that this new configuration takes into consideration the road-network topology and by looking at the projection alone, we can conclude that the black cluster is in fact the furthest from the agent location with respect to travel-time. It also indicates that the agent must first pass by the blue cluster to reach the black one.

The MDS projection is rarely perfect when road-network relationships are used as even simple ones may not be simply flattened on a 2D space (see Figure 13). MDS can generalize to more than 2 dimensions which sometimes can help to preserve these relationships, however, we did not notice this in practice and consider 2 dimensions to be enough. Even though some errors often exist, they are usually at the point-level and clusters are properly preserved (see Figure 12).

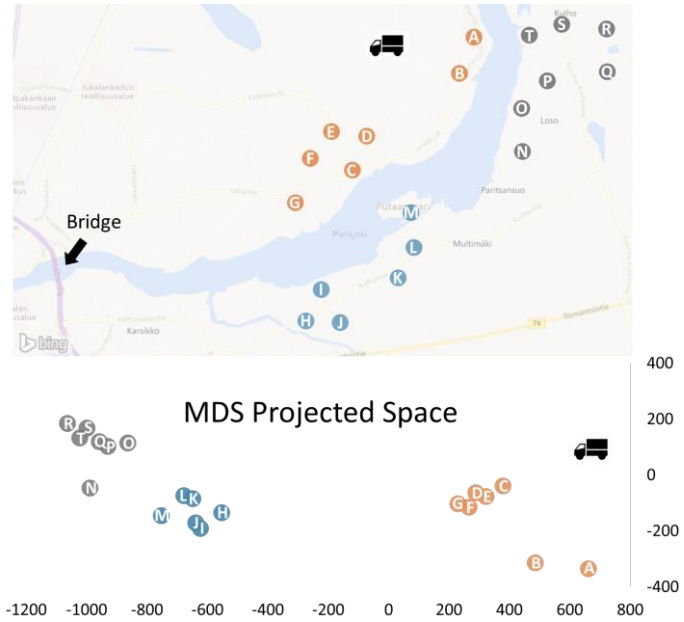


Figure 12. An example in the city of Joensuu, Finland, where targets exist on different sides of the Pielisjoki river. An MDS projection is generated using estimated travel-times. Even though targets A and T are physically close, they are far away in the MDS space because agents must take a long detour and cross the river to get to T.

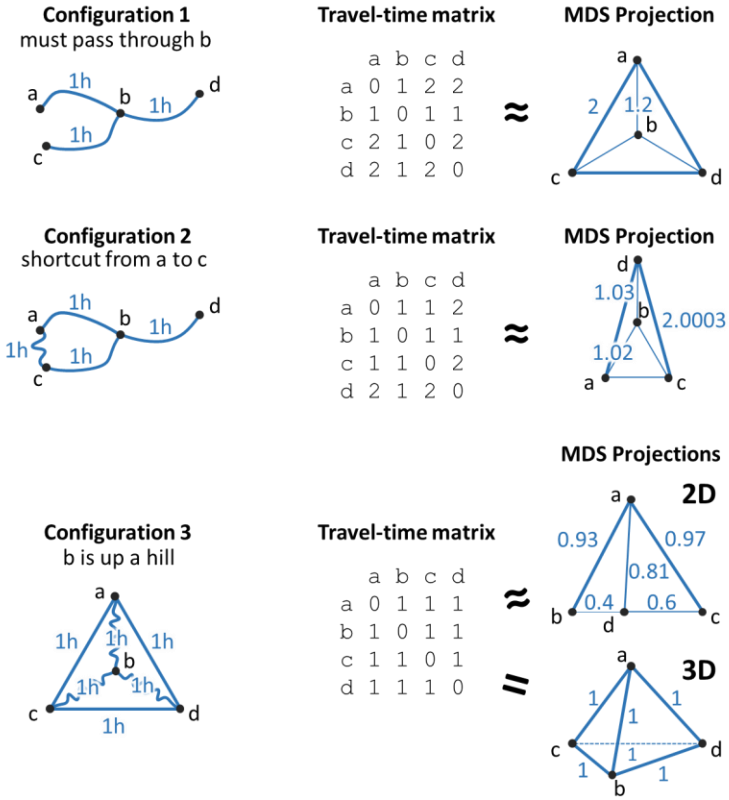


Figure 13. Examples of targets on different road-networks and respective travel-time matrices. MDS projections are computed based on these matrices. The Euclidean distances approximate the travel-times in the projected space. The travel-times are only perfectly preserved in the bottom example when projecting in a 3D space.

In the MDS projected space, the default implementation of clustering algorithms will work using the Euclidean distance. Using this technique, the Radial objective is affected as well because the angles are

different and Hybrid can now consider the angles as well. The downsides are that the projections are not perfect, and projection errors may affect the result as well. In addition to this, the classical MDS algorithm is slow ($O(N^3)$) and we need to subsample instances with more than 1k targets for it to work in a reasonable amount of time. In practice we do a random sampling and map the unchosen ones to the nearest using GCD.

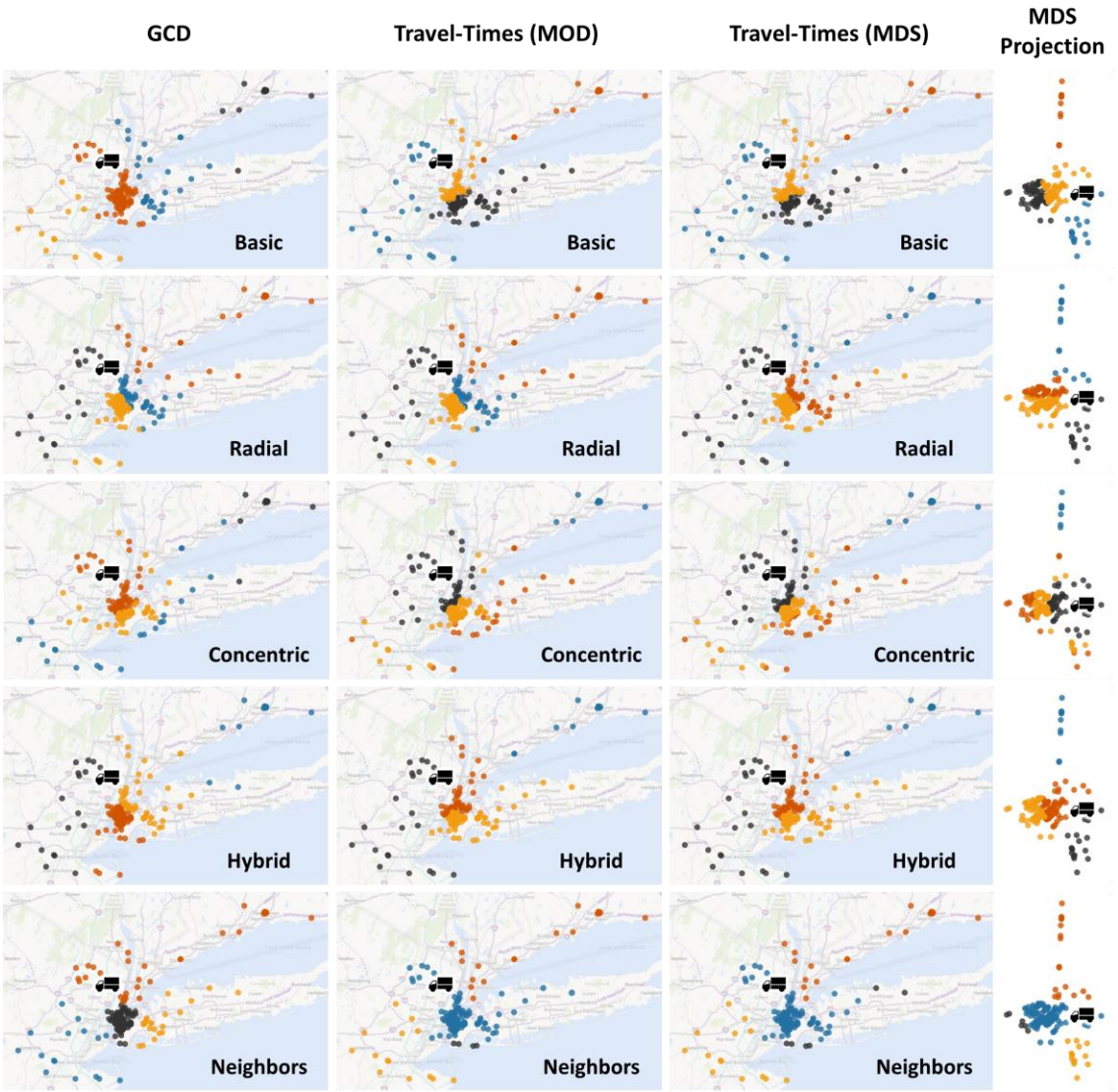


Figure 14. Clustering 200 targets with all objectives with GCD and estimated travel-times. The clusters are emphasized by different colors and the MDS projection is also shown preserving the colors of the clustering.

We demonstrate the differences when clustering using GCD and estimated travel-times in Figure 14. MOD and MDS techniques produce very similar outcomes. The most significant difference is that MOD using Radial objective is identical to GCD. We observe that using the Basic objective, the travel-time clusters appear less spherical on the map, and accommodate the geography well by not spreading over the Long Island Sound estuary. The clusters are, however, spherical in the projected space which is expected when minimizing the sum of squared errors. The Radial variant when using MDS seems to take the geography into consideration and produces 'rays' that bend slightly, especially the black colored cluster which surrounds the yellow one. We can also notice an artifact of the projection error with the two east-most yellow targets which should probably belong to the red cluster instead. The Concentric variant is least affected by travel-times. Slight variation is noticeable in the red and yellow clusters, their extremity is not convex due to the effect of the road infrastructure. This variant is the only one that produces clusters that spread over the estuary when

using travel-times, making it not ideal in this case. The Hybrid variant behaves similar to Radial near the focal point, with clusters emanating from it, however, the yellow and blue clusters appear disconnected and at the outskirts. Neighbors produces seemingly meaningful clusters already without the use of travel-times because the size of the estuary is large enough to separate the points so that their nearest neighbors are always on the same side of the estuary. When travel-times are used, the blue cluster is more meaningful when considering the road infrastructure carefully. We also notice the black cluster is relatively small which is typical of SLINK and the same artefact created by the MDS projection is here as well (the two east-most targets should belong to the blue cluster).

