Implementation and Evaluation of an On-demand Bus System

Sevket Gökay[1,2], Andreas Heuvels[1], Robin Rogner[1] and Karl-Heinz Krempels[1,2]
[1]*Informatik 5 (Information Systems), RWTH Aachen University, Aachen, Germany*
[2]*CSCW Mobility, Fraunhofer FIT, Aachen, Germany*

Keywords: Demand-responsive Transport, Dynamic Pickup and Delivery Problem, Vehicle Routing Problem with Time Windows.

Abstract: This work aims to develop and evaluate a dynamic bus system which abandons the concepts of a traditional bus service like bus line, bus station and timetable. The resulting system supports bringing customers from any location to any location, has a fleet of buses the routes of which are updated repeatedly as requests arrive and are accepted, and employs time windows in order to guarantee the desired pick-up and drop-off times of customers. We propose a technical realization and evaluate its effectiveness by running simulations in which traditional and dynamic systems are compared. Even though the operational cost and financial efficiency from a bus service provider's perspective is not the focus of this evaluation, the preliminary results show that both the provider and the customers might benefit from an on-demand dynamic system. We also hint at the feasibility of such a system in not only low-demand rural areas, but also high-demand urban regions.

1 INTRODUCTION

With the ever-improving technology landscape, traditional and existing systems are challenged with new ideas that aim for optimization by making them more intelligent. Transportation systems have always been such a target and more specifically, public transportation benefited immensely from advancements in IT and mobile Internet becoming more widespread. The operation of public transport bus services currently bases on buses traveling along *fixed* routes visiting *predefined* bus stops according to a *fixed* timetable. This is the product of efforts in public transportation planning where many factors of all stakeholders are taken into account in order to find an optimal solution covering the majority of the needs. Naturally, the main focus is on urban areas where a high demand for transportation exists. However, in less populated rural areas the service offerings are inferior and bus transportation should be adapted to their special requirements.

This work addresses the issue of fixed routes and bus stops by proposing a system in which the passengers put requests to the service to be picked up from any location to travel to any location within the service area. Subsequently, the service finds a bus in operation (from a list of buses), assigns the request to the bus and updates the route of the bus. This kind of service operation results in buses having dynamic routes.

We denote the system as *online full on-demand riding system*, in which requests are processed in real time during service time (*online* – as opposed to *offline* on-demand systems which collect requests before service, calculate a schedule which is fixed afterwards during the operation of the vehicle) and arbitrary coordinates are used for pick-up/drop-off (*full* – as opposed to *semi* on-demand systems which function based on fixed stations or locations).

Motivation

The aim of the project Mobility Broker[1] was to integrate multiple mobility services (e. g. bus, train, carsharing, bike-sharing) into one platform using standardized communication interfaces, i. e. IXSI[2]. The platform becomes the *one stop shop* for the mobility needs of its customers. This way customers have to interface with only one entity when planning an *intermodal* itinerary and can book one combined ticket instead of multiple mode-specific tickets (Beutel et al., 2014). It was, therefore, essential that the platform supports intermodal routing taking all integrated services into account (Beutel et al., 2016).

[1]https://mobility-broker.com/
[2]Interface for X-Sharing Information (Kluth et al., 2015)

Gökay, S., Heuvels, A., Rogner, R. and Krempels, K-H.
Implementation and Evaluation of an On-Demand Bus System.
DOI: 10.5220/0006300102170227
In *Proceedings of the 3rd International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS 2017)*, pages 217-227
ISBN: 978-989-758-242-4

During the analysis of status quo of mobility service offerings it was realized that low population density – *rural* – areas suffer from three shortcomings:

1. There are only a small number of mobility services being offered: Most service providers concentrate on urban regions, leaving rural areas out of their coverage plan (e. g. car-sharing).
2. Low service quality of existing services compared to urban regions: Considering the example of bus services, they mostly have less stations (i. e. stations are more distant from each other) and the buses are driving less frequently (i. e. increased waiting periods).
3. Rural areas can be sparsely connected to urban areas (or town centers).

These lead to bad customer experience which we wanted to challenge: Is it possible to provide an alternative, better service in rural areas and how can it be realized? How can we integrate this service into Mobility Broker, making the journeys between rural and urban areas seamless? Can the system be deployed in urban areas as well? In urban areas, can the system match or even surpass the quality of service of the traditional services (e. g. in peak hours)? And on a related note, would the system be only a supplement to existing offerings or could it replace existing services in urban areas?

The remainder of this paper is structured as follows: Section 2 lists related research and existing applications in the field of interest. Consequently, the taken iterative approach is explained in Section 3. Section 4 presents the tools and details of technical realization. It is followed by Section 5 that illustrates the evaluation methodology and results. Finally, Section 6 concludes the paper.

2 RELATED WORK

This section gives a general impression of related work done in the field of interest.

Demand Responsive Transport (DRT) is the umbrella term to describe and categorize the efforts to develop transportation services based on demand. In particular, the underlying problem is known as the *Dial-A-Ride Problem (DARP)*. DARP aims to find vehicle routes and schedules for k customers where each customer defines the pickup and delivery location, and the pickup or delivery time. DARP generalizes the Traveling Salesman Problem (TSP) and is therefore NP-hard (Gørtz, 2006). Due to the need in practical applications and its NP-hardness, many heuristic algorithms can be found in scientific literature. In (Savelsbergh and Sol, 1995) the authors present the general problem of pickup and delivery, its varying forms (DARP being one of them), the characteristics and solution approaches. An overview of models and algorithms can be found in (Cordeau and Laporte, 2007) and (Desaulniers et al., 2001). These works also unveil subcategories of DARP: Offline and online, single-vehicle or multi-vehicle, without time constraints or with time windows. In this context, a more focused overview, i. e. an overview of vehicle routing with time windows, is given in (El-Sherbeny, 2010).

