Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)

The Clustered Dial-a-Ride Problem

Fabian Feitsch
University of Würzburg
mail@fabian-feitsch.de

Sabine Storandt
University of Konstanz
sabine.storandt@uni-konstanz.de

Abstract

We study a variant of the classical dial-a-ride problem, with an application to public transport planning in rural areas. In the classical dial-a-ride problem, n users each specify a pick-up and a delivery location, and the aim is to plan the least cost route to cater all requests. This can be modeled as a traveling salesmen problem in a complete graph with precedence constraints (pick-ups need to happen before deliveries). In this paper, we consider the clustered dial-a-ride problem, where we do not operate on a complete graph but on a graph composed of serially numbered cliques where each clique is connected to the next one via a single edge. This setting is inspired by door-to-door transportation for people from remote villages who want to get to another village or the next city by a bus which operates on demand. We argue that in case the optimal route exhibits certain structural properties, it can be computed significantly faster. To make use of this observation, we devise a classification algorithm which can decide whether the optimal route exhibits these structural properties *before* computing it. Extensive experiments on artificial and real-world instances reveal that the majority of optimal routes indeed have the desired properties and that our classifier is an efficient tool to recognize the respective instances.

Introduction

There has been rising adoption of on-demand transportation in the last decades, globally. On-demand transportation services allow users to book a vehicle that picks them up from either home or a convenient nearby location, and take them to their desired destination. Taxis are still the prevalent incarnation of on-demand transportation. But they also exhibit some disadvantages as being rather expensive (especially on longer trips) and not being environment-friendly due to accommodating often only a single passenger. Companies as Uber or Lyft provide platforms for (cheap) ride-sourcing but this also leads to even more clogged streets in urban areas—while in rural areas both taxis and ride-sourcing services are only of very limited availability.

Unfortunately, also scheduled public transportation has poor coverage in many rural areas. With sparse bus stops and long waiting times between buses (especially at night time), the attractiveness of using this mode of transportation is often low. One interesting remedy are dial-a-bus services. Here people can reserve a seat on a (small) bus for door-to-door transportation. It is then critical to plan the bus route such that all requests are dealt with and only small detours are introduced for each rider when picking up and delivering other passengers.

We will formalize this rural bus route planning problem as a variant of the classical dial-a-ride problem. While such (NP-hard) problems are often tackled with heuristics in practice, we carefully study structural properties of optimal solutions which then allow us to compute optimal routes on real-world instances efficiently.

Related Work

There exists a plethora of work on dial-a-ride problems, also under terms as demand responsive transport, traveling salesman problem with pick-ups and deliveries (Dumitrescu et al. 2008), or traveling salesman problem with precedence constraints (Ascheuer, Jünger, and Reinelt 2000). In a broader context, dial-a-ride problems belong to the family of vehicle routing problems in which a single vehicle or a fleet of vehicles has to deliver certain amounts of goods from depots to customers (Dantzig and Ramser 1959). Psaraftis augmented the well known Held-Karp algorithm (Held and Karp 1962) for solving dial-a-ride problems (Psaraftis 1980). Other typical approaches to solve such problem types are ILP formulations or branch-and-bound algorithms (Cordeau et al. 2010). But as all of the mentioned problems are NP-hard in general, these exact algorithms are only applicable to small instances.

The dial-a-ride problem is also closely related to ride sharing problems. Algorithms to find good matchings of drivers and customers have been proposed e.g. in (Geisberger et al. 2009) and (Alonso-Mora et al. 2017). After the matching of offers and requests is done, the actual route is either computed with an exhaustive search (if the capacity of the vehicle is small), or heuristics are used (Alonso-Mora et al. 2017). In our envisioned dial-a-bus application, the number of seats is larger than in a conventional car. Hence exact algorithms for ride sharing cannot easily be applied.

There is also work on the clustered version of the traditional TSP problem (CTSP). In CTSP, the points are partitioned in predefined clusters and all points inside one cluster must be visited consecutively. Ding et. al. present a genetic algorithm that gives good heuristic results on CTSP

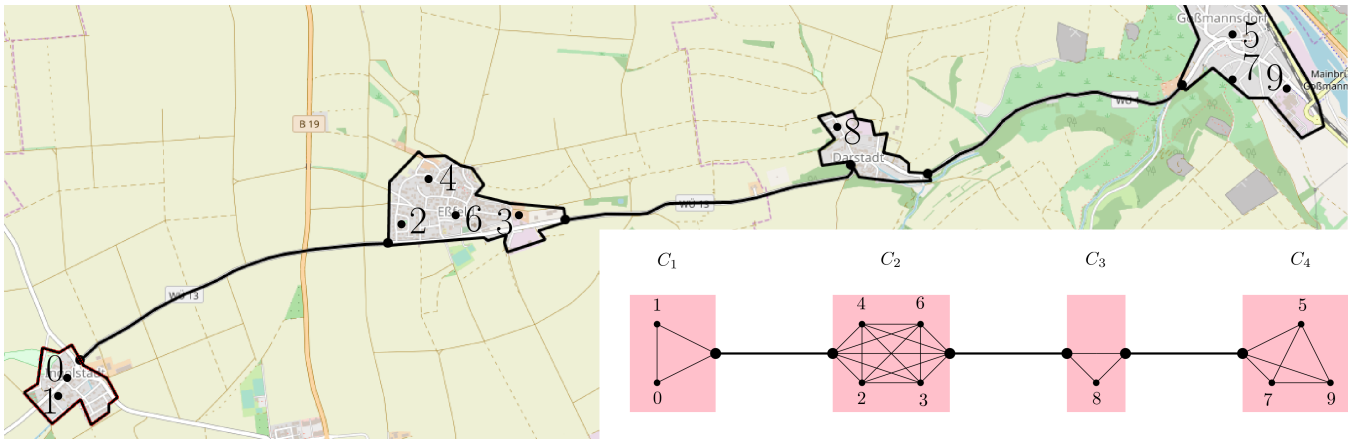

Figure 1: Clustered dial-a-ride instance in a real road network and the corresponding cluster graph.

instances (Ding, Cheng, and He 2007). Unfortunately, the optimal TSP tour through a CTSP instance often does not obey the clusters, so the lengths of the shortest routes of TSP and CTSP differ in general.