5 Agent assignation

A naive approach is to assign agents to clusters so that their number is proportional to the size of the cluster. This is not ideal as properties such as travel-times, dwell times, quantities and capacities play a significant role. We propose to estimate the *demand* of the clusters with respect to time, meaning how much time is required to visit all targets in that cluster. To do the estimation we use the following components:

$$\text{demand}_{\text{dwell}}(c_k) = \sum_{i=1,n} t_i^{\text{dwell}} \tag{9}$$

$$\text{demand}_{\text{travel\_inside\_cluster}}(c_k) = \text{MST}\left(t_{1,n}^{\text{location}}\right) \tag{10}$$

$$\text{trips\_to\_cluster}(c_k) = \left\lceil \frac{demand_{\text{dwell}}(c_k) + demand_{\text{travel\_inside\_cluster}}(c_k)}{\frac{1}{M}\sum_{j=1,M}\left(a_j^{eTime} - a_j^{sTime}\right)} \right\rceil \tag{11}$$

$$\text{demand}_{\text{travel\_to\_cluster}}(c_k) = \text{trips\_to\_cluster}(c_k) \cdot \min_{j=1,M}\left(\widehat{time}\left(c_k, a_j{}^{\text{sLoc}}\right) + \widehat{time}\left(c_k, a_j{}^{\text{eLoc}}\right)\right) \tag{12}$$

$$\text{trips\_to\_depot}(c_k) = \left\lceil \frac{\sum_{i=1,n} t_i^{\text{quantity}}}{\frac{1}{M}\sum_{j=1,M} a_j^{\text{capacity}}} \right\rceil \tag{13}$$

$$\text{demand}_{\text{travel\_to\_depot}}(c_k) = \text{trips\_to\_depot}(c_k) \cdot \min_{j=1,P}\left(\widehat{time}\left(c_k, d_j\right) + \widehat{time}\left(d_j, c_k\right)\right) \tag{14}$$

where $t_{1,n}$ are the targets of a given cluster c_k from the set $C = \{ c_k, k = 1 \ldots K \}$ and MST is the weight of the Minimum Spanning Tree [17] computed on the complete graph consisting of all pairwise travel-time estimates between the targets. Even though the MST is likely to be an underestimate of the travel-time between the items, we are interested in relative differences here. Multiple trips to the cluster are required if the agents do not complete the work in a single shift. Depending on how far the cluster is, these can be significant. If the capacities are not large enough, multiple trips to a depot are also required, and these can be significant too, especially if the depot is far away. In all these extra trips we consider the nearest agent and the nearest depot. To obtain the final demand estimate of the cluster and get the final number of agents to be assigned to that cluster we do as follows:

$$\text{demand}(c_k) = \text{demand}_{\text{dwell}}(c_k) + \text{demand}_{\text{travel\_to\_cluster}}(c_k) + \text{demand}_{\text{travel\_to\_depot}}(c_k) \tag{15}$$

$$\text{agents}(c_k) = \left\lfloor M \cdot \frac{demand(c_k)}{\sum_{i=1,K} demand(c_i)} \right\rfloor \tag{16}$$

It is important to note that if some properties are not given in the problem definition, they default to zero. For example, if a depot is missing $\text{demand}_{\text{travel\_to\_depot}}(c_k)$ becomes 0 for all clusters. It is possible for the final value calculated using Equation 16 to be 0, indicating that the cluster is too small even for a single agent. In this case we merge the cluster to its nearest neighbor and repeat this process until every cluster demands at least one agent. To better understand how all these components interact with each other we give several examples in Figure 15.

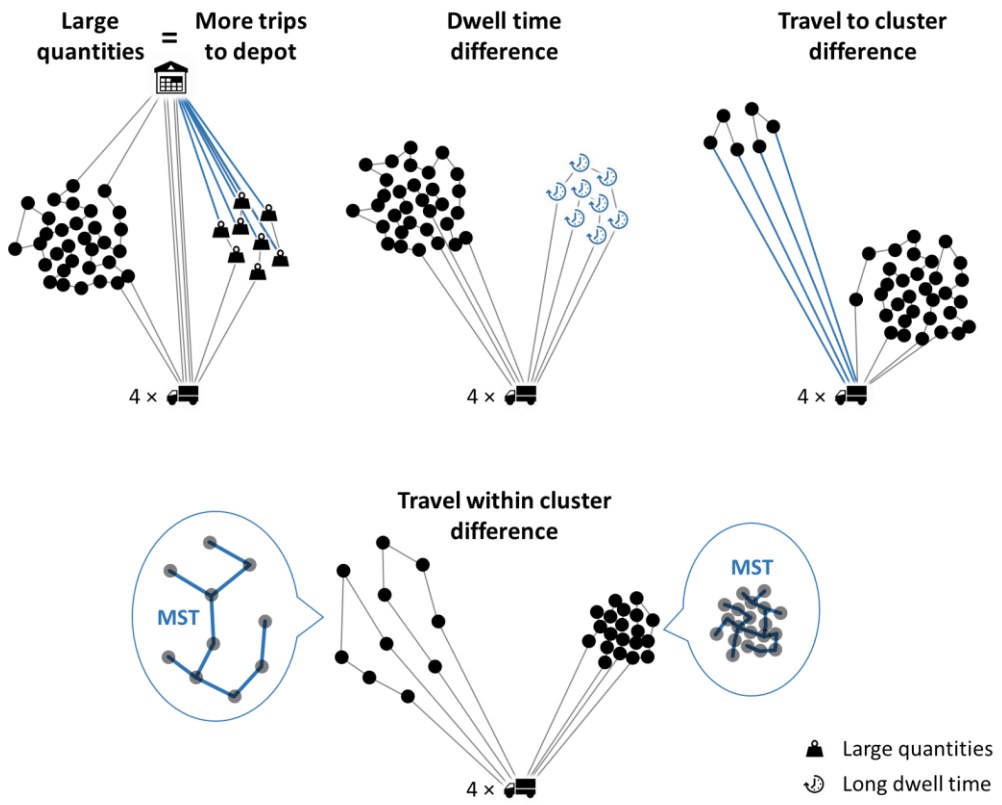


Figure 15. Four examples with two clusters of equal demand caused by different reasons.

After we know the number of agents demanded by each cluster, we next take into consideration the travel demands from every focal point to every cluster and perform optimal pairing using the Hungarian algorithm [30] between the focal points and the cluster centers. In practice, we assign a number of slots at the cluster centers equal to the value specified by Equation 15. The example in Figure 16 shows how both the cluster demands and the agent locations are taken into account. Now the agents are assigned in the order they are listed in the input. In other words, which one of the three bottom agents is delegated to the further away cluster is pretty much random. This does not matter if all agents have identical characteristics (working schedule and capacity), otherwise, we need to consider these aspects as well and we do so in the next step.

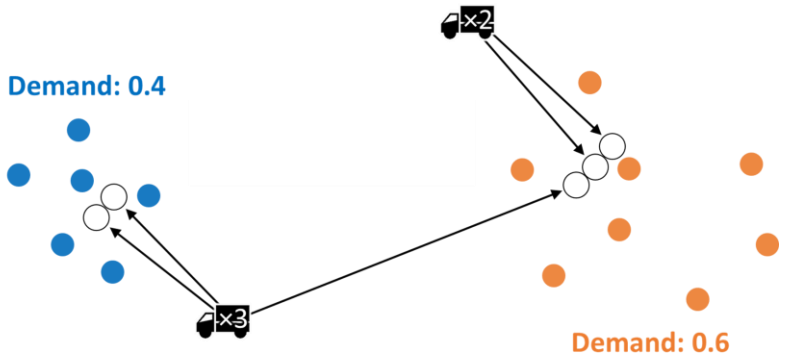


Figure 16: An example with 5 agents starting from two focal points and two clusters with different demands. The blue cluster has two slots and the red cluster has three due to the higher demand. The agents are assigned to these slots so that their travel-times to the clusters is minimized.

6 Compatibility maximization

At this stage, the clusters are VRP instances and contain both targets and agents with a minimum of one agent per cluster and the number of targets is less than *S* (if possible). Depending on the characteristics of the agents, it is possible that some of these instances are invalid based on the compatibility definitions from Equations 3, 4, 5 and 6 and some fine-tuning is needed. We now identify and resolve such issues in two steps. First, we compute the compatibility graph for each cluster and calculate a compatibility score based on the total number of disconnected components in all clusters. We then maximize compatibility using an iterative hill-climbing process. We randomly select a pair of agents from different clusters, switch them and check for improvements in compatibility score. If it improves, we keep the new assignment and continue to iterate. We first switch agents that start in the same focal point. When doing this, the optimal pairing performed earlier is not affected, just the agents become more properly assigned. It only takes a few iterations for this process to stabilize (100 iterations are enough). Then, we continue for another 100 iterations where we allow agents to switch between the focal points as well. Successful switches now affect the optimal pairing computed previously, however, such a switch makes sense if, for example, the only vehicle capable of moving some items from the blue cluster in Figure 16 is at the top. This step of switching the agents will keep the number of agents assigned to the clusters the same, but which clusters they are assigned to may change (see Step 1 in Figure 17). Then, in Step 2, we continue to maximize compatibility by moving targets to different clusters. We systematically try to allocate all incompatible targets to different clusters, beginning with the nearest ones in physical space. We know that these targets must fit somewhere because the complete instance passed the validation in Section 4.1. During this step, the clusters may change slightly (see Figure 17). It is possible that the size of the clusters exceeds *S* after this step. This happens on rare occasions, and when it does, *S* is exceeded by a small amount, in practice. Significantly exceeding *S* could potentially be problematic, however, we did not experience such as of yet. We believe that if S is exceeded by a larger amount it is a sign that the problem instance is problematic: only one or few agents can satisfy a large subset of targets. We included this discussion in the manuscript.

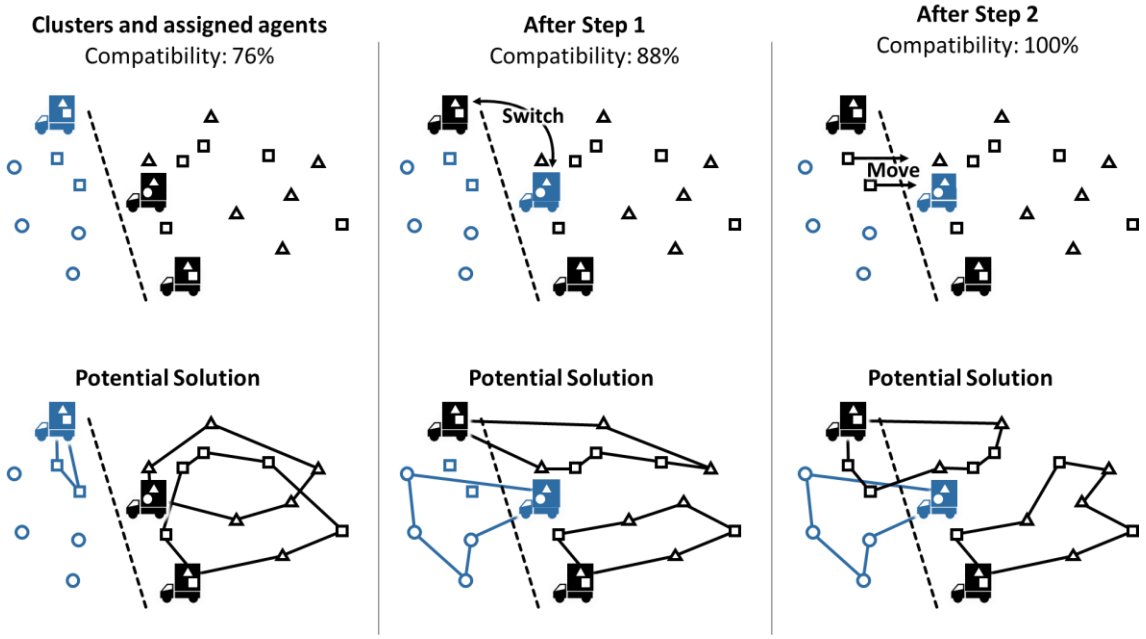


Figure 17: An instance divided in two clusters. The compatible targets are indicated for each agent using the triangle, square and circle symbols. Without compatibility maximization, four targets remain unallocated. In step 1 is number reduces to two as a result of switching agents. Then, two targets are moved to the other cluster.

This concludes the framework description; the merging step simply combines the itineraries from every cluster into a final result.

7 Experiments

In this section, we systematically test all different components of the proposed framework. Due to privacy limitations, we cannot experiment on real instances containing customer data, this leaves us only with synthetically generated data. There are multiple publicly available datasets for VRP problems but all of them seem to have coordinates projected into Euclidean space, mainly to simplify research work and focus on the optimization task while not worrying about real word travel estimations. Our framework, however, integrates into a system that considers real world travel and uses the Distance Matrix API[10] to obtain accurate travel-time and distance estimates. Therefore, to properly analyze the functionality of our proposed framework we had to generate a new dataset[11] and we used the method in [38]. The method uses characteristics learned from real problem instances to generate new ones with similar properties. The generated dataset contains instances in two distinct regions: California (Los Angeles and San Diego) and New York (and parts of New Jersey and Connecticut). We obtained target geo-locations by querying OpenStreetMap[12] for restaurants, cafes, fast-food places, ice-cream shops, bars and pubs. Using these locations, we generated multiple instances with varying sizes (200, 400, 600, 1k, 2k, 4k, 6k, 8k, 10k). For each of these we created variants with 1 and 2 focal points (where agents leave from and return to) which were chosen at random. For each of these we varied the number of agents to obtain low and high workload variants with 10 targets/agent and 40 targets/agent, respectively. For each of these we created an easier variant: 45-60 minute time windows and a harder variant: 30-45 minute time windows (see Table 1). Working times and the dwell times were automatically decided by the method in [38] to allow for realistic completion of the task during one day (completion not guaranteed) with working times for the agents varying throughout the day. This process resulted in a total of 80 VRP instances, 40 in California and 40 in New York (see Figure 14). These instances lack capacities, quantities and depots because the method in [38] does not support such properties, however, it does contain time windows, therefore, it corresponds to scenarios like providing health care services or maintenance (scenario **C** in Figure 1). We will refer to this dataset as **Realistic VRPTW** (Vehicle Routing Problem with Time Windows).