An optimization method for improving or designing a traditional bus network for a small- or medium-sized town has been proposed in (Amoroso et al., 2010). The authors present a modeling framework with an iterative approach to improve existing bus lines in order to maximize an objective function. The objective function takes user demand as well as operation cost of the network into account. The work in (Crainic et al., 2010) deals with the design of a master schedule for a demand-adaptive transit system (DAS), which is a hybrid between traditional bus service and an on-demand system usually offered in low demand transportation areas. The proposal employs a set of *mandatory* stops with corresponding time windows and *optional* stops which are inserted by passengers. An on-demand bus system which highly influenced the system proposed by this work is presented in (Tsubouchi et al., 2010). However, unlike this work, the presented system uses predefined bus stops. The system leverages cloud computing to lower operational costs and computation time. A simulation and field tests in multiple cities yield promising results and the authors conclude that, with their heuristic, the calculation time does not increase exponentially. In (Martinez et al., 2006) the authors propose an online on-demand shuttle system for the University of Maryland College Park, that continues to use existing bus stations. For evaluation, the authors compared the results of simulations of their and existing shuttle systems. It was concluded that their approach performs slightly better than the existing system. Nevertheless, it lacks the usage of time windows, so that arrival times cannot be predicted or guaranteed. The authors of (Mukai and Watanabe, 2005) test different on-demand transportation services, which they categorize in semi- and full-demand systems. The feasibility and realization of a DRT service for a low-demand density area is examined in (Li et al., 2007). A model to investigate the financial viability of a DRT system is presented in (Ronald et al., 2013).

3 APPROACH

We borrow the general idea from the work in (Tsubouchi et al., 2010) and adapt it to our needs. At the core, given a new request and a dynamic riding system with a fleet of multi-passenger vehicles, it addresses two problems:

- How to choose a fitting vehicle for the new request?
- Can the request be assigned to the chosen vehicle?: Does insertion of the new request into the schedule of the chosen vehicle violate the constraints of already accepted requests? Can this be overcome?

We assume that the system provides customer interfaces (web-based or mobile application) which can be used by customers to send requests. We also assume that the buses are equipped with GPS devices and mobile computers supporting wireless Internet access for bidirectional communication with the central system. Thus, GPS devices can report current location data of the bus to the central system. The central system can notify the bus about an updated schedule/route, which is shown to the driver in a screen on a map. Current state of the system is reached in two phases:

1. Online, semi on-demand: Usage of existing stations and routing between them
2. Online, full on-demand: Arbitrary locations for pick-up and drop-off and road based routing

3.1 Phase 1 - Online Semi On-Demand

As a starting point, the system uses existing bus stations and connections provided by a static GTFS[3] feed. During initialization, we read GTFS data from the database and construct an in-memory graph of the service network for routing.

An overview of the working principle is depicted in Figure 1. Requests are specified by number of people, origin, destination and either pick-up or drop-off time. Based on the origin and destination, the *Request Processor* calculates the shortest path and uses its duration to set the missing time parameter (pick-up or drop-off). Then, we relax both time constraints by applying relative and absolute slack times which at the end give us time windows for both pick-up and drop-off. The concept of time window (Figure 2) is essential to increase the overall flexibility of the system. Without it, the schedules would be too rigid since the

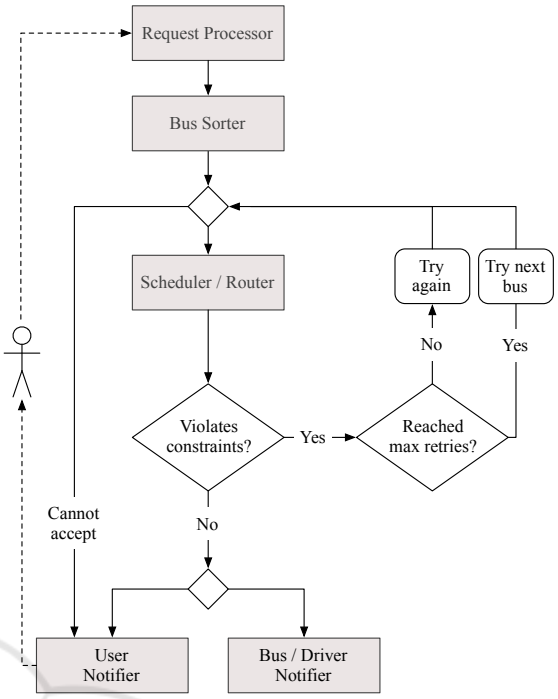

Figure 1: Working principle overview.

accepted requests would not tolerate to be late or early as a new request is inserted into the schedule. We actually model a request as containing two events (pick-up or drop-off), each of which contains a location, an actual time and time window.

Afterwards, the *Bus Sorter* takes over to sort the buses by their fitness for the new request. *Bus Sorter* employs a heuristic with a decision variable. The decision variable is calculated for the new request and each bus. Then, buses are sorted in descending order of the decision variables. The original idea in (Tsubouchi et al., 2010) describes only one heuristic: Favoring of the bus that already travels along a direction to the new request. Additional to that, we implement multiple strategies for comparison:

1. Prefer idle buses: Distribute the requests in a way, that the customers are transported faster at the cost of buses being longer in operation. This heuristic takes sides with the customer (Generally, the customer and bus service operator have conflicting objectives: The customer wants to travel faster and be earlier at the destination, whereas the bus service operator wants less buses in service in order to minimize operational costs.).
2. Avoid idling of buses: Prefer buses that are already in motion/operation, even though it might prolong the travel time of the customer. This heuristic takes sides with the operator.
3. Distance: This heuristic considers how far the

[3]General Transit Feed Specification. https://developers.google.com/transit/gtfs/

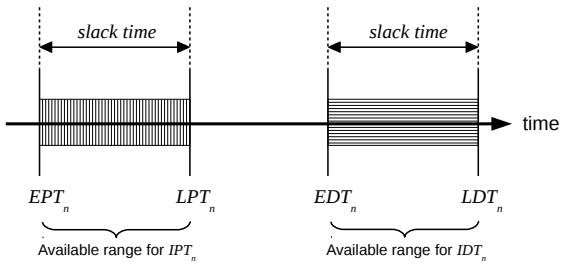

Figure 2: Time windows (Tsubouchi et al., 2010). EPT (LPT): Earliest (latest) pick-up time, EDT (LDT): Earliest (latest) drop-off time, IPT: Informed pick-up time, IDT: Informed drop-off time.

bus would be from the pick-up stop at the desired pick-up time of the new customer, and analogously from the drop-off stop.