Contribution

We introduce the clustered dial-a-ride problem and discuss its applicability to real-world instances. We argue that it captures the characteristics of on-demand transportation planning problems in rural areas better than previous models.

We then investigate structural properties of optimal bus routes on clustered inputs. In particular, we focus on so called unidirectional routes where a cluster once left by the bus is never visited again. We show that for instances with a unidirectional optimal route, exact algorithms can be instrumented to run significantly faster than on general instances. We design a classifier that allows to distinguish between instances with unidirectional optimal routes and others without having to compute the optimal route a priori. This allows to construct an exact algorithm which first classifies the instance and then depending on the outcome uses the fastest algorithm that ensures getting the optimal solution. Note that omitting the classification step but always using the faster version of the algorithm also yields a feasible heuristic.

In our experiments on artificial and real data, we demonstrate the effectiveness of our approach and study its sensitivity to parameters as the average inter-cluster distance.

Preliminaries

In this section, we first formally define our bus route planning problem as an optimization problem which aims at minimizing the total riding time of all riders. Subsequently, we discuss an exact baseline algorithm to solve the problem.

Formal Problem Definition

In the most basic dial-a-ride problem, we are given n pairs of pick-up and drop-off locations, as well as all pairwise distances between those $2n$ locations in form of a matrix d, and the goal is to compute the cheapest route that transports each of the n riders from his pick-up to his drop-off location.

In our scenario, we also consider the bus driver as a special kind of rider. Hence we deal with $m = n + 1$ persons. The bus driver also has a dedicated start and end location (e.g. bus depots), providing us with a list of $2m$ locations that need to be visited on the route. For easier reference, we enumerate the locations using the following scheme: Location 0 is the start (aka pick-up) location of the driver, locations 1 to n are the pick-up locations of the riders in arbitrary order, and locations $n+1 = m$ to $2m-1$ refer to the drop-off locations in the same order as the pick-ups.

Definition 1 (Tour)**.** *A feasible* tour *is a permutation T of $\{0, \dots, 2m-1\}$ such that $T[1] = 0$, $T[2m] = m$ and $T^{-1}[i] < T^{-1}[i+m]$ for all $i = 1, \dots, m$, so every pick-up precedes its corresponding drop-off.*

Inspecting real-world road networks in rural areas, we often observe similar structures, as exemplarily depicted in Figure 1: Remote villages are connected by a single bigger road, which usually leads to the next town. Hence a bus does not really have a choice how to get from one village to another but the route in between is determined by that bigger road and other villages aligned on that street can not by bypassed even if there are no pick-ups or drop-offs within. This inspires our interpretation of the villages as a sequence of clusters $C_1, \cdots, C_q$ in the order they appear along the road. Each cluster $C_i \subseteq [0, 2m-1]$ contains a subset of the pick-up/drop-off locations. Furthermore, with each cluster we associate two portal nodes over which the cluster can be entered or exited. So two clusters C_i and C_{i+1} are connected via an edge from the exit portal of C_i to the access portal of C_{i+1} where the edge represents the connecting road. Apart from those, there are no connections between the clusters. Within each cluster the contained locations and the portals form a clique. We refer to the resulting graph as the cluster graph, see Figure 1 (lower right corner).

Now we can define the clustered dial-a-ride problem:

Definition 2 (The Clustered Dial-a-ride Problem)**.** *We are given a weighted, undirected cluster graph with metric edge weights, and with the following restrictions: Location 0 is contained in C_0, location m in C_q. For each rider, the clus-*

ter C_i containing the pick-up location and the cluster C_j containing the drop-off location have to obey $i \leq j$. The goal is to find a feasible tour T in the cluster graph with minimum cost, where the cost is defined as

$$c(T) = \sum_{i=2}^{2m} w(i) \cdot d\big[T[i-1], T[i]\big]$$

with $w(i)$ denoting how many persons are on board while traveling from location $T[i-1]$ to $T[i]$ and $d\big[T[i-1], T[i]\big]$ being the shortest path distance between those two locations in the cluster graph.

The restriction of only allowing requests 'from left to right' is sensible as it reflects the case that there is one bus going towards the next town, and another one going from the town to the villages. Hence we can subdivide the requests based on their direction and then solve these two instances separately. The objective function then sums up the distances that each rider (including the driver) travels.

Psaraftis' Algorithm

Note that every algorithm solving the classical dial-a-ride problem can also solve the clustered version by simply ignoring all information about the clusters.

Psaraftis presented such an algorithm with a running time of $O^*(3^{n-1})$ (Psaraftis 1980). Due to the recursive nature of Psaraftis' algorithm it is also able to solve partial dial-a-ride instances. A *partial* instance consists of the current bus location as well as two state vectors s_a, s_b in which each rider is assigned a state $s \in \{w, t, f\}$, indicating that he is either still waiting to be picked up (w), currently travelling on the bus (t) or already dropped off and hence finished (f). The goal is to find the optimal tour that starts with the state vector s_a and ends when the state vector s_b is reached. Note that every dial-a-ride instance can be regarded as partial dial-a-ride instance with $s_a = [w]^n$ and $s_b = [f]^n$.

