Robotics and Autonomous Systems 170 (2023) 104559

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

On demand ride sharing: Scheduling of an autonomous bus fleet for last mile travel

Jörg Husemann [*], Simon Kunz, Karsten Berns

Robotics Research Lab, University of Kaiserslautern-Landau, Kaiserslautern, 67663, Germany

ARTICLE INFO

Keywords:
Mobility-on-demand
Bus fleet scheduling
Shared mobility

ABSTRACT

Autonomous buses are expected to expand the mobility-on-demand options in cities in the next 10 years. An essential aspect of ensuring optimal usage of fleets of autonomous buses is the task of scheduling. This paper presents a scheduling approach based on the construction of all possible trips used to formulate an optimization problem. While in the past most research is focused on the scheduling of taxi trips, there is an increasing interest in the research for the scheduling of last-mile travel options. A street network has been generated based on open street map data as a basis for the scheduling. All possible combinations of buses and requests are calculated, and for each of those trips, a close to optimal order of requests is created. These are used to formulate the optimization problem, calculating a close to optimal assignment of requests on the buses. The approach has considered constraints such as maximum waiting time, travel delay, and targets to exploit shared trips for higher efficiency. Experiments have been carried out in a simulated environment of a university campus area with fleets of up to 10 vehicles. By performing various trials with changing parameters, the influence of the constraints on waiting time and travel delay to the scheduling is determined. Depending on the setup, service rates above 90%, while trails with strict constraints show that the approach can handle short-term requests. Based on the results, a use case-specific composition of the autonomous bus fleet can be done.

1. Introduction

In recent years, the increase of mobility-on-demand services has been shaping the appearance of mobility within cities. A large number of options like scooter-, bike- and car-sharing concepts are available in most cities aiming to solve problems like air pollution and missing parking space while increasing the flexibility of people. The primary purpose of these mobility concepts is first-mile and last-mile transportation, in which the passenger is traveling, for instance, from a bus stop or train station to its target location. Due to the advances in the development of autonomous vehicles, autonomous bus shuttles are an option for these trips. These shuttles can complement the current services in a way that they can be used regardless of the physical limitations of the passengers. Another advantage comes from the fact that they are allowed to drive in traffic-calmed areas, such as pedestrian zones, where conventional traffic is not allowed.

One example for such a shuttle bus is the AutoBus (Fig. 1) currently developed at the TU Kaiserslautern. The hardware and software architecture is designed to drive in pedestrian zones. The central aspect is achieving a robust control architecture, interacting with the pedestrians. The interaction is realized by recognizing pedestrians' awareness regarding the bus and avoiding critical situations with them either by targeted driving maneuvers or by interacting with them via output interfaces, for instance, speakers and displays [1]. Furthermore, using safety certificated hardware and a validated software concept, it is aimed to pass the governmental validation processes for the bus [2].

For the success of a fleet of autonomous shuttles, a high degree of user acceptance is indispensable. This can be reached by achieving high reliability by minimizing waiting and driving times. Known mobility-on-demand offers are typically for single exclusive usage and follow a first come, first serve policy. Compared to these, the optimal use of a bus fleet has to include shared rides and a dynamic adaptation to new requests, leading to a more complex problem and requires a high degree of planning.

In this paper, a scheduling approach is presented, which calculates close to optimal routes for a fleet of autonomous buses. The approach is adapted from a concept for scheduling of shared taxi trips [3]. The database for the street network is based on freely available data of the OpenStreetMap (OSM) project [4], by which the approach is easily transferable to other areas. Experiments are done in an area of the campus of the TU Kaiserslautern and its immediate surroundings using

* Corresponding author.
E-mail address: joerg.husemann@cs.rptu.de (J. Husemann).

https://doi.org/10.1016/j.robot.2023.104559

Available online 12 October 2023

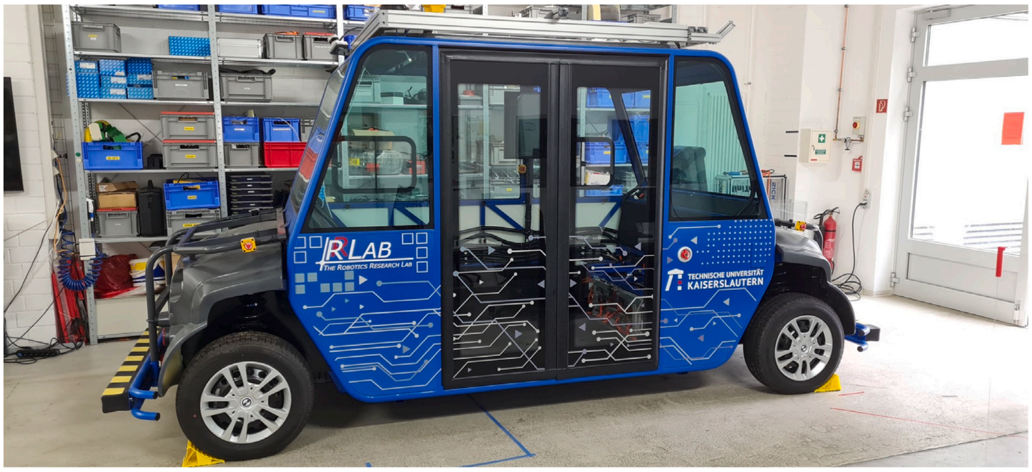

RLAB
THE ROBOTICS RESEARCH LAB
TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Fig. 1. Autonomous minibus (AutoBus) currently developed at TU Kaiserslautern.[1]