4. Random: Assign random decision variables for the sake of comparison how the bus sorting strategy influences the results.

The results are presented in Section 5. The *Scheduler* iterates over the sorted bus list and executes for each (bus, new request) pair the steps summarized as follows:

1. Collect all events of the bus.
2. Add 2 events of the new request into collection.
3. Sort the collection of events by actual times.
4. Since the order of events might have changed, compute new paths between events and update actual times.
5. Check feasibility of events: If there are violations, adjust infeasible events. Afterwards, if max retries is not reached, go to Step 3 and try again.
6. Check bus capacity constraint: If there is no violation, calculate new schedule and return it. Otherwise, if max retries is not reached, go to Step 3 and try again.

If a schedule is returned, we stop iterating over the remaining buses and conclude that the fitting bus is found. Overall, the scheduling algorithm aligns with the one described in (Tsubouchi et al., 2010), but differs in that, after sorting the events by actual times, we calculate the paths between successive events and, if necessary, update actual times (Step 4). This idea can be illustrated with an example (Figure 3).

Until the insertion of PRE_5, the route contains a path leg $PRE_1 \rightarrow PRE_2$. Since PRE_5 should be fulfilled after PRE_1 but before PRE_2, the vehicle has to make a detour to pick up the passenger with request id=5. So there are two new path legs ($PRE_1 \rightarrow PRE_5$ and $PRE_5 \rightarrow PRE_2$) that need to be calculated and

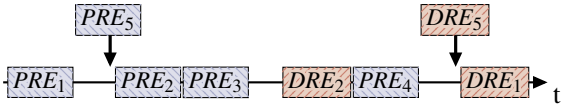

Figure 3: Example for insertion of the pick-up request event (PRE_5) and drop-off request event (DRE_5) for the request id=5.

used. This can cause a delay for arrival at PRE_2. If this is the case, then the actual time of PRE_2 is set to the newly calculated time (actual time of PRE_5 plus the duration of $PRE_5 \rightarrow PRE_2$). But now there might be a conflict between PRE_2 and PRE_3, or between PRE_3 and its successor, and so on. As it turns out, the insertion of PRE_5 can start a chain reaction of shifting of subsequent events until reaching an event whose actual time is not affected (i. e. there is a huge time gap to its predecessor). The insertion of DRE_5 can cause the same effect and chain reaction.

The potential violation of constraints (e. g. actual pickup time is later than latest pickup time) are discovered by applying the feasibility rules and are solved subsequently (e. g. by setting actual pickup time to latest pickup time) as described in (Tsubouchi et al., 2010). If there are no violations for a bus, we compute the new schedule and notify the user and the bus driver. If we iterated over all buses but could not find one without violations, we conclude that this request cannot be accepted.

3.2 Phase 2 - Online Full On-demand

In the second phase, the capabilities of the system are extended by switching from using a static station/connection network to dynamic locations (geocoordinates) for pick-up and drop-off. To achieve this, we need knowledge about the map and road network, more specifically a routing solution based on road network.

Moreover, inspired by (Martinez et al., 2006) we implement a new bus sorting strategy using *fuzzy c-means clustering*. Regular *k-means*, given a distance function d and a predefined number of k clusters, assigns each of n points to one of the k clusters. Instead of assigning a point to one cluster, *fuzzy c-means* calculates the membership/likelihood degree of a point for every cluster. Since we do not choose only one bus but rather try out all buses, membership/likelihood degrees of buses as fitness values allow us to stay within our framework. This way we can sort clusters (buses) for a point (route request). Since the already implemented bus sorting strategies were quite primitive, with this, we wanted to evaluate a more sophisticated approach whether it would deliver better estimation for request-bus mapping.

These extensions do not require any modification in Figure 1, only replacing some of the lower level components with new implementations. The main component of the system, the scheduler, can be used as it is. This is the case because it uses the *time* of the events as the primary constraint. The *space* constraint (whether we use fixed stations or arbitrary locations) is only relevant for the underlying routing engine that finds a path and its duration, which we use to calculate/set/adjust the times of pick-up and drop-off events.

4 IMPLEMENTATION

The system is implemented as a standalone Java application interacting with PostgreSQL for persistence. The implementation details specific to evaluation are described in Section 5.

In the first phase, in which stations are used, the GTFS data is imported into PostgreSQL once during the development using the GTFS SQL Importer[4]. During the initialization, the application fetches the stations, the arrival/departure times of bus trips at stations from the database. With this information we build an undirected graph using JGraphT[5]. In the graph, the stations become the nodes and we construct an edge between two subsequent stations of the same bus trip. As the edge weight we use the calculated travel time from a station to the next station in the same trip. Since we are dependent on the routing between nodes, JGraphT offers a reliable and fast A* implementation which we utilize to calculate shortest paths in the graph. The *Bus Sorter* component is implemented using the strategy design pattern to make the exchange of various heuristics easier. The *Scheduler* implementation is the direct translation of the approach presented in Section 3.1 into Java code.