Even though the runtime of Psaraftis' algorithm is exponential, it is fast for small inputs or partial instances in which s_a is similar to s_b. This is a crucial property which will help to exploit the structure of clustered dial-a-ride instances.

Instance-based Classification

As already outlined above, exact algorithhms for the general dial-a-ride problem are usually limited in practice to small instances. For our clustered dial-a-ride problem, we make the following important observation: Intuitively, the optimal tour handles the clusters in the order of their enumeration ('from left to right') without ever going back to a cluster which was exited before. We call a tour which never returns to an earlier cluster an *unidirectional* tour and refer to the optimal unidirectional tour by $\vec{T^*}$. We will next show that in case the intuition applies, we can design an algorithm based on Psaraftis' algorithm which solves clustered dial-a-ride instances significantly faster than conventional dial-a-ride instances. Unfortunately, we will then provide examples of clustered dial-a-ride instances where the optimal tour is not unidirectional. To still be able to make use of the faster algorithm for clustered dial-a-ride instances with unidirectional optimal tours, we present an efficient classifier which determines whether the optimal tour is unidirectional or not without having to compute the optimal solution in the first place.

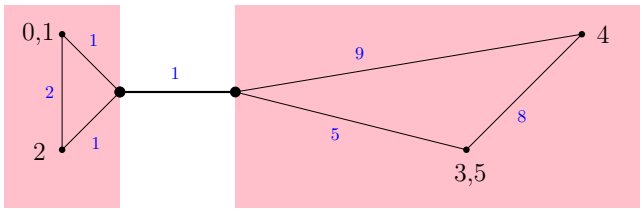

Figure 2: Example instance for $m = 3$. Location numbers are given in black, edge costs in blue. The tour has to start at location 0 and end at location 3. The best unidirectional tour would be $T = 0, 1, 2, 5, 4, 3$ with $c(T) = 2 \cdot 2 + 3 \cdot 1 + 3 \cdot 1 + 3 \cdot 5 + 2 \cdot 8 + 1 \cdot 8 = 49$. However the tour $T' = 0, 1, 4, 2, 5, 3$ has a smaller cost of $c(T') = 2 \cdot 1 + 2 \cdot 1 + 2 \cdot 9 + 1 \cdot 9 + 1 \cdot 1 + 1 \cdot 1 + 2 \cdot 1 + 2 \cdot 1 + 2 \cdot 5 = 47$ despite not being unidirectional.

The $\vec{T^*}$-Algorithm for Unidirectional Solutions

We now assume that the optimal tour is indeed unidirectional. Note that two unidirectional tours can only differ in the order of locations inside the clusters. The state of all riders when entering a cluster is fixed, as well is the state of all riders when leaving the same cluster (as the bus never returns to a previously visited cluster). More precisely, when entering cluster C_i via its access portal, all riders with drop-off locations in clusters $C_{j<i}$ are in state f (finished), all riders with a pick-up location in $C_{j \geq i}$ are in state w (waiting), and all other riders are in state t (currently traveling on the bus). When exiting C_i via the exit portal, the states of riders with drop-off and/or pick-up locations within C_i changed appropriately.

Thus, one can solve partial dial-a-ride instances for every cluster using Psaraftis' algorithm and then assemble the partial tours to get an optimal unidirectional tour $\vec{T^*}$. This approach will be referred to as the $\vec{T^*}$-algorithm. Its running time is exponential in the maximal size M of a cluster, but *not* exponential in the number of requests. Therefore, for instances where $M \ll m$ holds, the $\vec{T^*}$-algorithm runs much faster than Psaraftis' original algorithm.

Instances with $T^* \neq \vec{T^*}$

The drawback of the $\vec{T^*}$-algorithm is that it does not guarantee to find a globally optimal tour T^* on every instance because the optimal tour might indeed go back and forth between the clusters in order to minimize the objective function. Figure 2 shows a small example instance in which the optimal tour is *not* unidirectional. As also evident from the example, having small distances between the clusters but large distances within the clusters makes it more likely that the optimal tour is not unidirectional.

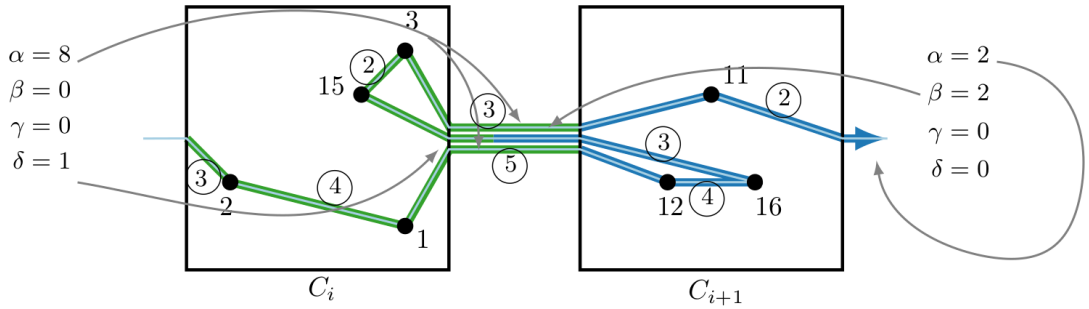

Figure 3: Values of the four counters for C_i (left) and C_{i+1} (right). The number of persons m is 10, thus locations < 10 are all pick-ups and locations ≥ 10 are all drop-offs. The rider at pick-up l wants to travel to the drop-off $l + 10$. The encircled numbers show how many passengers are on board while traveling the respective leg of the journey.

On such instances the $\vec{T^*}$-algorithm does find a feasible tour but not an optimal tour. Hence we should use the original algorithm of Psaraftis instead. In order to make that decision, we now design a fast instance classifier.