Table 1. Properties of the Realistic VRPTW dataset

	California / New York								
Size	**Small**			**Medium**	**Large**				
Targets	200	400	600	1k	2k	4k	6k	8k	10k
Workload	L & H	L & H	L & H	L & H	L & H	L & H	L & H	L & H	L & H
Agents	20 & 5	40 & 10	60 & 15	100 & 25	200 & 50	400 & 100	600 & 150	800 & 200	1000 & 250
Focal Points	1 and 2 locations								
Time Windows	Long: [45-60 minutes] Short: [30-45 minutes]								

[10] https://docs.microsoft.com/en-us/bingmaps/rest-services/routes/calculate-a-distance-matrix

[11] https://github.com/cristianalex81/vrpdiv

[12] https://www.openstreetmap.org

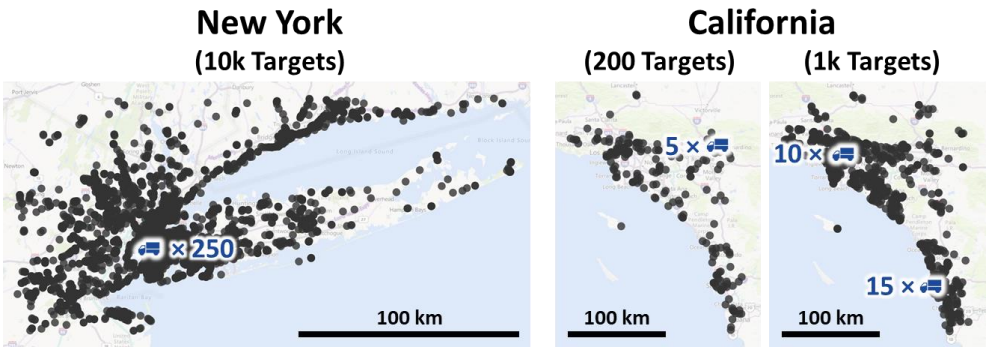


Figure 18: Three example instances from the Realistic VRPTW dataset. The one on the right has two focal points, the others have only one.

The **Realistic VRPTW** dataset we generated satisfies our desire to analyze the behavior of the travel-time estimation and most of the framework components apart from those concerning capacity limitations, and depots. To test these we use the Capacitated Vehicle Routing Problem (CVRP) instances from [39 and 40] which best resemble the waste collection scenario (scenario **A** in Figure 1). The dataset in [39] contains **Large** instances generated in 5 regions: Leuven, Antwerp, Ghent, Brussels and the whole region of Flanders with sizes shown in Table 2. The dataset contains two variants for each of these regions: in the first, the depot is in the center and vehicle capacities are smaller, and in the second, the depot is located somewhere at the edge (eccentric) and capacities are relatively large meaning that longer trips are expected (see Figure 19). The instances in the other CVRP dataset [40] are **Small**. There are 100 instances with a number of targets between 100 to 1000 targets, synthetically placed at random or clustered. These instances have a single depot positioned in various locations (center, eccentric or random). Both datasets from [39 and 40] have coordinates projected in an Euclidean space, which means that the mapping to the road-network is lost and we cannot use realistic travel-times. This is not a problem for our framework which supports trivial distance functions like the GCD or the Euclidean distance when necessary. We will refer to these datasets as **Artificial CVRP** even though the distribution of targets in the **Large** set is quite realistic (see Figure 19).

Table 2. Properties of the Large Artificial CVRP dataset

		Leuven	Antwerp	Ghent	Brussels	Flanders
Variant 1 Central Depot	Targets	3k	6k	10k	15k	20k
	Capacity	25	30	35	50	50
	Known Best	193 092	478 091	470 329	503 407	7 256 529
Variant 2 Eccentric Depot	Targets	4k	7k	11k	16k	30k
	Capacity	150	100	170	150	200
	Known Best	111 860	292 597	259 712	349 602	4 405 678

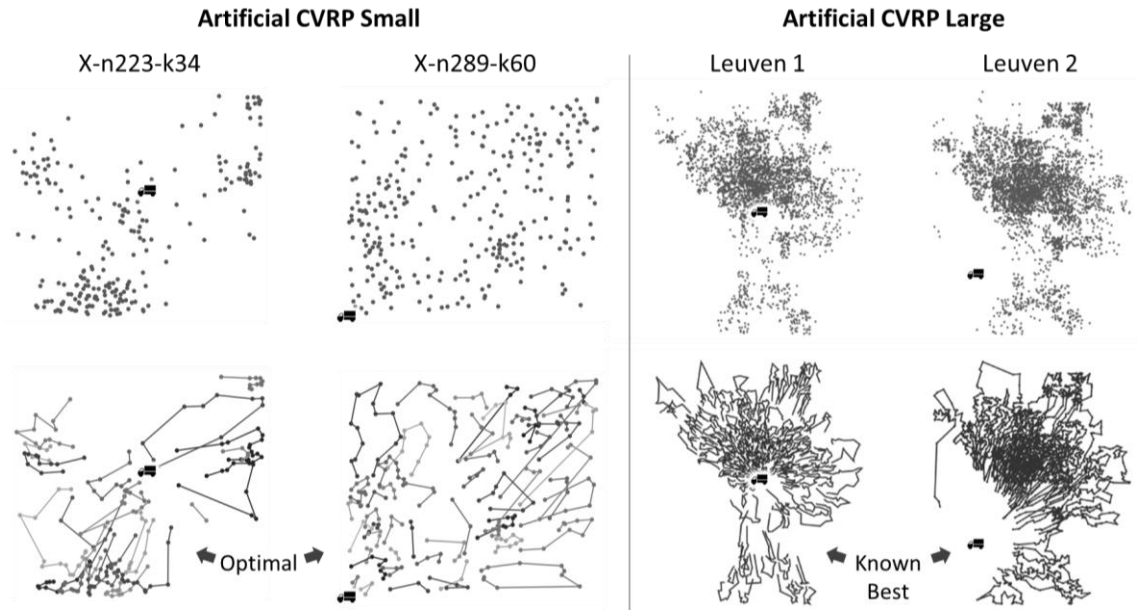


Figure 19: Example instances from the Artificial CVRP datasets. The focal point and the depots are the same. The paths to these are now shown.

The known-best itineraries are available in the **Artificial CVRP** datasets. These were optimized over the years by many researchers and are considered to be close to optimum. For many of the **Small** instances, the optimal itineraries are sometimes even known. These are very useful as they allow us to measure the quality of our generated itineraries by more than just visual inspection. Because the set contains no time window or maximum travel restrictions, it means that the number of agents is not important and even a single agent is enough to visit all targets if multiple visits to the depot are allowed.

In all our tests we integrated the proposed framework with the Multi-Itinerary Optimization[13] (MIO) *Solver* [14]. MIO uses a heuristic *solver* based on ALNS that can efficiently optimize itineraries for hundreds of targets and multiple agents. MIO can handle all constraints that we consider like: specified time-windows and dwell times, quantities, limited capacities as well as some other ones like priorities and pickup and delivery scenarios. MIO uses travel-times to generate itineraries that can be completed in practice by following the navigation instructions provided by Bing Maps with the option to use predictive traffic as well. If the locations are projected in a Euclidean space, MIO has the option to use it as a distance function.

7.1 Choosing the cluster size

One important question is: how big can our clusters be? To answer this, we first need to understand how MIO performs, we run an experiment on the **Small Artificial CVRP** dataset [29]. We test MIO on every instance as a whole, and by dividing into 2, 3 and 4 clusters, and summarize the results in Figure 20. We calculate the difference to the known-best (delta) in all these cases. When no division is applied, the delta increases with the number of targets, as expected; however, the increase is not necessarily proportional to the number of targets. Other factors do affect the solution quality like the target locations, quantity amounts, capacity limits, and depot location which make some instances more difficult to solve than others. When dividing, we see a negative effect on the smaller instances (< 300 targets). This is because on this size, MIO finds solutions, which are near optimal, and dividing prevents it from reaching that (see Figure 6). Here, the quality of the solution worsens as we increase the number of clusters because a higher number means a higher probability to interfere with the optimal itineraries. Then, dividing starts to be of help with significant improvements

[13] https://www.microsoft.com/en-us/maps/multi-itinerary-optimization

when there are over 600 targets. The negative effect of the division at these larger sizes are significantly outweighed by the suboptimal performance of MIO.

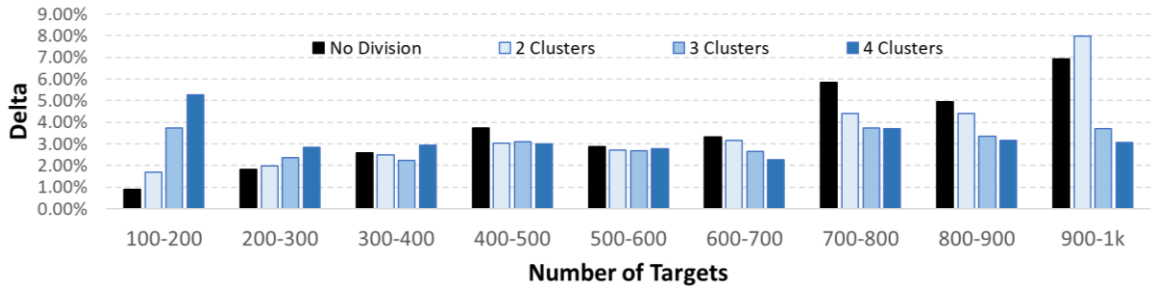


Figure 20: Summary of executing MIO with and without division on the Small CVRP dataset aggregated by number of targets. These are single runs, however, an averaging effect happens because there are multiple instances per size group.

As a consequence of this evaluation, in all following experiments we set a maximum number of targets per cluster *S* equal to 300. Where MIO still performs well without division. Here, MIO is expected to perform with a 2.6% delta and a run time of 80 seconds, a value we consider to be a good balance between quality and speed in practical scenarios.

7.2 Assessing quality of travel estimates

Large instances, where the number of targets is very high, pose an engineering problem from both a time and memory perspective when calculating the real world pairwise travel-cost for all locations, therefore, we estimate it using an Overhead Graph [19]. The question we address here is: how good is this estimation?

To evaluate the quality of this estimation we perform the following experiment on the **Small Realistic VRPTW** dataset (instances with 200, 400 and 600 targets). These instances are small enough to call the Distance Matrix API on all locations so that we can measure the quality of the estimation relative to them. We generate multiple overhead graphs of varying sizes beginning with 10 nodes and incrementing by 10 until the graph contains all locations and compare the absolute difference between the true travel-time and the estimate for all pairs. We summarize the average error and the correlation between these values in Figure 21. We notice that the average error in travel-time is somewhere around 450 seconds when the graph is small and reaches zero when the graph contains all targets. The decrease in error is highly correlated with the number of nodes of the overhead graph, but a slight noise is noticeable because nodes are chosen differently for each graph. A noticeable knee-point appears when the overhead graph has roughly 50 nodes, here, the graph models fairly well the region already and mostly fine-tuning happens after that. The knee-points occur at roughly the same place because the size of these regions is similar (see Figure 18). The decreasing average errors are not enough to indicate that clustering will improve. For example, it is possible to lower the average error by simply using GCD and scaling it with a fixed value and in [41] the authors demonstrate the value of 1.4 to be an optimum in the United States. Doing so, however, has no impact on the clustering, therefore, we also plot the Pearsons' correlations here as well. We notice that adding more nodes in the overhead graph makes the estimates closely reflect the true travel-times. These relative differences are what matter in clustering and we can spot a similar knee-point here as well.