In the second phase, in order to support routing between arbitrary locations, we opt for using OTP[6]. OTP is a Java library with an embedded server for efficient pathfinding through multi-modal transportation networks. Transportation networks are represented by OSM[7] graph data and optionally extended by GTFS feeds for transit routes. OTP provides a REST API[8] for programmatic integration of third-party applications. The bus system acts as a REST client, and sends routing requests to an OTP instance running locally. The REST requests specify the transportation mode as *car* to trigger pathfinding using only the road network. *Fuzzy c-means* implementation bases on the one of Apache Commons Math library[9] with slight modifications for our use case.

[4]https://github.com/cbick/gtfs_SQL_importer

[5]JGraphT is a graph library that provides graph-theory data models and algorithms. http://jgrapht.org/

[6]OpenTripPlanner. http://www.opentripplanner.org/

[7]OpenStreetMap. http://www.openstreetmap.org/

[8]Representational State Transfer Application Programming Interface

5 EVALUATION

In this section, we first overview our methodology and approach how to test the system, as well as the components that complement the core implementation to render such an approach possible. Afterwards, we present the evaluation results.

5.1 Phase 1 - Online Semi On-demand

The first version of the system, which uses existing bus stations and connections, is evaluated with a simulation module, that

1. updates the position of the bus as time passes, and
2. generates new requests which are processed by the system.

It was necessary to simulate the operation of buses within a short period of time while the service operation covers a wider range of time. To realize this we utilize a ticking clock, that is used throughout the system, to add an offset to the current time when necessary. Regarding the first point, it is checked whether the bus has arrived at the approaching station in schedule. If it is the case, the related entities are updated (i. e. the schedule, paths, number of passengers). Regarding the second point, we randomly generate requests within a bounding box on the map (which corresponds to service coverage derived from the GTFS data). In order to achieve a more realistic simulation, we assign to each station a *local density*, that is a function of its distance to all other stations. It is formulated in Equation (1), where V is the set of stations in the network and $d(u,v)$ is the direct distance between $u, v \in V$.

$$density(v) = \sum_{u \in V, u \neq v} \frac{1}{(d(u,v)+1)^2} \qquad (1)$$

The local density is used in Equation (2) to calculate the probability ω of a station to be used as origin or destination in request generation. With this we prevent generating excessive number of requests with stations that in reality are used less frequently.

[9]http://commons.apache.org/proper/commons-math/userguide/ml.html

$$\omega(v) = \frac{density(v)}{\sum_{u \in V} density(u)} \quad (2)$$

The objective was to run our *dynamic* system and simulate a conventional bus service (*static* system), send them requests, monitor their status and compare the results on the basis of following criteria:

Speed denotes how fast the customers arrive at their destinations. Its value is within the interval $[0,1] \subset \mathbb{R}$. The value of 1 represents the fastest possible journey from origin to destination, whereas a value of 0 represents an infinite journey (in theory). The higher the value, the better.

Capacity Utilization: denotes the average percentage of occupied seats during the service time of a bus. A lower value implies empty trips. The higher the value, the better.

Success Rate of Requests: expresses the percentage of accepted requests. The slack time parameter has a huge effect on the success rate, since a lower slack time infers less flexible time windows, which influences the acceptance of new requests negatively.

Bus Activity: denotes the average percentage of time during which the buses are in service (motion). Its value is within the interval $[0,1] \subset \mathbb{R}$. A high value could imply acceptance of a huge number of requests, but also an inefficient utilization of buses (i. e. assignment of a request to a suboptimal bus). Therefore it should only be interpreted within context of other criteria. Moreover, this measure only applies to the dynamic system, since in conventional services the buses are in motion regardless of the fact that there is demand or not.

Simulation Configuration. The region used for the simulation was the south-east rural area of Aachen, Germany with 52 stations in total (Figure 4). The simulation of the static system is configured in a way that the bus lines as defined in GTFS are used, but the frequency of a line has to be set. The default bus line frequency is set to 30 minutes, which corresponds roughly to the reality in the rural area. In the dynamic system, the number of buses can be adjusted. Moreover, the relation between the time the request is sent to the system and the time for requested departure is important. Journey requests *at short notice* do not let the schedule to be adjusted in a timely fashion. Therefore, the requests must be sent at least 30 minutes and at most 90 minutes before. In all instances, the simulation time period is 72 hours. Within this period approx. 6480 requests are processed (2160 requests

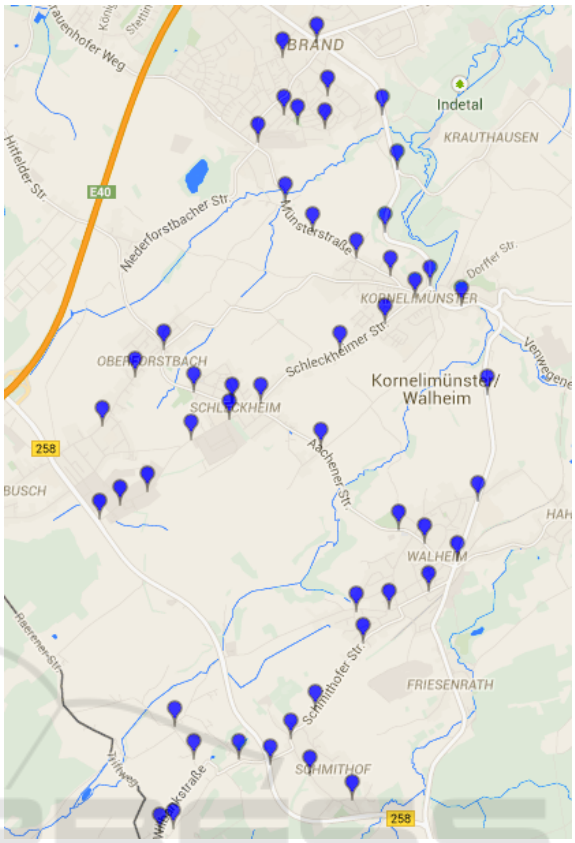

Figure 4: Stations for GTFS in a rural area (south-east of Aachen, Germany).

per day, 1 request every 40 seconds). Unless stated otherwise, the absolute slack time is set to 15 minutes and the relative slack time to 25% of the travel time. It is also important that both static and dynamic systems use the same bus capacity which is set to 20.