Basic Idea for Classification

We obviously can not compute the optimal solution prior to the classification as this would make the classifier useless. Hence then we can only rely on the structural characteristics of the particular instances. The idea for proving the optimality of a tour T is to distribute the costs $c(T)$ of T carefully to the clusters, and to try to match them with lower bounds for the costs within each cluster. The following definition of induced costs formalizes the distribution idea.

Definition 3 (Induced Costs)**.** *For an instance I let $\mathcal{C}$ be the set of clusters C_i and $\mathcal{T}$ the set of all feasible tours on I. A function $\Upsilon\colon \mathcal{C} \times \mathcal{T} \to \mathrm{R}^+$ which has the property that $\sum_{C_i \in \mathcal{C}} \Upsilon(C_i, T) = c(T)$ for all tours T represents the* induced costs $\Upsilon(C_i, T)$ *of cluster C_i in T.*

Let $\Phi(C_I) \leq \Upsilon(C_i, T^*)$ be a lower bound on the induced costs of C_i in the optimal tour T^*. Then the following theorem holds:

Theorem 4. *If for all $C_1, \ldots, C_q \in \mathcal{C}\colon \Upsilon(C_i, \vec{T^*}) = \Phi(C_i)$, then $\vec{T^*} = T^*$, i. e. the optimal tour is unidirectional.*

Proof. Suppose that $T^* \neq \vec{T^*}$ holds. Then there is an non-unidirectional route $\overleftrightarrow{T^*}$ with $\overleftrightarrow{T^*} = T^*$. In $\overleftrightarrow{T^*}$ there must be a sequence $K = [C_i, \ldots C_{i'}]$ of contiguous clusters which are all at least visited twice with $|K| \geq 2$. The costs $c(K)$ of K are related to $\Phi(C_i)$:

$$c(K) = \sum_{j=i}^{i'} \Upsilon(C_j, \overleftrightarrow{T^*}) \geq \sum_{j=i}^{i'} \Phi(C_j) \tag{1}$$

By definition of $\Upsilon(C_i, \vec{T^*})$, the clusters in K can be handled unidirectionally with costs $c'(K)$ defined by the summation:

$$c'(K) = \sum_{j=i}^{i'} \Upsilon(C_i, \vec{T^*}) = \sum_{j=i}^{i'} \Phi(C_j) \tag{2}$$

By combining equations 1 and 2 we conclude that $c'(K) \leq c(K)$ which is a contradiction to the assumption that the optimal route was not unidirectional. □

The theorem does not depend on a fixed implementation of $\Upsilon(\cdot, \cdot)$. We will describe a suitable realization of the induced costs as well as an approach to lower bound $\Upsilon(C_i, T^*)$ in the next two subsections.

Part 1: Cost Distribution

This subsection introduces a concrete cost distribution function $\Upsilon(\cdot, \cdot)$ which also has the nice property that it can be computed in linear time.

Let r be a rider, p_r the index of his pick-up cluster and d_r the index of his drop-off cluster. For a feasible tour T and a cluster C_i, the following counters are defined: α counts the events that a rider r with $p_r \leq i$ exits to the right, and β the events that a rider r with $d_r \geq i$ exits to the left. In the same spirit, the counters γ and δ are used: γ counts how often a rider r with $p_r \geq i$ enters from the left, and last, δ counts the occurrences of riders with $d_r \leq i$, that enter from the right. Then $\Upsilon(C_i, T)$ is given by

$$\Upsilon(C_i, T) = (\alpha + \delta) \cdot d[C_i, C_{i+1}] + (\beta + \gamma) \cdot d[C_i, C_{i-1}] + \mathsf{inside}(C_i, T) \tag{3}$$

where $d[C_i, C_{i+1}]$ denotes the distance from the exit portal of C_i to the access portal of C_{i+1}, $d[C_i, C_{i-1}]$ the distance from the access portal of C_i to the exit portal of C_{i-1}, and $\mathsf{inside}(C_i, T)$ the costs of the subtour of T strictly within C_i.

Figure 3 shows an example of the induced costs of cluster C_i in a feasible tour T. In this instance, one can easily check that the sum of the induced costs of all clusters is exactly the cost of the tour T. This property is formally stated for all feasible tours in the next theorem. This theorem requires a brief lemma:

Lemma 5. *Consider a journey from C_i to C_{i+1}. Then the journey can* either *be counted by C_i's α-counter* or *by C_{i+1}'s γ-counter, but not both. The same applies for a left-bound journey between C_{i+1} and C_i.*

Proof. If the right-bound journey is covered by C_i's α-counter, then $p_r \leq i$. Yet, C_{i+1}'s γ-counter catches the journey only if the condition $p_r \geq i+1$ is true. That cannot happen since the first condition prohibits it. Thus, the γ-counter

can only be used if the α-counter of the cluster to the left does not fire. Symmetrically, the same argumentation shows that either C_i's δ-counter or C_{i+1}'s β-counter is responsible for a journey from the cluster C_{i+1} to the cluster C_i. □

With the help of this lemma the validity of the induced costs can be shown:

Theorem 6. *For every feasible tour T, the sum of induced costs according to Equation 3 equals the total cost of T:* $\sum_{C_i \in \mathcal{C}} \Upsilon(C_i, T) = c(T)$.

Proof. All costs generated inside the clusters are covered once by the last term of Equation 3, so the main task is to show that the costs that are generated between the clusters are counted exactly once. We pick an arbitrary journey of one person between the two clusters C_i and C_{i+1}. If this person is the driver and the journey goes from cluster C_i to cluster C_{i+1}, then the α-counter of C_i carries this journey and no counter of C_{i+1} counts the same movement. This applies for a journey of C_{i+1} to C_i symmetrically.