We now fix the size of the overhead graph to 200, which is well beyond the knee-point. Here, errors are expected to be below 200 seconds which is not bad considering the average travel-times in these regions are above 30 minutes. These times are longer in California because there are two high-density regions (Los Angeles and California) which generate longer pairwise distances compared to a single high-density region in New York. The errors do not grow in proportion with the time between the targets because overhead graph node locations represent the target locations fairly well as a result of clustering.

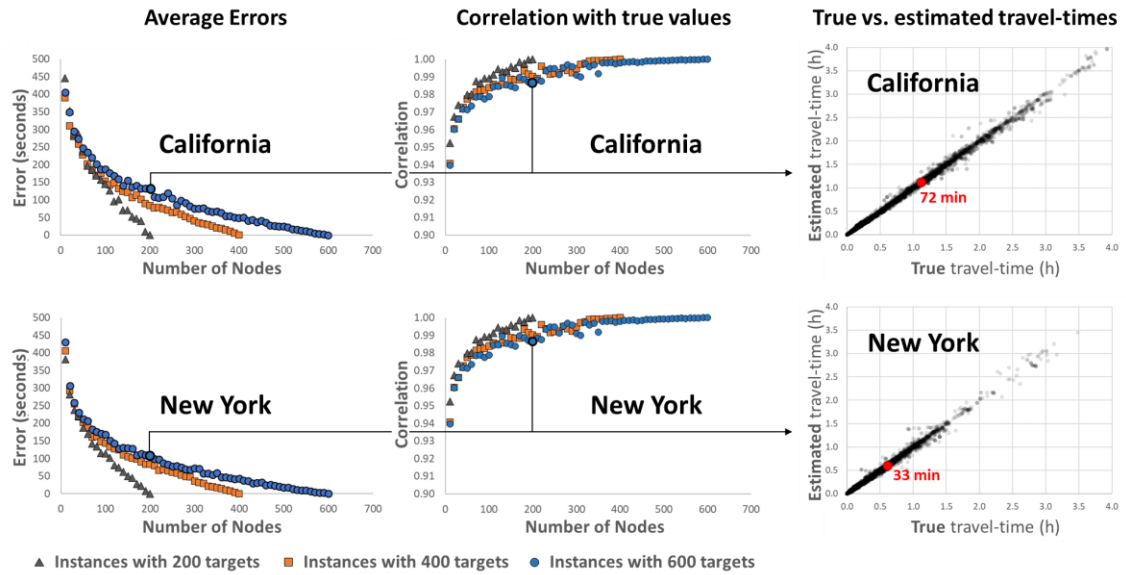


Figure 21: Travel-time estimation errors and correlations to true values on the Small Realistic VRPDIV dataset.

7.3 Clustering using travel-time

The most common distance function used in clustering is Euclidean or the great circle distance (GCD) when locations are on the surface of a globe. It stands to reason that using the travel-time when clustering should produce a better division that takes into consideration the road-network characteristics. In this subsection we investigate if this is true.

We evaluate the two methods that incorporate travel-times, MOD and MDS, into K-Means clustering variants. To do this, we again use the **Small Realistic VRPDIV** dataset because it is small enough to systematically test multiple settings. We cluster each instance thrice: once with the first method (MOD – modifying the clustering algorithms), once with the second method (MDS - multidimensional scaling and clustering using Euclidean distance in the projected space) and as a baseline, we use GCD (great circle distance - no road-network information). The baseline measures distance and not time like the other two. However, it could be converted to time by specifying a fixed speed. Doing so, would not change the relative values between the target locations, therefore using it as such is also possible. To have control over the number of clusters, we use K-Means and vary the *K* between 2 and 15. We measure how the Total Squared Error (TSE) computed as the sum of squared travel-times from every target to its cluster representative improves relative to that from the baseline (see Figure 22) and notice that the TSE improves differently depending on the region, with a more significant improvement in New York. This difference is largely due to geographical differences (the Long Island Sound estuary), which creates greater differences between GCD and the travel-times (see Figure 23). We notice how the north-eastern locations belong to the central cluster when travel-times are considered instead of belonging to different clusters which spread over the estuary, which cause a high SE because of detours to the mainland. In California, an area where there are few natural barriers, the cluster partitions do not change much when travel-times are used, the most significant difference being in the north-west due to the road infrastructure, which makes travel-times from east to west shorter than from north to south.

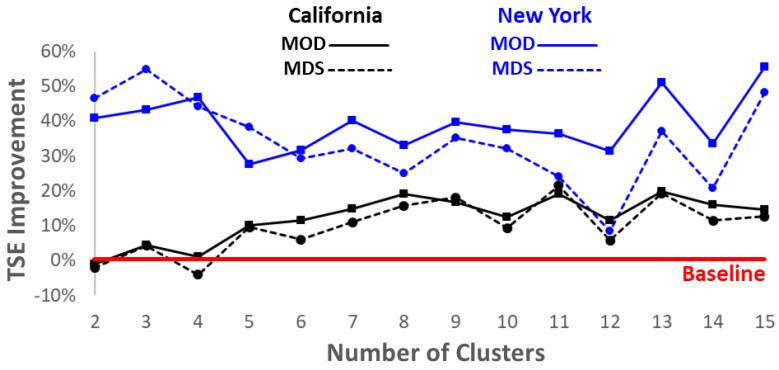


Figure 22: Improvement in quality of clustering (TSE) when travel-times are used for MOD and MDS.

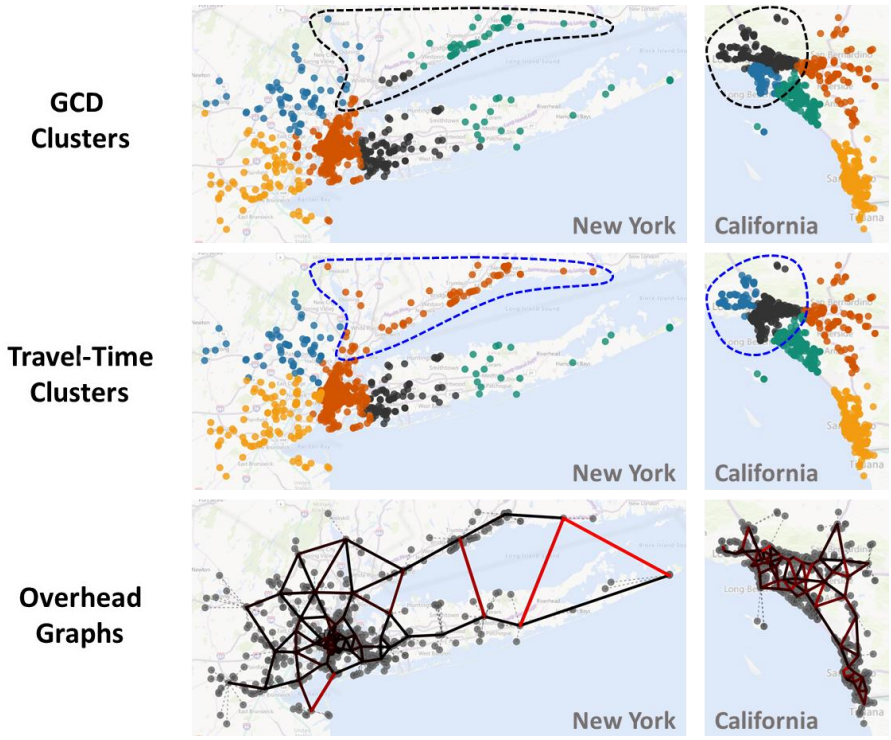


Figure 23: Differences in clustering with GCD and with travel-times. Significant differences are highlighted. Overhead graphs with 64 nodes are shown for reference (only nearest neighbor links to avoid congestion).

Both MOD and MDS variants are viable options to create more meaningful clusters especially in complex topographies that contain natural barriers. In practice, both variants produce similar levels of improvement with the observation that MDS tends to worsen slightly as the number of clusters increases. This effect, however, is not due to the number of clusters, but due to their sizes becoming too small in our experiment. On small clusters the inaccuracies in the MDS projection become more apparent which is not as common for larger clusters (300 targets). Small clusters (that demand less than one agent) are merged to adjacent ones as described in Section 5, therefore, they are even less likely to happen.

Both MOD and MDS produce meaningful clusters which impact the quality of the solution when MIO is used to generate itineraries (see Figure 24). The clusters formed using GCD have higher demand, especially those that spread over the Long Island Sound estuary. This impacts allocation, but a higher impact is on the agents travel times where we note a 9.8 % improvement when travel-times are used. On average, MOD clustering improves allocation by 5% and MDS by 5.2%. MDS brings additional benefits as well, such as

affecting the behavior of all objective functions like Radial and Hybrid and makes it easy to integrate new algorithms in the future, without changes to their implementation. As a conclusion, we recommend the use of MDS and fix this variant in upcoming experiments.

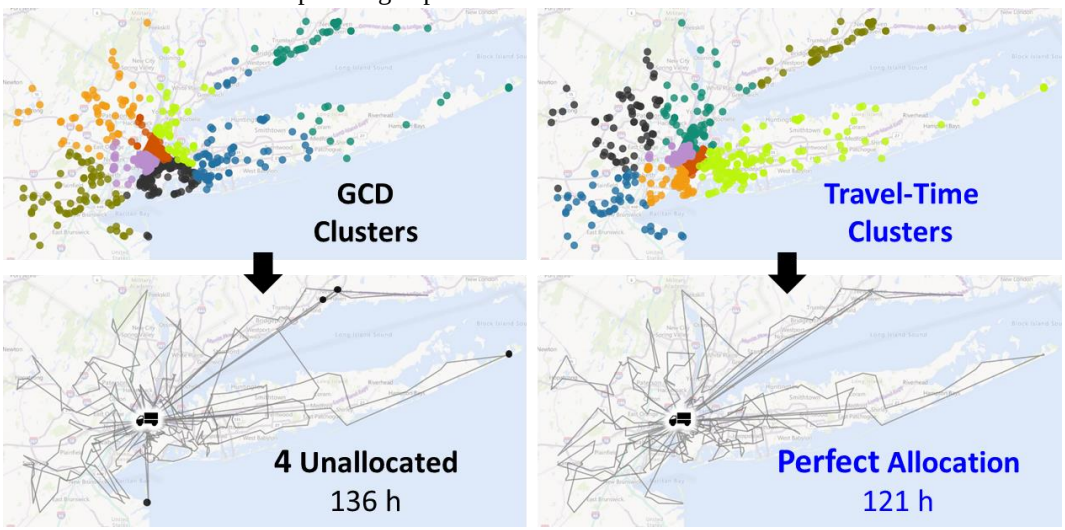


Figure 24: How allocation and travel-times improve when clustering is done using travel-times.

7.4 Agent assignation

In this subsection we investigate how different assignation techniques affect target allocation. We use the **Medium Realistic VRPTW** dataset (1000 targets per instance) and consider three different methods:
1. a baseline where naive agent assignation is done (number of agents proportional to the number of targets)
2. our proposed demand-based assignation and
3. our compatibility maximization postprocessing step (on top of the demand-based assignation).

We divide into clusters with a maximum size $S = 300$ targets using every clustering objective and solve the clusters using MIO. We summarize the results in Table 3.