Results. Figure 5 illustrates the effect of different frequencies. This test is conducted to ensure that the simulation of the static system works as intended. Success rate decreases as the frequency decreases (i. e. as the interval between two bus dispatches of the same bus line increases), which is expected since the customers are not willing to wait for a longer time. Figure 6 depicts the results of using different strategies. Interestingly, the results are more or less the same regardless of the chosen strategy. We argue that this is due to the chosen, rather small region. On larger regions this should affect the results profoundly. Figure 7 illustrates the effect of the chosen window size. With increasing time window the speed decreases, which means it takes longer for the customers to arrive at the destination, and therefore the bus capacity utilization increases as well. Figure 8 aligns with the educated guess "the more buses the higher the success rate" but also hints at a satura-

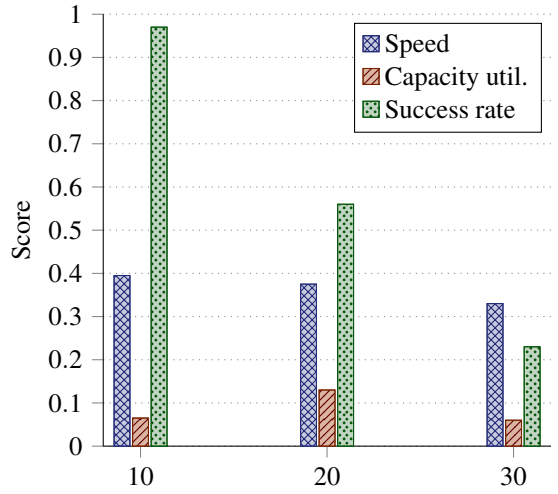

Figure 5: Static system - Comparison of different frequencies (in min).

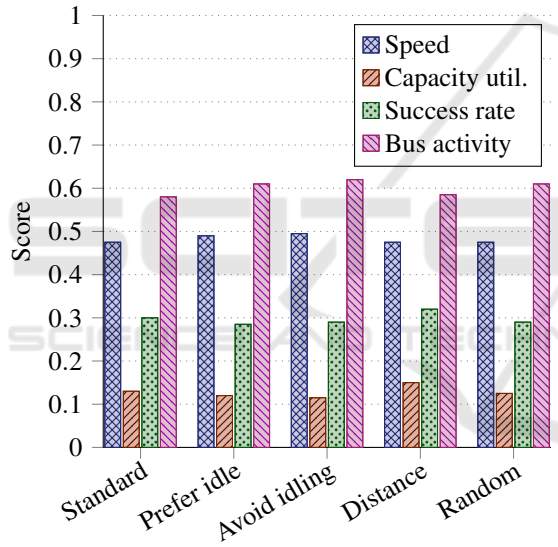

Figure 6: Dynamic system - Comparison of bus sorting strategies using 6 buses.

tion point for the maximum number of buses needed which would be interesting for the operator. Table 1 provides a summary of the comparison of the static/dynamic systems: The static system with 12 buses could only fulfill 24% of the requests, whereas the dynamic counterpart with the same number of buses can accept 59%. The trips take less time with a dynamic system as well (67%). Since more requests are accepted, the dynamic system has a higher capacity utilization (15% compared to 4.5%). In the dynamic system, the buses are in motion/service 55% of the time. During the remaining time they are waiting idle to service requests. Bus activity calculation of the static system is omitted, since they are in service all

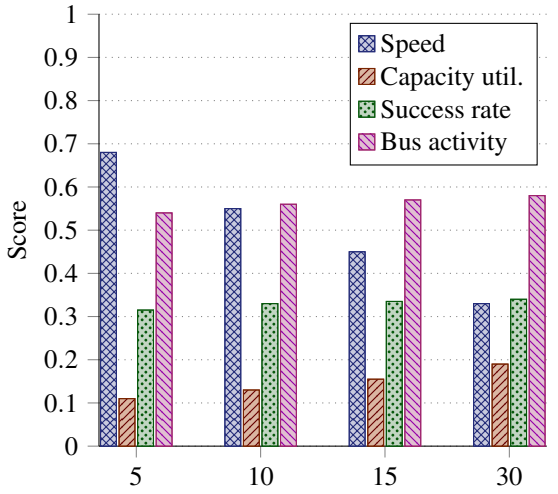

Figure 7: Dynamic system - Success rate with increasing time window (in min).

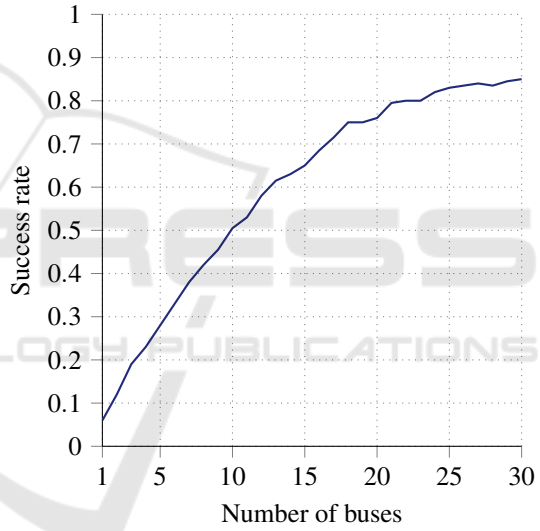

Figure 8: Dynamic system - Success rate with increasing number of buses.

the time during the service operation, which could be interpreted as 100% bus activity.