If the journey's person is a rider r and the journey goes from C_i to C_{i+1}, then it can only be counted by C_i's α counter or C_{i+1}'s γ counter. It suffices to show that at least one of these counters catches the journey. The previous lemma ensures that no other counter catches the same journey. The two locations of r can lie in five different ways relative to C_i:

Both locations are left of C_i. This case occurs when $p_r < d_r \leq i$, i. e. the rider travels further than he needed. This journey is counted by C_i's α because $p_r < i$.

Pick-up left of C_i and drop-off in or right of C_i. This is the case if $p_r < i \leq d_r$. Since $p_r \leq i$, it is counted by C_i's α counter.

Both locations lie in C_i. Then $p_r = i = d_r$ and C_i's α counter is responsible for counting the journey.

Pick-up left of or in C_i and drop-off right of C_i This condition can be expressed mathematically as $p_r \leq i < d_r$. It is counted by C_i's α counter because $p_r \leq i$.

Both locations are right of C_i. In formal terms, $i < p_i \leq d_i$ and thus, only C_{i+1}'s γ counter considers this journey.

The same steps can be applied to a right-to-left journey and C_i's δ counter and C_{i+1}'s β counter. Therefore, every journey is counted exactly once in $\sum_{C_i \in \mathcal{C}} \Upsilon(C_i, T)$. □

The next subsection introduces a non-trivial lower bound on $\Upsilon(C_i, T^*)$ which can be computed in bearable time for moderate cluster sizes.

Part 2: Cost Estimation

This subsection presents a method to compute a lower bound $\Phi(C_i) \leq \Upsilon(C_i, T^*)$. Note that it is easy to come up with just some lower bound but for our classifier to be useful in pratice, we require lower bounds that are tight. Our lower bounding technqiue is based on the observation that a tour that visits C_i on several subtours partitions the cluster into subsets. Thus the approach is to enumerate all possible partitions and compute a lower bound for each of them. The smallest of these lower bounds is the desired value.

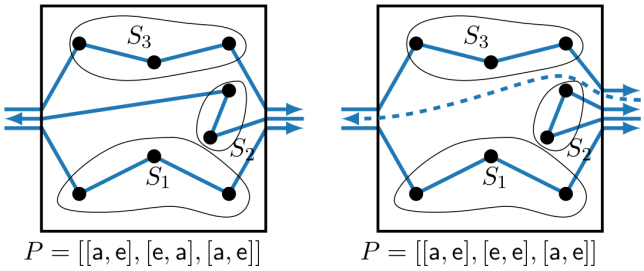

Figure 4: The pictures show two different paths that realize the same partition $\mathcal{S} = S_1, S_2, S_3$ of the cluster but different vectors P. On the right side, the bus has to traverse C_i without visiting a location inside C_i to make the tour possible. This journey is indicated as dashed line.

Let C_i be a cluster and $\mathcal{S}$ an ordered partition of C_i into subsets $S_1, \ldots, S_k$. Then $\Phi_{\mathcal{S}}(C_i)$ represents the lower bound of serving cluster C_i via the subsets defined by $\mathcal{S}$. The overall lower bound $\Phi(C_i)$ is given by Equation 4.

$$\Phi(C_i) = \min_{\mathcal{S} \text{ ordered partition of } C_i} \Phi_{\mathcal{S}}(C_i) \tag{4}$$

For a given partition $\mathcal{S}$ of C_i there are several ways to connect the subsets inside S because every subset $S \in \mathcal{S}$ can be entered from the left or from the right of the cluster. Analogously, the bus can leave the cluster through the access portal or through the exit portal. Therefore, for $k = |\mathcal{S}|$ there are 2^{2k} possibilities to realize $\mathcal{S}$. Let P be a list of length k of 2-tuples, where the i-th tuple in P determines through which portal node the i-th set of $\mathcal{S}$ is entered and exited. We use the symbols a and e to refer to the access portal or the exit portal. Figure 4 illustrates these concepts.

Let now $\Phi_{\mathcal{S},P}(C_i)$ denote a lower bound to handle the partition $\mathcal{S}$ with respect to the P. Then the value of $\Phi_{\mathcal{S}}(C_i)$ can be calculated with the following equation.

$$\Phi_{\mathcal{S}}(C_i) = \min_{P \in (\{\mathsf{a},\mathsf{e}\}^2)^k} \Phi_{\mathcal{S},P}(C_i) \tag{5}$$

Now it remains to compute $\Phi_{\mathcal{S},P}(C_i)$ itself. This is a four step procedure in which every step takes care about different parts of the cost, as illustrated in Figure 5.

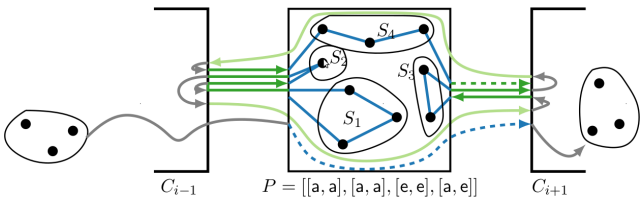

Figure 5: Lower bounding the cost for a given partition S and a vector P. The dashed green line shows the costs of which step 1 takes care, step 2 is responsible for the costs depicted in solid blue. Costs bounded in step 3 are drawn in solid green and step 4 is represented in dashed blue. The gray lines are not part of the costs of cluster C_i but rather assist the reader to trace the route.

Step 1 collects the costs at the beginning and at the end of $\mathcal{S}$. In order to get those we compute the number of riders that must inevitable be inside the bus when S_1 is entered and S_k is exited.