Table 3. Agent assignation effect on allocation

Component	Improvement in allocation	Unallocation per Cluster (Standard Deviation)	Unallocated
Naïve assignation (baseline)	-	4	2.0%
Demand-based assignation	25%	3	1.5%
+ Compatibility maximization	15%	2	1.3%

We see that the unallocated targets reduce significantly (25%) when agent assignation is done based on demand. We also observe a lower standard deviation of unallocated targets per cluster than in the baseline method. This indicates that agents are assigned unevenly in the baseline, causing some clusters to have too few and other clusters to have more than necessary. This can also be seen in the example from Figure 25, where the biggest effect happens on the green cluster on Long Island. The demand of that cluster is high due to the inside cluster travel estimated using the MST. As a result, the number of agents to be assigned is 6 instead of 3. Even though the red and purple clusters have many targets, they are nearby and close to the focal point which lowers their demand. This increase causes all targets in that cluster to be allocated.

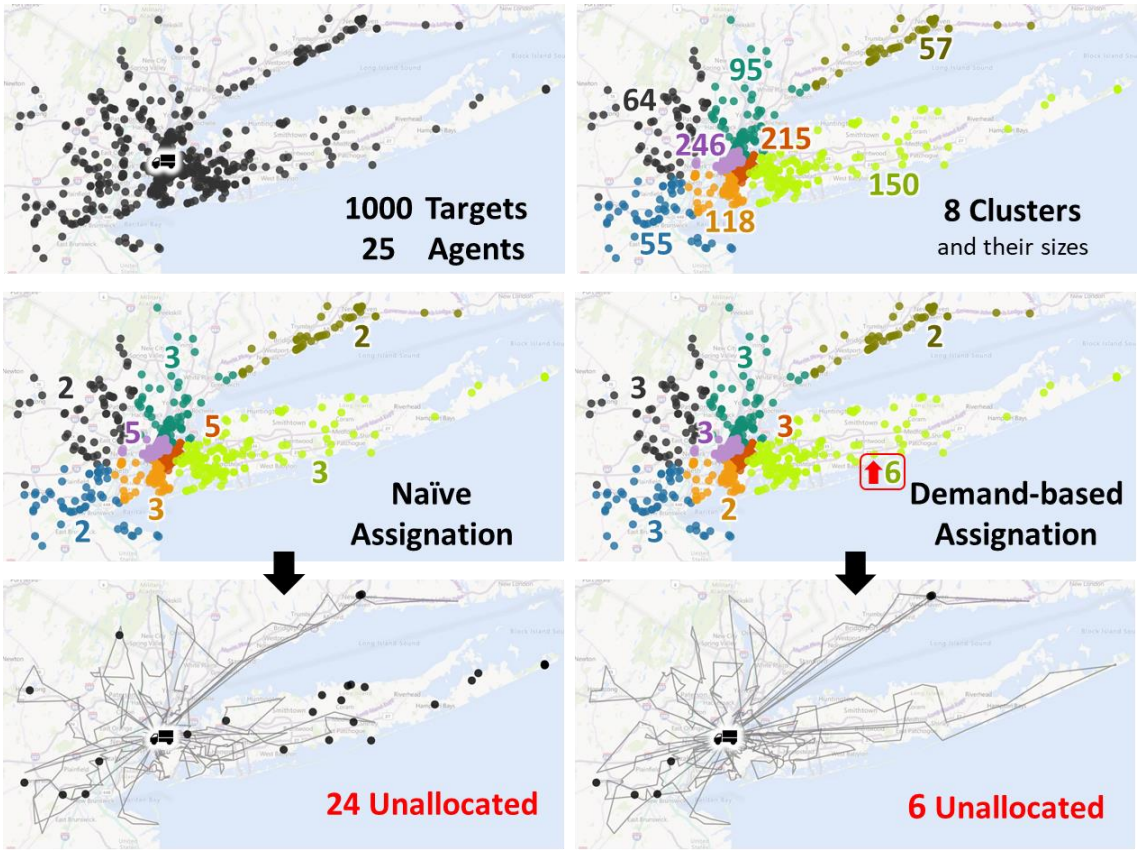


Figure 25: A VRP instance divided into 8 clusters. The number of agents assigned to the clusters differ for the naïve and demand-based variants (the most significant difference is highlighted). The bottom row shows the MIO itineraries and unallocated targets in black

When adding compatibility maximization, the number of allocated targets improves again by 15%. To explain the impact of this step, we first note that the 6 unallocated targets from Figure 25 were not caused by an insufficient number of agents assigned to those clusters. Instead, they are unallocated because they are incompatible with the assigned agents (target time-windows are outside the working hours). In the compatibility maximization step, some agents are swapped between the clusters and, consequently, all targets were allocated (see Figure 26).

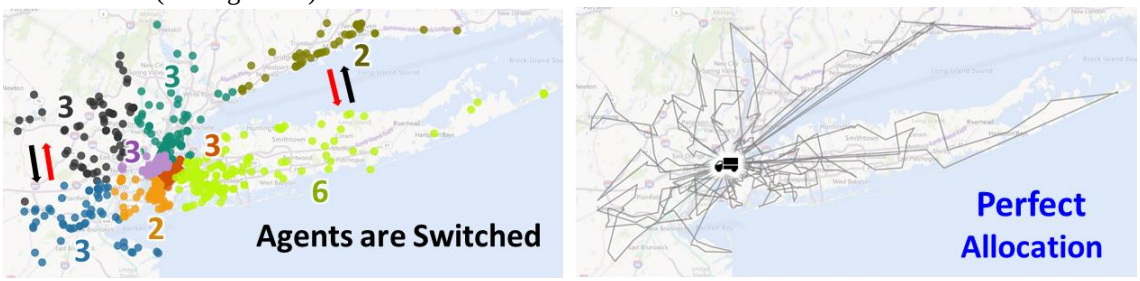


Figure 26: Assigned agents are switched between the indicated clusters. All targets are allocated after that.

We obtain a 98.8% allocation on this dataset, which means that on average 12 targets remain unallocated per instance, with a standard deviation of 18. Noticeable differences in allocation depend on properties of the instance (see Figure 27). More unallocated targets are likely to exist in the California region because there are two places with high target density (Los Angeles and San Diego) which are harder to reach, especially from a single focal point. Two focal points can be beneficial if they appear in the two regions, however, since they are generated at random this does not usually happen. Allocation is a bigger problem on instances with

high workload (40 targets per agent) however, in the low workload cases (10 targets per agent) we usually achieve maximum allocation, with a note that some targets are labeled as unreachable by the Distance Matrix API and therefore, cannot be allocated (like those on the island in California in Figure 18). We choose to include such targets in the dataset because VRPDiv must be able to cope in these situations and it makes for proper testing. Allocation is less on short time windows which are harder to satisfy, but only in the high workload cases.

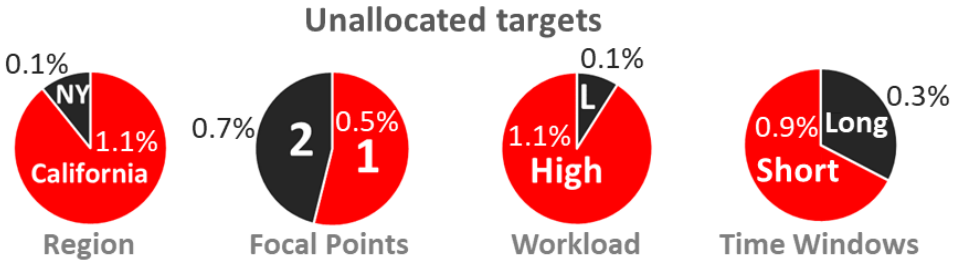


Figure 27: Allocation depends on properties of the instance.

The improvements over the baseline appear for every clustering algorithm and every objective function; however, some improve more than others (see Figure 28). Radial and Neighbors improve the least. This is because radial clusters are long and narrow, causing them to have similarly long minimum spanning trees, which means that the demand is mostly estimated by the dwell times and the distance to the cluster. The number of agents in this case is more similar to that of the baseline. Neighbors, does not improve much because the clusters already optimize distances to the nearest neighbors. This causes minimum spanning trees to have shorter links, which again, diminishes the effect of this component making the demand more similar to that of the baseline. All other objectives show a significant improvement of around 50% in allocation.

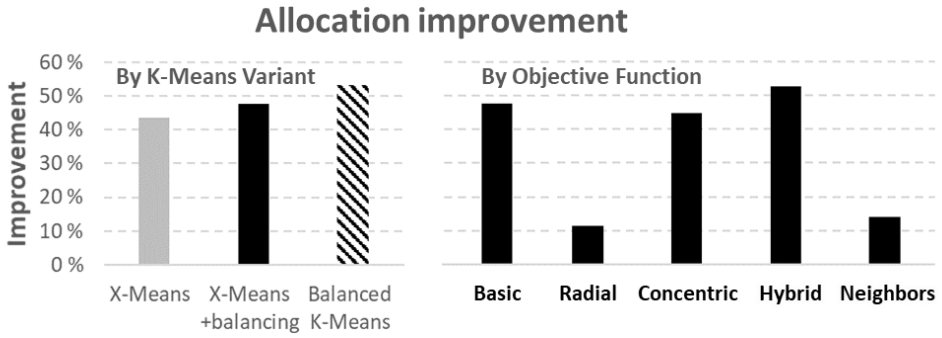


Figure 28: How allocation improves relative to the baseline for each K-Means variant and by objective function.

We conclude that assignation based on estimated-demand and using compatibility maximization are critical factors, which have a positive impact of 40% on allocation for most objective functions including the Basic one used in research.

7.4.1 Discussion on the Number of clusters

We do not know what the maximum possible allocation is for these instances, but in the upcoming analysis, we consider the known-best to be the maximum of all tested methods. We notice that Balanced K-Means finds the known-best allocation most often (68% of the time) making it the winners in a sense (see Figure 29). X-Means produces 8 clusters on average, and 6 clusters when balancing is added. Balanced K-Means always produced 4 clusters. Based on this and their relative standing in Figure 29, we can draw a conclusion that having less clusters is indeed better, and balancing is one way to achieve this. Balanced K-Means is too slow for larger instances, therefore, we set X-Means +balancing to be used in all further experiments and use it to optimize all objectives: Basic, Radial, Concentric and Hybrid.

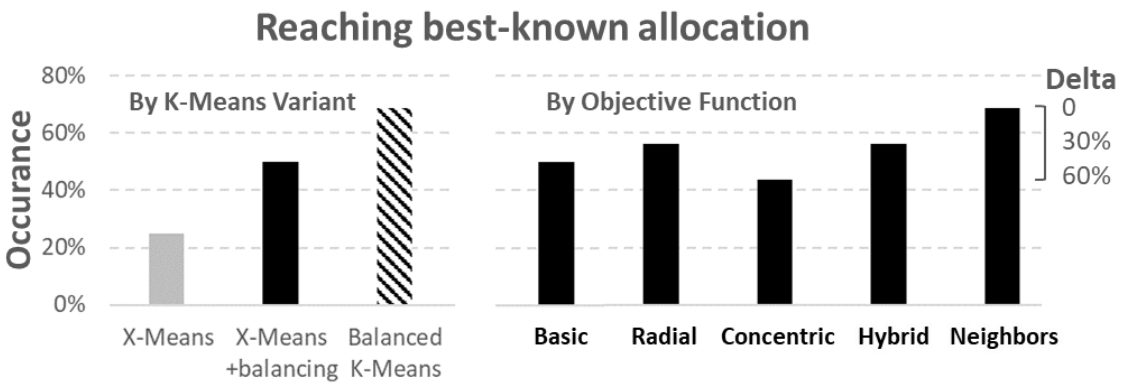


Figure 29: Probability to reach the best-known allocation by algorithm and by objective function.

7.5 Effect of different clustering objectives

We now study how the different clustering objectives affect allocation and to the total travel-times of the agents. We first use the **Large Realistic VRPTW** dataset (80 instances with 2k – 10k targets) and divide it into clusters with a maximum size of 300 using X-Means +balancing and all objectives: Basic, Radial, Concentric and Hybrid and SLINK for the Neighbors objective.