5.2 Phase 2 - Online Full On-demand

The second version of the system, now operating with arbitrary pick-up and drop-off geocoordinates, is evaluated by performing *quality tests* that compare the request processing of the dynamic and static system with their results. But first, we want to demonstrate that bus sorting with fuzzy c-means clustering can offer better estimates than prior heuristics.

Bus sorting with fuzzy c-means. In order to evaluate the clustering strategy, we constructed a test bed with 3 requests (Figure 9). The number of buses was set to

Table 1: Static/dynamic comparison overview.

	Static (12 buses)	Dynamic (6 buses)	Dynamic (12 buses)
Success rate	24%	33%	59%
Relative trip duration	100%	70%	67%
Capacity utilization	4,5%	15%	15%
Bus activity	–	59%	55%

2, so that the choice of each sorting strategy is easily visible. Request 1 and 2, color-coded with orange and green paths respectively, were sent first to the system, specifying the same pick-up time. Figure 10 illustrates the resulting bus route of the one bus, to which all 3 requests were assigned, using the distance strategy (not tested with others since Figure 6 indicated similar results). This is a viable but sub-optimal bus route. It causes delays for customers and yields longer routes for the bus in use. Figure 11 shows the result using the clustering approach. It employs both buses (color-coded as green and purple), but the routes of the buses are shorter and more compact.

Quality Tests

As with the first phase of development, we construct an environment where we run two simulations (dynamic and static), generate a request set, let both systems process the same requests and interpret the outcome.

Simulation Configuration. The simulation is modeled to mirror the public bus service in Ulm, Germany operated by Stadtwerke Ulm/Neu-Ulm (SWU). Ulm is a small to medium-sized city with a comprehensible transit service. SWU puts its GTFS feed online which we use for our static system simulation[10]. We import the OSM graph of Ulm and the GTFS feed of SWU into OTP. In REST requests for the dynamic system the *mode* parameter is set to *car* to use only the OSM graph, and for the static counterpart it is set to *bus* to make OTP use the GTFS data. According to the SWU website, it offers a bus service including 63 buses (with an average capacity of 142), distributed over 12 bus routes[11]. The dynamic system is configured in a way to match the static system. The absolute slack time of the dynamic system was set to 20 minutes, which is the average bus frequency calculated from all bus schedules[12]. After having looked at the

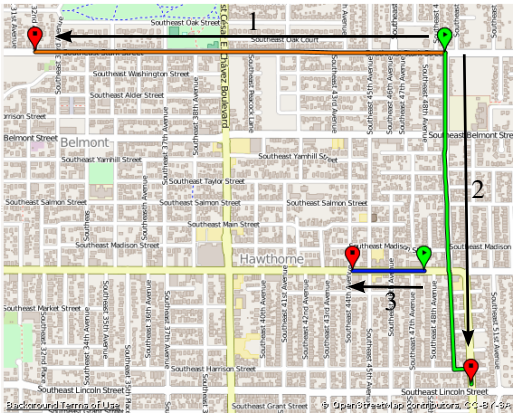

Figure 9: Requests.

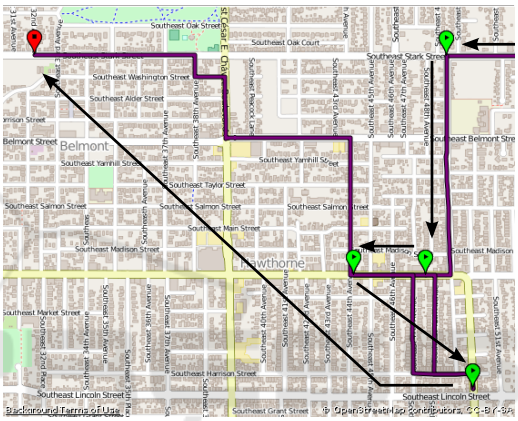

Figure 10: Using the distance bus sorting strategy the three requests are assigned to the same bus.

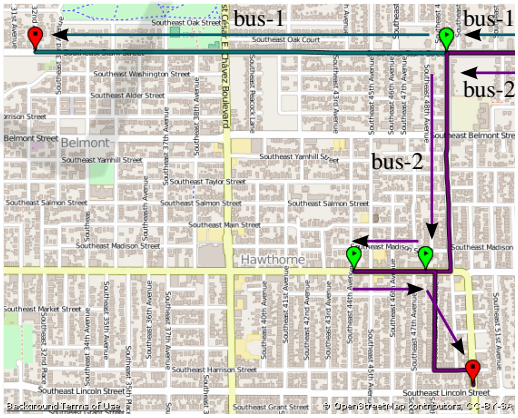

Figure 11: With fuzzy c-means bus sorting strategy the three requests are assigned to two buses.

[10]https://www.swu.de/privatkunden/swu-nahverkehr/gtfs-daten.html

[11]https://www.swu.de/privatkunden/swu-nahverkehr/fuhrpark/busse.html

[12]https://www.swu.de/privatkunden/swu-nahverkehr/fahrplan-liniennetz/linienfahrplaene.html

GTFS data, we decided to generate requests that fall in two time periods: From 16:15 to 17:50 and from 21:15 to 22:50. During the first time period the static bus system makes use of all available buses. Within the second time period only 5 of 11 bus lines are active in the static bus system. 5700 requests were generated for both time periods, which approximates one request per second. For each time period, the start and end locations of generated requests fall in one of the following categories:

1. At bus stations of the same route in static system
2. At bus stations from different routes in static system, which means that the customer of the static system has to switch buses during her journey. This causes longer travel times and potentially forces customers to walk a part of their route.
3. Arbitrary locations within the operation boundaries independent of any bus route.

This setup presumably should yield results that are in favor of the static system for the first category, and the dynamic system for the third.