Step 2 covers all costs generated inside the cluster C_i. To this end, for every $S \in \mathcal{S}$ we compute the initial and final state vector of all rides before serving S and afterwards. These state vectors are used to define partial dial-a-ride instances which are then solved by Psaraftis' algorithm.

Step 3 consists of aggregating the costs happening outside of C_i, but caused by riders with at least one location in C_i. These are the costs which are covered by the counters α, β, γ and δ, hence we need to compute up with bounds for those. This can be achieved in a similar way as in the first step by keeping track of riders which are inevitably on the bus on intermediate legs of the tour.

Step 4 deals with the costs generated by riders passing through C_i, so riders whose pick-up cluster is left of C_i and whose drop-off cluster is right of C_i. Those riders are called *hoppers*. There are three ways to transport the hoppers through C_i of which the cheapest one is chosen. The first possibility is to fetch them in an extra tour after C_i was handled. The second possibility is only available if there exists a $S \in \mathcal{S}$ which is entered from the left and exited to the right. The hoppers can be transported through C_i using the tour handling S. The costs are h times the length of the shortest left-right-tour through any such S, if there is one. The last possibility exists if there are two consecutive tuples $(\cdot, \mathsf{a})$ and $(\mathsf{e}, \cdot)$ in P. This constellation means that the bus must drive directly through the cluster from its access portal to its exit portal.

Exact Classification-Based Algorithm

Combining all of the results obtained above, we are now ready to state the overall algorithm for tackling clustered dial-a-ride instances. The high-level pseudo-code is provided in Algorithm 1.

The correctness of the algorithm follows from Theorem 4 and the validity of the lower bounds on $\Upsilon(C_i, T^*)$.

In case the unidirectional tour is optimal, the running time of the algorithm is dominated by the time to compute the lower bounds $\Phi(C_i)$ in the for-loop. Here we need to iterate over all possible ordered partitions of C_i and all possible vectors P. The number of ordered partitions $p(n)$ of a set of size n is described by the ordered Bell number (or Fubini number), which can be computed recursively via $p(n) = \sum_{i=1}^{n} \binom{n}{i} p(n-i)$ and $p(0) = 1$ (Gross 1962). To account for the possible choices of the vector P as well, we have to consider $p'(n)$ cases with $p'(n) = \sum_{i=1}^{n} 4\binom{n}{i} p'(n-i)$ and $p'(0) = 1$. For $n = 6$, we already need to check 5,227,236 configurations, for which we need to solve partial instances via Psaraftis' algorithm. Nevertheless, we show in our experiments that for moderate cluster sizes our classifier is significantly faster than running Psaraftis' algorithm on the whole instance. Hence in case the unidirectional tour is not optimal, the time overhead induced by the classifier is negligible.

Note that if we just omit lines 2–8 and output the best unidirectional tour right away, the algorithms turns into an efficient heuristic.

Algorithm 1: Classification and Exact Computation

```
   Input: clustered dial-a-ride instance I
   Output: optimal tour T*(I)
   /* compute optimal unidirectional
      tour                                   */
1  T⃗* ← result of T⃗*-algorithm on I
2  for C_i ∈ C do
3  |  Φ(C_i) ← lower bound for cost induced by C_i
   |  /* check whether induced cost and
   |     lower bound match                  */
4  |  if Φ(C_i) < ϒ(C_i, T⃗*) then
   |  |  /* in case of violation run
   |  |     Psaraftis' algorithm            */
5  |  |  T* ← result of Psaraftis' algorith on I
6  |  |  return T*
7  |  end
8  end
   /* if no violator exists return the
      optimal unidirectional tour            */
9  return T⃗*
```

Experimental Results

This section covers the performance of the classifier in view of running time and precision as well as some other more specific properties. Our implementation was written with the Go Programming Language, Version 1.10.3. Experiments were carried out on an Ubuntu Xenial 16.04.3 LTS computer with Linux core 4.13.0-36. It possessed an AMD Ryzen Threadripper 1950X 16-Core Processor (3.4 GHz) with hyper threading enabled, so 32 cores in total.

Benchmark Data Sets

To evaluate our classifier we tested it on a variety of artificial and real-world instances.

For investigating scalability and to measure the influence of certain parameters in a sound way, we created artificial clustered dial-a-ride instances where locations are points in the Euclidean plane. There, clusters are represented by squares which are placed with their centers on the x-axis. Squares were not allowed to overlap. The x-axis plays the role of the main road connecting the clusters. Then, for varying values of m, we randomly placed $2m$ locations inside the clusters (obeying the constraint that pick-ups need to be left of the respective drop-off locations). For drop-off locations we made it more likely to end up in a cluster with a high index, as this more faithfully simulates that people want to go towards a bigger town. The edge weights of the cluster graph are the Euclidean distances between the points. The maximal number of locations inside a cluster was set to 6.

To check whether our approach can cope with real instances as well, we selected existing bus lines operating in Germany and simulated user requests within the areas served by the bus line. In particular, we investigated three scenarios:

- rural bus lines (connecting six to eight small villages)

# riders (n)	6	8	10	12
Psaraftis	0.053 s	1.010 s	17.238 s	232.107 s
$\vec{T^*}$-algo.	0.001 s	0.002 s	0.003 s	0.004 s
classifier	0.504 s	0.581 s	3.810 s	4.759 s

Table 1: Running times on artificial instances.