The allocation statistics are summarized in Figure 30, where we again consider the known-best allocation to be the maximum reached by the five objectives. This known-best allocation is most often reached using the Neighbors objective, followed by Radial and Hybrid, in that order. These three objectives have also a complementary effect, meaning that they are the only ones to reach the known-best allocation on several occasions. Basic and Concentric are superior only once and when they are, the other three methods are not far behind (1 – 3 targets difference in allocation). A counter-intuitive fact is that the percent of unallocated targets decreases as the instances get larger. This is, in fact, true because the number of agents is proportional to the number of targets, but the surface areas are roughly the same which means that in the larger instances, there is a higher density and targets are easier to reach.

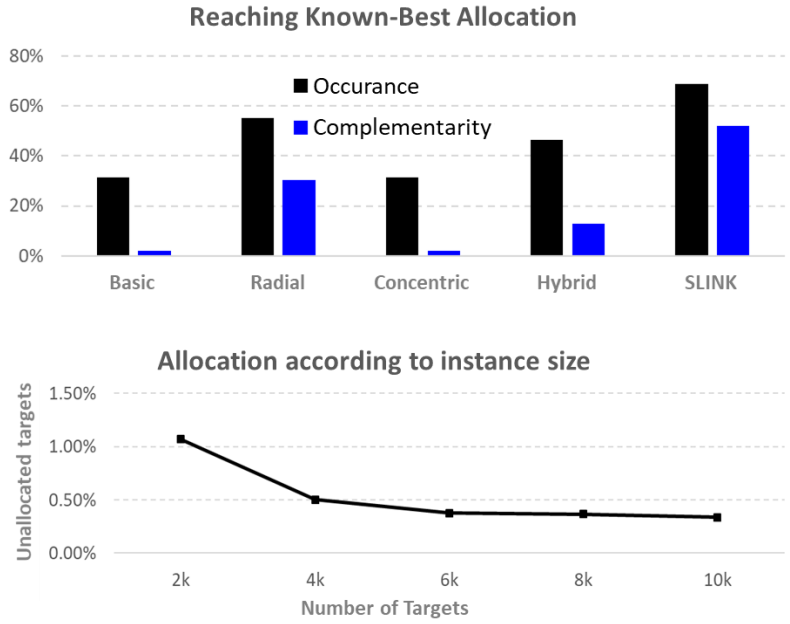


Figure 30: Analysis on allocation for the Large Realistic VRPTW dataset.

To be able to better understand the results, we explain based on a representative instance clustered using all objectives in Figure 31. On this instance the resulting numbers of clusters differs depending on the objective used. These values are representative of the average cluster counts (Basic: 33, Radial: 31, Concentric: 30, Hybrid: 31, Neighbors: 30). Every one of these clusters has less than $S = 300$ targets. The basic objective produces the most clusters and even though the difference is small compared to others (2-3 clusters) we believe this to be one main reason for the lower allocation because when dividing agents into more groups, cluster compatibility scores tend to be lower (in larger groups agents can help each other out).

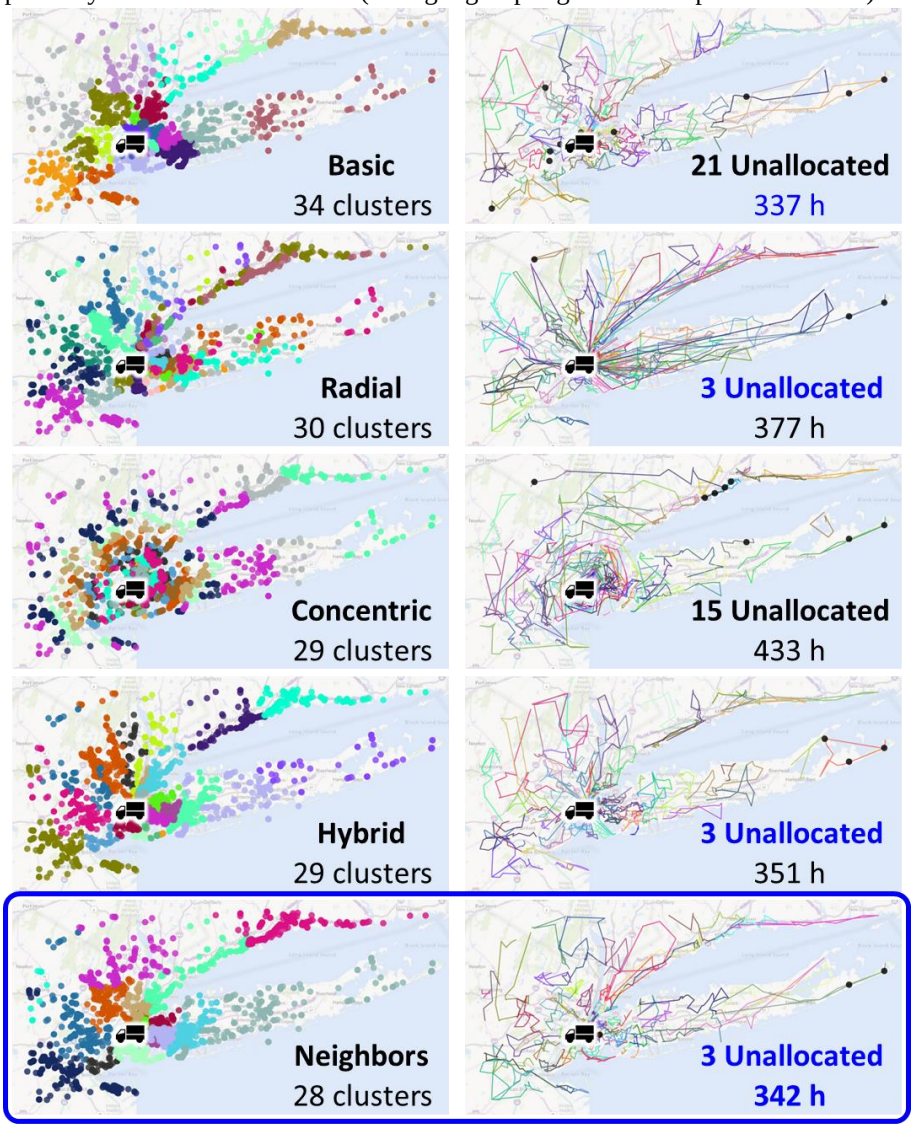


Figure 31: An example instance with 6000 targets and 150 agents divided with all clustering objectives. The respective itineraries from MIO are shown in different colors (omitting the links to the focal point). Unallocated targets are shown as black circles. The truck icon hides some unallocated targets.

For the instance in Figure 31, Basic produced the highest number of clusters (34) when compared to other methods. As a result, it is more difficult to assign agents properly and there are many unallocated targets as a result. The travel-times are lowest here, however the main reason for this is that less targets are reached compared to other methods. Radial gives one of the best allocations for this instance, however the travel-times are too long because many agents are forced to make long trips, often to the outskirts and back. The reason for such long trips is that the rays become thin on large instances and the time-windows of the targets vary along the ray so that more than one agents are needed throughout the way. We can notice the clusters

have some overlap, especially on Long Island, making it difficult to distinguish the rays at times. This is due to several reasons like the road-network topology inducing some curvature as result of the MDS projection and also due to the compatibility maximization step where some targets switch clusters. Concentric produces a the least visually appealing result because traveling on such thin concentric bands appears very restrictive especially where agents are forced to traverse the estuary by making detours to the mainland. This appears to be the main reason for a lower allocation compared to more suitable objectives. Surprisingly, the allocation is slightly better than when using the Basic despite Concentric having 28% longer travel-times. This indicates just how important it is to minimize the number of clusters as much as possible to facilitate agent assignation. Hybrid produces one of the best allocations, same as Radial, however a the travel-times are superior because the thinner rays are only near the focal point. The best result we consider to be that of the Neighbors objective. It allocates as many targets as Radial and Hybrid, however, the travel-times are the lowest. It forms clusters in higher-density regions which appear meaningful and the number of clusters is also the lowest. There are few noticeable artefacts caused by the bucketing strategy that can be seen in the western side where the light-blue cluster separating items in an unnatural way, however the impact on the itineraries is minimal.

Next, we analyze the objectives with a focus on the travel-times. To do this, we limit to those instances where all methods obtain the known-best allocation, otherwise it is unclear if the travel-times are smaller because the paths are optimized or because there are less allocated targets. There are only 17 / 80 instances where all objectives produced the known-best allocation. It is no surprise that these 17 instances are of the easier kind where allocation is not a concern: most of them are in New York (13 / 17), most of them have 2 focal points (10 / 17) and all have low workload. Since we achieved the maximum possible allocation on the low workload instances (apart from unreachable targets) we know that the unallocated targets in these 17 instances are the same for all objectives. The instances have all possible sizes but a slight tendency towards larger ones. These are all in line with observations from Figures 27 and 30.

We present the results in Figure 32. And notice that the Basic variant produces the shortest travel-times, with only a 1 hour difference from the known-best. Basic is followed closely by Neighbors and Hybrid, both with only a 2 hour difference which is not significant when considering the total working time of agents. Radial produces itineraries which are 20 longer, on average, which roughly means the salary of 3 full work days and we consider this to be significantly worse. Itineraries generated using the concentric objective are even worse (70 extra hours).

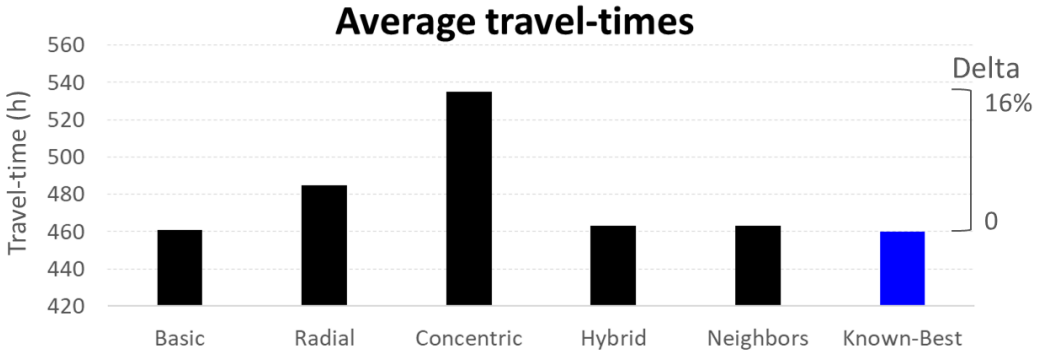


Figure 32: Total travel-times of agents averaged over all 17 instances where all objectives achieve the known-best allocation.

Based on this analysis, the Basic objective can be applied on easier instances where allocation is not a concern (low workload, multiple focal points and long time windows). However, because of its tendency to produce too many clusters (2 – 3 more clusters on average even on the subset of 17) it makes agent assignation more difficult and often results in unallocated targets. Depending on the scenario, it may be acceptable to move unallocated targets to the next day, like in parcel delivery or electrical maintenance or it may not be acceptable, like in providing health care services where the consequences of such can be fatal. Because the

Neighbors objective is very close to Basic in terms of travel-time and achieves the best allocation by far on this dataset, we recommend using it when dividing **Realistic VRPTW** type instances.

We next consider the **Large Artificial CVRP** dataset. These instances are different from the **Realistic VRPTW** in several ways. First, the travel is artificial. Even though the distribution of the targets is realistic, the locations were projected in the Euclidean space, losing the true latitude and longitude values in the process and the relationship to any road network. This dataset has been optimized by other researchers using the Euclidean distance and the known-best values are already available [39] allowing us to compare our results against other state of the art solutions. We will also use the Euclidean distance for a proper comparison. The second difference to the **Realistic VRPTW** dataset is the lack of time windows (including agent start and end times) which essentially means that allocation is no longer a concern because the agents can work indefinitely and the targets can be visited at any time. The third difference to the **Realistic VRPTW** dataset is that of capacity limitations which generate a need for multiple trips to the depot. Otherwise the problem would essentially be a TSP.