Results. For the sake of brevity we only present the results of the time period from 16:15 to 17:50. Time period from 21:15 to 22:50 yielded similar results. Figure 12 shows the average amount of time the customers save using the dynamic system instead of the static system. In this context, the total travel time represents the time starting from the desired pick-up time to the actual drop-off time. This time span thus includes possible waiting and walking times of customers, too. It reveals the tendency of having least saved time for requests with locations at the same route, and most saved time for arbitrary request locations. This matches the expectations since the granted advantage for the static system decreases with arbitrary locations. Moreover, in all cases the amount of saved time is positive, which shows that customers have shorter travel times even in scenarios that should benefit the static bus system. Figure 13 illustrates the average walking times in the static system since it requires customers to walk to a pick-up station, or between stations (if there are switches), or from a drop-off station to their actual destination. In the dynamic system, there is no such condition. The fact, that the customers have to walk more for arbitrary locations, correlates with the expectations. Figure 14 compares the number of rejections in both systems. The low number of rejections in the dynamic system can be explained by having this many buses (63). It almost always finds a fitting bus. While the dynamic system can by nature reject requests, the rejection for the static bus system had to be simulated. Therefore, it was assumed that customers would not wait longer or

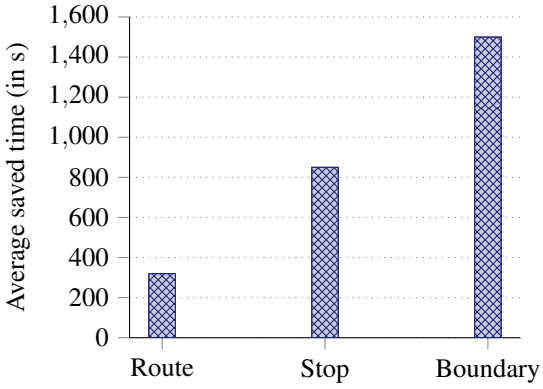

Figure 12: Average saved travel time.

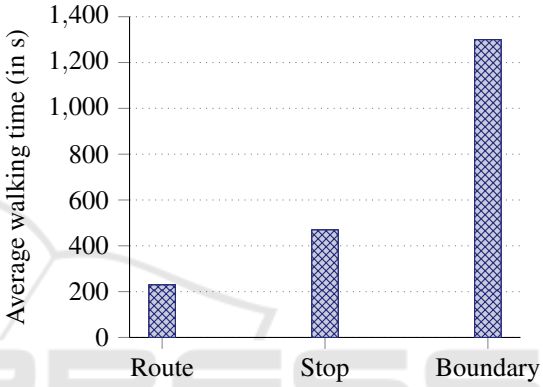

Figure 13: Average walking time for static system.

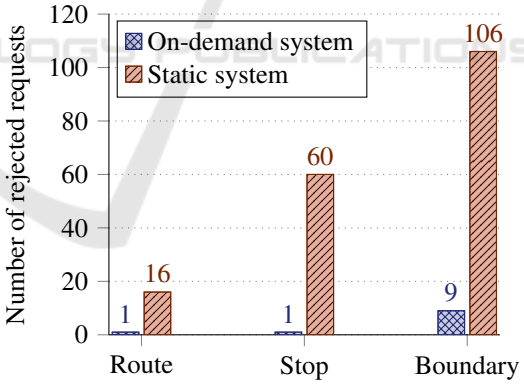

Figure 14: Comparison of rejections.

arrive later than the slack time of the dynamic system (20 minutes – average bus frequency). After having evaluated both systems using the same configuration, we wanted to see how much we can scale down the dynamic system while it still performs at least as good as the static one. Figure 15 depicts the relation between the number of buses and the number of rejections. So, while the static system has 106 rejections with 63 buses, the dynamic system can match the same success rate with 16 buses (107 rejections). This is valuable for the service provider to cut down

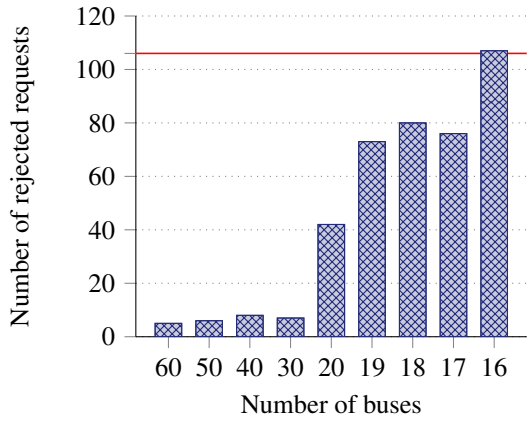

Figure 15: Optimal number of buses. The horizontal line represents 106 rejections of the static system with 63 buses.

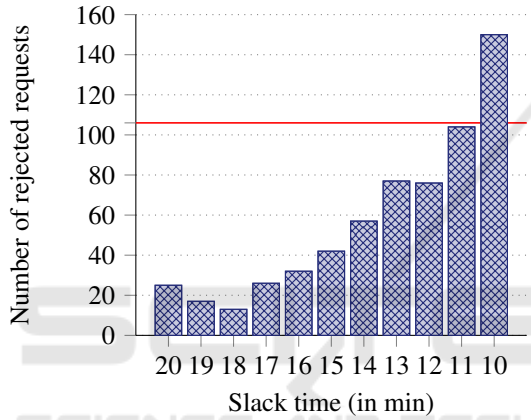

Figure 16: Optimal slack time for 30 buses. The horizontal line represents 106 rejections of the static system with 63 buses.

operational costs. Next, we wanted to see how much we can reduce the slack time. A broader slack time provides the needed flexibility to our dynamic system, but is inconvenient for the customer. Figure 16 shows that we can reduce the slack time to 11 minutes with 30 buses and match the success rate of the static system (63 buses with 20 minutes average bus frequency).