- regional bus lines (connecting six towns along federal highways)
- one intercity bus (connecting six major German cities: Munich – Ingolstadt – Nuremberg – Erfurt – Magdeburg – Berlin)

Evaluation on Artificial Instances

Guided by measuring the diameters of typical villages and the distances of consecutive villages along a bigger road, we initially used the following parameters to create 100 artificial instances randomly: We placed eight squares with a mean width of 3 000 meters and a mean inter-cluster distance of 4 000 m (standard deviation 2 000 m). Then we placed the desired number of pick-up and drop-off locations randomly within the clusters. To measure the influence of the inter-cluster distance, we took the created instances and pushed the clusters further apart until the mean inter-cluster distance reached d while maintaining the positions of the locations relative to their clusters. With that, we created a new set of 100 instances for each $d = 4000, 5000, 6000, \ldots, 24000$ meters. Hence, for every choice of n we investigate a total number of 2100 instances.

Table 1 shows the average running times to solve these instances to optimality using Psaraftis' algorithm, as well as the timings for the $\vec{T^*}$-algorithm and the classifier. We observe that the computation of the best unidirectional route is very efficient. The running time barely increases when more riders are added. In contrast, the running times of Psaraftis' algorithm increase roughly by an order of magnitude every time two additional riders are added. The running time of the classifier is larger than that of Psaraftis' algorithm for 6 riders but significantly smaller for 12 riders. In case the classifier certifies that the computed unidirectional tour is optimal, the combined running time of the $\vec{T^*}$-algorithm and the classifier is about a factor of 50 faster than Psaraftis' algorithm.

The ratio increases further when more riders are added but the maximum number of locations to be visited in a single cluster stays the same. If we use 6 riders in all 8 clusters, we get a total of 48 riders. While the running time of our classifier barely increases compared to the running time for 12 riders (still taking less than a half minute), Psaraftis' algorithm did not produce a result within a day on such an instance. The same is true for the other exact algorithms mentioned in the related work section. Due to the exponential running time of all exact algorithms, we can always outperform them if the maximum cluster size M is significantly smaller than the number m of riders. In general, though, our approach can be seen as a framework where Psaraftis' algorithm can also be replaced by another exact algorithm. We

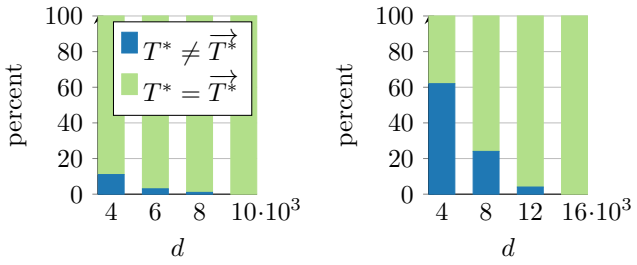

Figure 6: Percentage of instances where $T^* = \vec{T^*}$ in dependence of the mean inter-cluster distance d for $n = 6$ (left) and $n = 12$ (right).

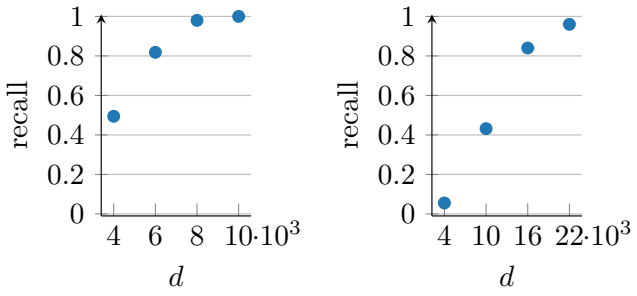

Figure 7: The classifier's recalls for different d. Left: $n = 6$, right: $n = 12$.

choose Psaraftis' here as it allows to compute partial solutions easily but it would be also interesting to try substitutes for Psaraftis' in future work.

If we would increase the maximum number of locations inside a cluster from 6 to 7, the classification time increases roughly by a factor of 30, which is still manageable. For 8 locations or more, the combinatorial explosion makes the classifier too slow. Hence 7 is the current limit our approach can handle. However, we see great potential for time reduction by exploiting redundancies among the partitions.

But now the crucial question is how often the unidirectional tour is indeed optimal and how often the classifier can certify it. Figure 6 shows that for $n = 6$ in at least 90 % of the cases we have an unidirectional optimal tour. For $n = 12$ this ratio is approximately 40 % for the smallest considered mean inter-cluster distance and then rapidly increases for growing d. Figure 7 shows the recall of the classifier for selected values of d. It gets more accurate the higher the inter-cluster distances are. For $n = 6$ and $d = 6$ kilometers, we can already certify the optimality of the unidirectional tour in over 80 % of the cases. For $n = 12$ and $d = 10$ kilometers we have a recall of roughly 40 %.

Based on these numbers we may be tempted to use the $\vec{T^*}$-algorithm as a heuristic without the classifier when dealing with many riders and clusters with a rather small distances in between. We found that the empiric approximation ratio of the $\vec{T^*}$-algorithm (the cost of the best unidirectional tour divided by the best optimal tour) is at most 1.1 among all considered instances but significantly better on average.

instance	mean ICD	% uni opt.	recall
rural 1	1.2 km	77	0.05
rural 2	3.4 km	82	0.16
rural 3	4.1 km	89	0.29
regional 1	7.9 km	55	0.59
regional 2	8.2 km	100	0.52
regional 3	10.7 km	97	0.55
intercity 1	129.0 km	100	0.93

Table 2: Results on real-world instances. ICD in the second column stands for inter-cluster distances. The third column shows the percentage of optimal tours that are unidirectional and the last column is the recall of our classifier.

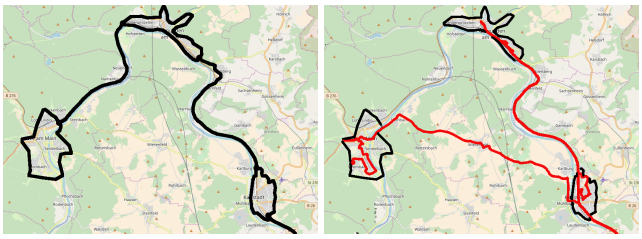

Figure 8: Left image: Intended order of clusters. Right image: Many of the optimal tours bypass the nothern city or visit it later to avoid detours for riders boarded in the leftmost town.