We execute our framework with $S = 300$ and compare the quality of the results by analyzing the deltas (differences to the known-best values) and present these results in Figure 33. The deltas are calculated with respect to the total travel distance computed using the Euclidean distance. We notice a different ranking when compared to that on the **Realistic VRPTW** dataset. The best solutions here are obtained by Radial (8 / 10), followed by Hybrid (2 / 10), however, the differences when Hybrid outperforms Radial are small (<1%). Basic, Concentric and Neighbors follow in quality with the last two significantly worse; the Neighbors showing a complete turnaround from the previous experiment. This is not necessarily surprising as this is a different dataset where the instances have different properties. Here there are no time-windows and there are specified quantities at each target. The lack of time-windows, in general, means less zig-zag in the optimized itineraries, whereas the added quantities mean multiple trips to the depots are necessary. Another important reason is that this dataset does not consider realistic travel. When moving as-the-crow-flies the itineraries are simpler and detours are never a concern and the itineraries travel-times do not cause additional artifacts (see Figures 19 and 34).

We next focus on Radial, the best performing objective in more detail and see how it behaves on each instance (see Figure 33). We notice relatively low deltas for the first variant of each region, where the capacity is less. Lower capacities mean that multiple short trips are needed with frequent visits to the depot. The clustering does not interfere as much with the shorter trips as with the longer ones from the second variant which is the main reason for the difference in deltas.

We are able to solve some instances with between 3k and 20k targets with the same quality as on instances between 600 and 900 (see Figure 20). For the second variant of each region, however, when the capacities (workload) are higher, the trips are significantly longer and the division interferes more causing higher deltas (15%). There is a strong (-0.93) Pearsons' inverse correlation between the wrokload of the agents, and the quality of the solutions.

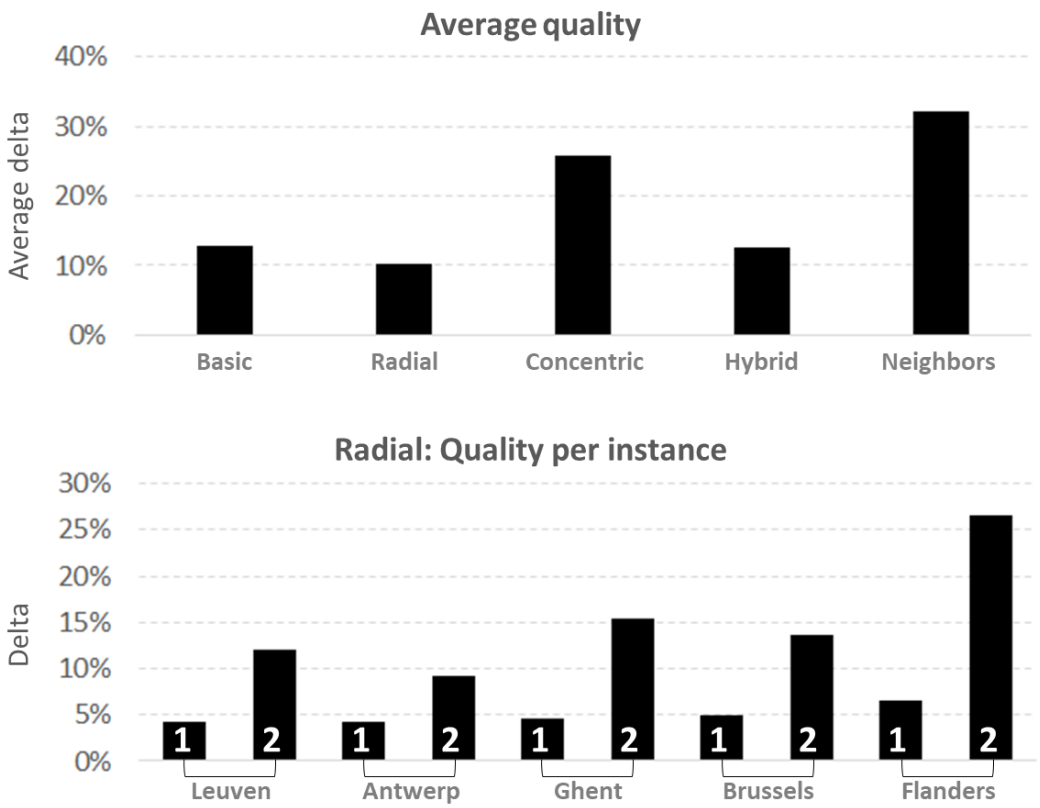


Figure 33: Average differences to known-best per clustering objective (top) and the performance of Radial X-Means per instance, which performs the best overall (bottom).

We next examine the behavior of the different objective functions by analyzing Figure 29. We notice that in the known-best solution, the agents tend to reach the extremities while picking up quantities along the way and the same happens when they return, pretty much without them deviating from a straight path. This effect is especially noticeable in the central region with higher density. At the outskirts, agents show a different pattern where they spend some time picking up targets to avoid another trip. The paths also vary more in the lower density region in the south. The Basic objective creates spherical clusters due to the lack of travel-times. Most of them are far from the depot which means that agents move long distances without the opportunity to pick up targets along the way. Radial objective produces an outcome that resembles the known-best solution the most. In the regions with higher density, the rays are thinner and the agents follow pretty much a straight path. On lower density regions, the rays are thicker, allowing MIO to obtain a similar pattern as in the known-best solution. The only issue are the outskirts, where agents cannot investigate a wider region, which affects the number of trips. Concentric is least useful in this dataset as well. The agents need to travel long distances without the opportunity to visit targets along the way and cluster shapes are also more restrictive, forcing them to follow the concentric bands. The Hybrid objective provides a result between Radial and Concentric in quality. On this particular instance it does not produce a particularly good result because the depot is eccentric and nearest targets to it are far away, making the angle component affect less than the distance. Having said that, we feel that a parameterized version of Hybrid, where a weight is assigned to the two characteristics (angle and distance) would function better for this instance. Finally, the Neighbors produces a poor result in this case. First because the clusters need agents to travel long distances to them (same as Basic), but also because of artifacts caused by the bucketing step which in few cases produces disconnected clusters, causing an increase in distance. We believe that if cluster separation would not happen during the bucketing step, SLINK would perform slightly better than the Basic variant, but still worse than Radial.

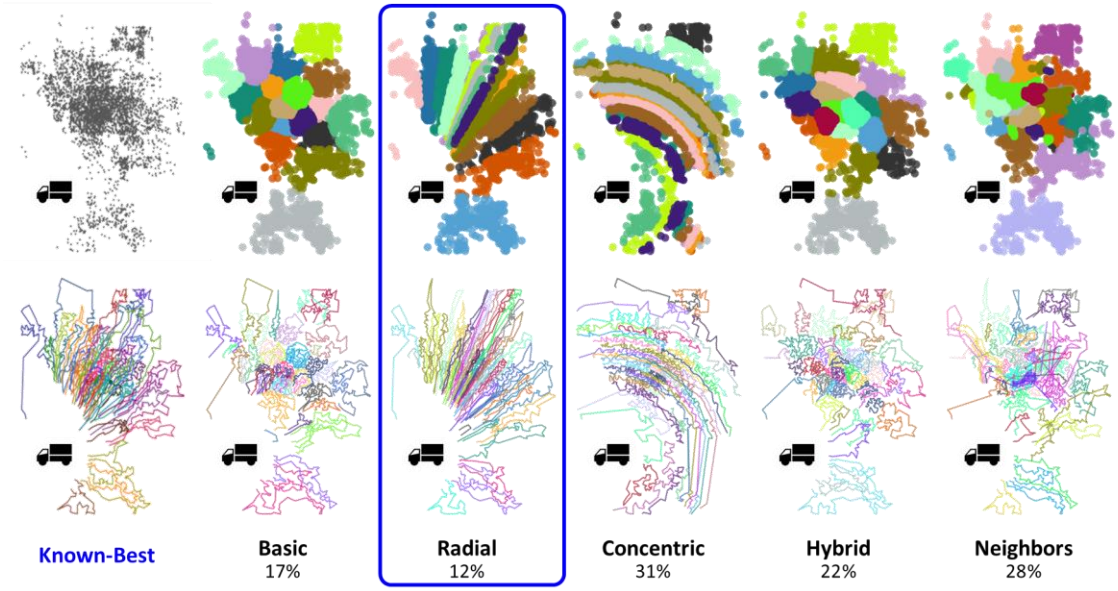


Figure 34: Instance Leuven 2 from the Large CVRP dataset and its known-best solution. Instance is clustered using all objective functions and MIO solutions are shown with respective deltas.

In conclusion, the Radial objective is more fitting when dividing instances of the type in the **Artificial CVRP** datasets. With the observation that a parameterized version of Hybrid could potentially yield better results.

7.6 How to select the objective function

We noticed how different clustering objectives performed in different scenarios and conclude that there is no one single winner in our collection and choosing the proper one depends on properties of the instance. For **Realistic VRPTW** instances with time windows and realistic travel using the road network, the Neighbors objective achieved the best allocation. However, on easier instances where allocation was not a problem (usually low workload) the Basic objective worked slightly better. On **Artificial CVRP** instances, where travel is as-the-crow-flies with capacities but no time windows the ideal objective is Radial. We, can therefore, use a decision tree to select which objective to apply depending on the properties of the instance (see Figure 35). Now, this decision tree is only a stump, however, as more requests are made to the Multi Itinerary Optimization service, we will attempt to divide each instance using multiple objectives and record which is the most suitable one in each case. We envision that extracting features from the road topology and problem instance itself we will be able to construct a larger tree capable of choosing the ideal objective in each case.

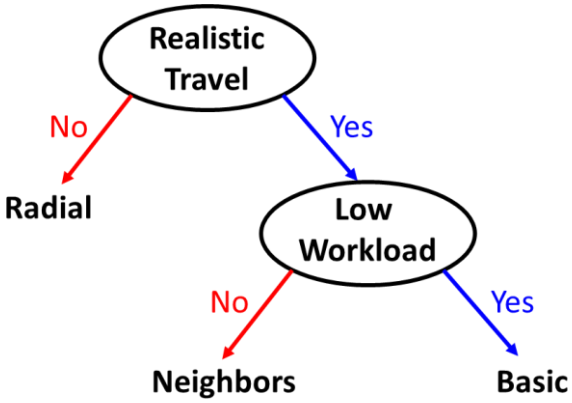


Figure 35: Example decision tree that chooses the clustering objective based on properties of the instance.

7.7 Time complexity and processing time

We give a description of the framework components in terms of their time complexities and also give the expected running time for a large instance of 10k targets and 1000 agents. We consider a **Realistic VRPTW** instance which uses the Distance Matrix API to generate travel-time-estimates using an overhead graph with 200 nodes, followed by the MDS projection (see Table 4). The travel-estimates take approximately a minute to compute with time complexity being governed by the multidimensional scaling $O(N^3)$ which we remind that we do for a maximum of 1000 subsampled locations. This step is followed by clustering the targets, where the compatibility graph clustering takes the longest time. This is followed by one of the X-Means variants or SLINK, which all process in roughly 10 seconds, despite having different time complexities. The Radial, Concentric and Hybrid objectives are slower if multiple focal points F are added. In practice, we consider a maximum of 10 focal points obtained by K-Means clustering of all focal points with $K = 10$ and do not consider this to be a bottleneck. The agent assignation is one of the slowest steps of the division process due to the Hungarian algorithm with cubic time complexity. However, this depends on the number of agents M, which is far less than N, the number of targets. The compatibility maximization step performs multiple demand-estimations with the minimum spanning tree being the slowest component $O(N^2 \log N)$ and it needs to repeat as many times as updates are done to the clusters π, usually this is a small number. MIO is expected to perform in 80 seconds on each cluster with size 300, however, this step is in parallel and in principle, all clusters can be solved in just 80 seconds assuming many machines are available. The merging step is the fastest and just combines the result (O(N)). We symbolize K-Means iterations as i.