6 CONCLUSION

In this work, we implemented and evaluated an online full on-demand bus system in two phases. We started with a preliminary version that routes freely from station to station, but visits only the stations where customers are to be picked up or dropped off. This allowed us to evaluate the feasibility of an on-demand system before advancing the approach. We evaluated it in comparison with a traditional bus service simulation using a rural area (south-east of Aachen, Germany), where the quality of bus services usually suffer from low density of bus stations and low bus frequencies. The comparison was done on the basis of efficiency criteria such as speed, capacity utilization, success rate and bus activity. The evaluation showed that the dynamic system can deliver better results with the same (or less) number of buses as in the static system.

In the second phase, we extended the system to route between arbitrary locations and replaced the bus sorting strategy with a more sophisticated one. The tests showed that the new bus sorting approach yielded a better order of buses for the to-be-processed request. Furthermore, comparisons were performed with the static counterpart by running simulations within an urban region (as opposed to a rural area of the first phase), namely the transit service area of Ulm, Germany. These demonstrated clear advantages in terms of saved travel time, non-existing walking time and number of accepted requests when using the same configuration as in the static system. We also determined that the dynamic system requires less resources in order to provide the same quality of service of the static system.

Future Work

This work presented the basic approach, implementation and evaluation. It serves as a basis for further experimentation and improvements.

The evaluation methodology takes priority in the list of planned improvements. Even though the comparison of a new solution (on-demand bus system) to an existing solution (traditional bus service) is valuable, its realization is crucial for the end result. In particular, we need better insight and more statistical data about the bus service or the usage/travel patterns of customers for the evaluated area. *Simulating* a traditional static bus system or *generating* customer requests might not represent the reality. In order to improve request generation we plan to utilize *agent-based modeling*. We also would like to conduct field tests in order to evaluate the system, but also to validate the simulation results. Another aspect, the request-processing performance of the system, is not considered so far and is part of the future work.

Then there are features that we want to implement to enhance the capabilities of the system. In the current implementation, the bus picks up and drops off customers precisely at the given locations with no flexibility in this regard. But in a real setting, these locations might be very close to each other and still force the bus to stop for every single customer which

would not be desirable due to caused traffic congestion or bad user experience of customers within the bus. To overcome this, *assembly points* could be derived, where customers close to each other (e. g. within a radius) can meet before the bus picks them up, or multiple customers can be dropped off at the same location if their desired locations are close to each other. This brings us to the next feature of incorporating user preferences. The radius for assembly points can be set system-wide and identical for every customer, which would not be desirable. In the same light, the slack time for the requests is currently set system-wide and identical for every customer. Customers should be able to provide their own preferences, which are used in the request processing pipeline. This is also part of future work.

ACKNOWLEDGMENTS

This work was partially funded by German Federal Ministry of Economic Affairs and Energy (BMWi) for the project Mobility Broker (01ME12136) as well as for the project Digitalisierte Mobilität – Die Offene Mobilitätsplattform (DiMo-OMP).

REFERENCES

Amoroso, S., Migliore, M., Catalano, M., and Galatioto, F. (2010). A demand-based methodology for planning the bus network of a small or medium town. *European Transport / Trasporti Europei*, 44:41–56.

Beutel, M. C., Gökay, S., Kluth, W., Krempels, K.-H., Ohler, F., Samsel, C., Terwelp, C., and Wiederhold, M. (2016). Information Integration for Advanced Travel Information Systems. *Journal of Traffic and Transportation Engineering*, 4:177–185.

Beutel, M. C., Gökay, S., Kluth, W., Krempels, K.-H., Samsel, C., and Terwelp, C. (2014). Product oriented integration of heterogeneous mobility services. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1529–1534.

Cordeau, J.-F. and Laporte, G. (2007). The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 153(1):29–46.

Crainic, T. G., Errico, F., Malucelli, F., and Nonato, M. (2010). Designing the master schedule for demand-adaptive transit systems. *Annals of Operations Research*, 194(1):151–166.

Desaulniers, G., Desrosiers, J., Erdmann, A., Solomon, M. M., and Soumis, F. (2001). The Vehicle Routing Problem. In Toth, P. and Vigo, D., editors, *The Vehicle Routing Problem*, chapter VRP with Pickup and Delivery, pages 225–242. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

El-Sherbeny, N. A. (2010). Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods. *Journal of King Saud University - Science*, 22(3):123–131.

Gørtz, I. L. (2006). Hardness of preemptive finite capacity dial-a-ride. In in Computer Science, L. N., editor, *Proceedings of 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2006)*.

Kluth, W., Beutel, M. C., Gökay, S., Krempels, K.-H., Samsel, C., and Terwelp, C. (2015). IXSI - Interface for X-Sharing Information. In *11th International Conference on Web Information Systems and Technologies*.

Li, Y., Wang, J., Chen, J., and Cassidy, M. J. (2007). *Design of a demand-responsive transit system*. California PATH Program, Institute of Transportation Studies, University of California at Berkeley.

Martinez, M. V., Simari, G. I., Castillo, C. D., and Peer, N. J. (2006). A GPS-Based On-Demand Shuttle Bus System. Technical report, Department of Computer Science, University of Maryland College Park.

Mukai, N. and Watanabe, T. (2005). Dynamic Transport Services Using Flexible Positioning of Bus Stations. *Autonomous Decentralized Systems*, pages 259–266.

Ronald, N., Thompson, R., Haasz, J., and Winter, S. (2013). Determining the viability of a demand-responsive transport system under varying demand scenarios. In *Proceedings of the Sixth ACM SIGSPATIAL International Workshop on Computational Transportation Science*, IWCTS '13, pages 7:7–7:12, New York, NY, USA. ACM.

Savelsbergh, M. W. P. and Sol, M. (1995). The General Pickup and Delivery Problem. *Transportation Science*, 29:17–29.

Tsubouchi, K., Yamato, H., and Hiekata, K. (2010). Innovative on-demand bus system in Japan. *IET Intelligent Transport Systems*, 4(4):270–279.