Hence the $\vec{T^*}$-algorithm as a standalone can be regarded as an efficient, complete heuristic.

Evaluation on Real-World Data

For our real-world scenarios (three rural bus lines, three regional bus lines, and one intercity bus line), we created 100 instances each for $n = 10$. Table 2 summarizes our results.

For the rural lines, we observe that over three-fourths of the instances have a unidirectional optimal tour with the number increasing with the mean inter-cluster distance. For villages lying very close to each other, our classifier is unfortunately often not able to certify optimality of the tours. But for a mean inter-cluster distance of 4.1 kilometers, our lower bounds are tight already for 29 % of the instances.

The regional bus line 1 was designed to push the classifier to its limits as it does not satisfy the assumption made for our classifier to work (note that this can easily be checked a priori). As shown in Figure 8, there are two towns for which the shortest path bypasses the cluster between them (because the highway follows the bends of a river). This results not only in fewer instances in which $T^* = \vec{T^*}$ but also in several false positives. In these cases the classifier said that $T^* = \vec{T^*}$ but that was not true. Note that is impossible for instances which meet our assumptions. The reason for the classifier failing is that there is a huge discrepancy between the shortest path distance of the leftmost and the rightmost town in the road network and their shortest path distance in the cluster graph used by the classifier.

In all other instances we considered, our assumptions were met and the precision of the classifier was 100 %, accordingly. For the other two regional settings (2 and 3), the recall is over 50 %, leading to a significant reduction of the average running time on these instances.

Unsurprisingly, all instances in the intercity scenario have an unidirectional optimal tour, of which the classifier found 93%. In all but one of the seven false negative instances the classifier reached only Nuremberg. Note that the distance between Nuremberg and Ingolstadt is the smallest inter-cluster distance in the scenario. But this also kept the classification time short for the instances on which we could not use the $\vec{T^*}$-algorithm.

For the intercity 1 and the regional 2 instances, the heuristic version of our algorithm where we only run the $\vec{T^*}$-algorithm would provide us with 1.0 empirical approximation factor, as all created instances exhibit a unidirectional optimal tour. For the rural 1 instances, we get an average approximation factor of 1.008 and a maximum value of 1.282. For the regional 3 instances, the average was 1.013 and the maximum was 1.100. So the empirical approximation is indeed quite good for sensible instances. Note that we could also use our lower bound computation techniques to get an upper bound on the approximation factor the $\vec{T^*}$-algorithm achieves. This also allows for an approach where Psaraftis' algorithm is only applied in case the estimated approximation factor on the unidirectional solution is too large.

Conclusions and Future Work

We introduced a classification-based algorithm for the clustered dial-a-ride problem which always returns the optimal tour and significantly saves running time compared to the baseline algorithm by Psaraftis on instances with a unidirectional optimal tour. As the cost for the classification is negligible compared to the running time of Psaraftis' algorithm in case the optimal tour is not unidirectional, it always makes sense to use the classifier.

Our experiments on over 9,000 artificial and real-world instances reveal that indeed a large percentage exhibits a unidirectional optimal tour. However, the classifier is not always able to certify the optimality. We looked at the gaps between the cost of the optimal tour and our lower bound for those instances and observed that these are usually smaller than the mean inter-cluster distance. Hence already a slight improvement in the lower bound construction could yield a significant increase of the recall. Furthermore, the running time of the classifier could possibly be reduced by exploiting redundancies among the partial instance computations or performing instance-independent preprocessing.

A sensible extension of the problem setting could be to include a constraint on the number of seats on the bus. Given a limit L on the number of riders that are allowed on the bus at the same time, there is an easy way of certifying that the optimal route is *not* unidirectional: If more than L riders have a pick-up location left of a cluster C_i but a drop-off location in C_i or right of C_i, then they can not all be transported to C_i together. In case an unidirectional tour exists, we could again try to prove its optimality via cost distribution and lower bound computation. The lower bound computation would be more complicated in this scenario, though.

References

Alonso-Mora, J.; Samaranayake, S.; Wallar, A.; Frazzoli, E.; and Rus, D. 2017. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences* 114(3):462–467.

Ascheuer, N.; Jünger, M.; and Reinelt, G. 2000. A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints. *Computational Optimization and Applications* 17(1):61–84.

Cordeau, J.-F.; Iori, M.; Laporte, G.; and Salazar González, J. J. 2010. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with lifo loading. *Networks* 55(1):46–59.

Dantzig, G., and Ramser, J. 1959. The truck dispatching problem. *Management Science* 6(1):80–91.

Ding, C.; Cheng, Y.; and He, M. 2007. Two-level genetic algorithm for clustered traveling salesman problem with application in large-scale tsps. *Tsinghua Science and Technology* 12(4):459–465.

Dumitrescu, I.; Ropke, S.; Cordeau, J.-F.; and Laporte, G. 2008. The traveling salesman problem with pickup and delivery: polyhedral results and a branch-and-cut algorithm. *Mathematical Programming* 121(2):269.

Geisberger, R.; Luxen, D.; Neubauer, S.; Sanders, P.; and Volker, L. 2009. Fast detour computation for ride sharing. *arXiv preprint arXiv:0907.5269*.

Gross, O. A. 1962. Preferential arrangements. *The American Mathematical Monthly* 69(1):4–8.

Held, M., and Karp, R. M. 1962. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics* 10(1):196–210.

Psaraftis, H. N. 1980. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science* 14(2):130–154.