Table 4. Time Complexity and Running Time

Steps		**Complexity**	**Running Time**
Generating Travel-Estimates (1 minute total)		Kmeans: $O(iN^2)$ Distance Matrix API MDS Projection $O(N^3)$	5 seconds 15 seconds 30 seconds
Clustering Targets (30 seconds total)	Compatibility	$O(N^2)$	20 seconds
	X-Means +balancing Basic	$O(iN^2)$	10 seconds
	Radial	$O(iN^2F)$	10 seconds
	Concentric	$O(iN^2F)$	10 seconds
	Hybrid	$O(iN^2F)$	10 seconds
	Neighbors (**SLINK**)	$O(N^2)$	10 seconds
	Balanced K-Means	$O(ikN^3)$	Not applicable
Agent assignation		$O(M^3)$	35 seconds
Compatibility maximization		$O(\pi N^2 \log N)$	10 seconds
MIO			80 seconds
Merging		O(N)	<1 second

In conclusion, the framework is expected to divide the large instance in approximately 2 minutes, execute MIO in parallel with a theoretical best running time of 80 seconds if each cluster is sent to a warmed up machine with an empty queue. In practice, we expect the entire process to complete in less than 10 minutes, which is suitable for an online service.

8 Conclusion

In this paper, we presented VRPDiv, a framework to divide large vehicle routing problems. Integrating the VRPDiv with the Multi Itinerary Optimization service[14] enhances it, allowing us to handle two orders of magnitude larger requests (>10k) in a reasonable amount of time for an online service (<10 minutes). VRPDiv uses real-world travel information to generate itineraries that are feasible in practice by following on-board navigational instructions.

We demonstrated how each framework component influences the quality of the solution: target allocation was overall improved by a total of 40% when agents are assigned based on cluster demands (25%) and accounting for compatibility (15%). Allocation is also improved by clustering using estimated travel-times (5%) which also reduces the total duration of itineraries (by 9.8%).

When compared to state-of-the-art results, we get an average delta of 10% and <5% on instances with low workload which are easier to divide. Both the division and MIO contribute to this error. This compromise, however, is made with the benefit of having a system that is usable in practice from both a processing duration perspective (user waiting time) and real world constrains (realistic travel times). We consider this trade to be a good starting point for the system which we are improving as we speak.

We experimented with four clustering algorithms and five objective functions and found that methods that produce less clusters are better because fewer cluster borders mean a lower probability for interference with the optimal itineraries. Two objective functions stood out in our experiments: Neighbors and Radial, however, all others were also useful at times. Moreover, when Neighbors was most useful, Radial was not and vice-versa, suggesting that the ideal objective function to use depends on properties of the instance.

These findings open up avenues in the fields of clustering. More specifically, what clustering algorithms and which objective functions are suitable in case of VRP. We have yet to explore fuzzy [42] and density-based methods [43], which may be ideal in some scenarios. Supervised machine learning (classifiers) can be used to choose the correct objective function for a given instance. This needs further exploration when more large VRP instances become available. The framework can be experimented using other *Solvers* as well (such as optimal ones or those tuned for a specific VRP scenario). To facilitate advancing research in this field we also made the **Realistic VRPTW** dataset publicly available and encourage the use of the Overhead Graph [19] to estimate travel in the region.

REFERENCES

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

[2] Golden, B. L., Raghavan, S., & Wasil, E. A. (Eds.). (2008). The vehicle routing problem: latest advances and new challenges (Vol. 43). Springer Science & Business Media.

[3] Bräysy, O., Dullaert, W., & Gendreau, M. (2004). Evolutionary algorithms for the vehicle routing problem with time windows. Journal of Heuristics, 10(6), 587-611.

[4] Gendreau, M., Potvin, J. Y., Bräumlaysy, O., Hasle, G., & Løkketangen, A. (2008). Metaheuristics for the vehicle routing problem and its extensions: A categorized bibliography. In The vehicle routing problem: latest advances and new challenges (pp. 143-169). Springer, Boston, MA.

[5] Braekers, K., Ramaekers, K., & Van Nieuwenhuyse, I. (2016). The vehicle routing problem: State of the art classification and review. Computers & Industrial Engineering, 99, 300-313.

[6] Li, et al. (2019). A pedestrian level strategy to minimize outdoor sunlight exposure in hot summer. arXiv:1910.04312.

[7] Krumm, & Horvitz (2017). Risk-Aware Planning: Methods and Case Study for Safer Driving Routes. In AAAI (pp. 4708-4714).

[14] https://www.microsoft.com/en-us/maps/multi-itinerary-optimization

[8] Kuo & Wang (2011). Optimizing the VRP by minimizing fuel consumption. Management of Environmental Quality: An International Journal.
[9] Lin et al. (2014). Survey of green vehicle routing problem: past and future trends. Expert systems with applications, 41(4), 1118-1138.
[10] Gagliano, A., Villani, P. G., Manelli, A., Paglia, S., Bisagni, P. A., Perotti, G. M., ... & Lombardo, M. (2020). COVID-19 epidemic in the middle province of Northern Italy: impact, logistics, and strategy in the first line hospital. Disaster medicine and public health preparedness, 1-5.
[11] Applegate, D. L., Bixby, R. M., & Chvátal, V. Cook (2006). The Traveling Salesman Problem.
[12] Sengupta, L., Mariescu-Istodor, R., & Fränti, P. (2019). Which Local Search Operator Works Best for the Open-Loop TSP?. Applied Sciences, 9(19), 3985.
[13] Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., & Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. Operations research, 46(3), 316-329.
[14] Cristian, A., Marshall, L., Negrea, M., Stoichescu, F., Cao, P., & Menache, I. (2019, November). Multi-Itinerary Optimization as Cloud Service (Industrial Paper). In Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (pp. 279-288).
[15] Helsgaun, Keld. "An effective implementation of the Lin–Kernighan traveling salesman heuristic." European Journal of Operational Research 126, no. 1 (2000): 106-130.
[16] Taillard & Helsgaun (2019). POPMUSIC for the travelling salesman problem. European Journal of Operational Research, 272(2), 420-429.
[17] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms. MIT press.
[18] Reinelt (1991). A traveling salesman problem library. INFORMS J. Comput. 3, 376–384.
[19] Mariescu-Istodor, Radu, and Pasi Fränti. "Fast travel-distance estimation using overhead graph." Journal of Location Based Services (2021): 1-19.
[20] Vidal, Thibaut. "Hybrid Genetic Search for the CVRP: Open-Source Implementation and SWAP* Neighborhood." arXiv preprint arXiv:2012.10384 (2020).
[21] Oyola, Jorge, and Arne Løkketangen. "GRASP-ASP: An algorithm for the CVRP with route balancing." Journal of Heuristics 20, no. 4 (2014): 361-382.
[22] Zhu, Kenny Qili. "A new genetic algorithm for VRPTW." In Proceedings of the international conference on artificial intelligence. 2000.
[23] Qi, Chengming, and Yunchuan Sun. "An improved ant colony algorithm for VRPTW." In 2008 International Conference on Computer Science and Software Engineering, vol. 1, pp. 455-458. IEEE, 2008.
[24] Geisberger, R., Sanders, P., Schultes, D., & Delling, D. (2008, May). Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In International Workshop on Experimental and Efficient Algorithms (pp. 319-333). Springer, Berlin, Heidelberg.
[25] Geisberger, R., & Vetter, C. (2011, May). Efficient routing in road networks with turn costs. In International Symposium on Experimental Algorithms (pp. 100-111). Springer, Berlin, Heidelberg.
[26] Forgy, E. W. (1965). Cluster analysis of multivariate data: efficiency versus interpretability of classifications. biometrics, 21, 768-769.
[27] MacQueen, J. (1967, June). Some methods for classification and analysis of multivariate observations. In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability (Vol. 1, No. 14, pp. 281-297).
[28] Lloyd, S. (1982). Least squares quantization in PCM. IEEE transactions on information theory, 28(2), 129-137.
[29] Malinen, M. I., & Fränti, P. (2014, August). Balanced k-means for clustering. In Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR) (pp. 32-41). Springer, Berlin, Heidelberg.
[30] Burkard, R., Dell'Amico, M., & Martello, S. (2012). Assignment problems, revised reprint (Vol. 106). Siam.
[31] Pelleg, D., & Moore, A. W. (2000, June). X-means: Extending k-means with efficient estimation of the number of clusters. In Icml (Vol. 1, pp. 727-734).
[32] Fränti, P., & Kivijärvi, J. (2000). Randomised local search algorithm for the clustering problem. Pattern Analysis & Applications, 3(4), 358-369.
[33] Rokach, Lior, and Oded Maimon. "Clustering methods." Data mining and knowledge discovery handbook. Springer US, 2005. 321-352..

[34] R. Sibson (1973). "SLINK: an optimally efficient algorithm for the single-link cluster method". The Computer Journal. British Computer Society. 16 (1): 30–34. doi:10.1093/comjnl/16.1.30.
[35] Yiu, M. L., & Mamoulis, N. (2004, June). Clustering objects on a spatial network. In Proceedings of the 2004 ACM SIGMOD international conference on Management of data (pp. 443-454).
[36] Wickelmaier, F. (2003). An introduction to MDS. Sound Quality Research Unit, Aalborg University, Denmark, 46(5), 1-26.
[37] Kaiser, C., Walsh, F., Farmer, C. J., & Pozdnoukhov, A. (2010, September). User-centric time-distance representation of road networks. In International Conference on Geographic Information Science (pp. 85-99). Springer, Berlin, Heidelberg.
[38] Marshall, L., & Tankayev, T. Practical Risk Modeling for the Stochastic Technician Routing and Scheduling Problem.
[39] Arnold, F., Gendreau, M., & Sörensen, K. (2019). Efficiently solving very large-scale routing problems. Computers & Operations Research, 107, 32-42.
[40] Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., & Subramanian, A. (2017). New benchmark instances for the capacitated vehicle routing problem. European Journal of Operational Research, 257(3), 845-858.
[41] Boscoe, F. P., Henry, K. A., & Zdeb, M. S. (2012). A nationwide comparison of driving distance versus straight-line distance to hospitals. The Professional Geographer, 64(2), 188-196.
[42] Bezdek, James C. Pattern recognition with fuzzy objective function algorithms. Springer Science & Business Media, 2013.
[43] Ester, Martin, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. "A density-based algorithm for discovering clusters in large spatial databases with noise." In Kdd, vol. 96, no. 34, pp. 226-231. 1996.
[44] Liu, Huiping, Cheqing Jin, Bin Yang, and Aoying Zhou. "Finding top-k optimal sequenced routes." In 2018 IEEE 34th International Conference on Data Engineering (ICDE), pp. 569-580. IEEE, 2018.
[45] Sharifzadeh, Mehdi, Mohammad Kolahdouzan, and Cyrus Shahabi. "The optimal sequenced route query." The VLDB journal 17, no. 4 (2008): 765-787.
[46] Yang, Bin, Chenjuan Guo, Christian S. Jensen, Manohar Kaul, and Shuo Shang. "Stochastic skyline route planning under time-varying uncertainty." In 2014 IEEE 30th International Conference on Data Engineering, pp. 136-147. IEEE, 2014.
[47] Yang, Bin, Jian Dai, Chenjuan Guo, Christian S. Jensen, and Jilin Hu. "PACE: a PA th-CE ntric paradigm for stochastic path finding." The VLDB Journal 27, no. 2 (2018): 153-178.
[48] Christofides, Nicos. Worst-case analysis of a new heuristic for the travelling salesman problem. Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.
[49] Arora, Sanjeev. "Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems." Journal of the ACM (JACM) 45, no. 5 (1998): 753-782.
[50] Applegate, David L., Robert E. Bixby, Vašek Chvátal, and William J. Cook. The traveling salesman problem. Princeton university press, 2011.
[51] Kelly, Charlotte, Claire Hulme, Tracey Farragher, and Graham Clarke. "Are differences in travel time or distance to healthcare for adults in global north countries associated with an impact on health outcomes? A systematic review." BMJ open 6, no. 11 (2016): e013059.