a bus fleet of up to 10 vehicles. In Experiments of several hours, delays and the rate of serviced requests are evaluated under varying degrees of utilization, testing the limits of the presented approach.

The paper is structured as follows. Section 2 introduces previous works in the field of scheduling for shared mobility concepts. Section 3 details the scheduling and assignment process. In Section 4 the evaluation of the concept is done, evaluating simulated experiments giving suggestions for the composition of the bus fleet. Finally, Section 5 summarizes the work showing the further way to the scheduling of a real fleet of autonomous buses.

2. Related work

The field of mobility-on-demand concepts, especially concerning autonomous vehicles, has already been investigated in several studies. Dia et al. [5] investigated the feasibility of agent-based simulation tools to evaluate the effects of shared autonomous vehicles, stating a drastic decrease of the traffic volume with the example of Melbourne traffic. Similar studies with varying focus were evidencing positive aspects of shared-mobility concepts in terms of pollution, congestion, and the availability of parking space [6] and were also predicting benefits from an economic point of view [7]. Besides simulation scenarios, which were used to show the advantages of mobility-on-demand concepts, there were also studies about real autonomous vehicles. Kim et al. [8] presented experiments of a campus mobility service using an autonomous taxi, further targeting to extend to a fleet of vehicles in the future.

For the general categorization of vehicle fleet management systems, Hyland et al. [9] presented a taxonomy to differentiate between various aspects of them. One part of the mentioned research was predefined aspects as the vehicle fleet's composition, defined timing constraints, and the availability of information about the vehicle states. Another aspect mentioned with a massive influence on the scheduling was the knowledge about the occurrence of passenger requests. The scheduling efficiency can be increased if knowledge about the occurrence of future requests is provided, for instance, by using stochastic models. Furthermore, various categorizations regarding ride-sharing, the road network, and reservation policies were set out.

So far, the application of fleet scheduling mostly refers to the organization of taxi trips. To reduce the number of required trips, the notion of shareability networks has been introduced [10]. The application of shareability networks includes transforming the spatio-temporal sharing problem into a graph-theoretical problem targeting to take advantage of well-known methods for the solution. *Shareability networks* are hypergraphs in which trips of different sizes are saved. Those trips are created by combining smaller trips while satisfying predefined constraints for delay and seating capacity. The concept was verified on a large set of recorded taxi trips, capable of scheduling trips smaller than size four in tractable time. Based on shareability networks, an anytime-optimal algorithm for scheduling was created, enabling the calculation of trips of arbitrary size [3]. In this approach, possible trips with a dedicated vehicle were incrementally constructed. After determining the optimal route for each trip, an integer linear programming problem was formulated, calculating an optimal assignment. Later the concept was extended to take stochastic information into account to bias future requests [11]. Based on experiments using a data set of actual taxi trips, the authors state an opportunity to strongly reduce the number of required vehicles while keeping waiting times and travel delays low.

Simonetto et al. [12] created a similar approach as presented before while making some assumptions to reduce the computational complexity while reaching similar results. In contrast to the previous approach, only one person can be assigned to each vehicle per step. The problem of multiple requests spawning simultaneously, which ideally can be handled by the same vehicle, should be avoided by reducing the sampling time to 10 s. Then, the problem is solved by formulating a linear assignment model. Furthermore, the complexity is reduced by only considering a limited number of vehicles for each request, consisting of a fixed number of free vehicles closest to the request and the same amount of already occupied but still available vehicles. In the experiments, roughly 10% of the available vehicles are considered for each request. In addition, the insertion costs for a request are calculated in a distributed manner within the corresponding vehicles, which spreads the computing load but also introduces a higher communication effort.

3. Bus fleet scheduling

The fleet scheduling aims to create a system with a high level of user acceptance. This acceptance should be achieved by creating a distribution of all requests to the bus fleet while creating optimal routes. In the scenario, a route is optimal by minimizing the sum of delays over all requests while satisfying given constraints. Considered constraints of the scheduling are the maximum waiting delay which defines the latest time the trip is allowed to start, and the maximum travel delay, which indicates the maximal additional driving time compared to the optimal travel time of each passenger in the bus.

An overview of the presented system is shown in Fig. 2. Passengers can send requests for a ride from a specified start to a target position. The requests of the passengers are sent to the request management and are collected there until the next scheduling cycle is triggered. The scheduling is creating plans for the bus fleet and sending them to the

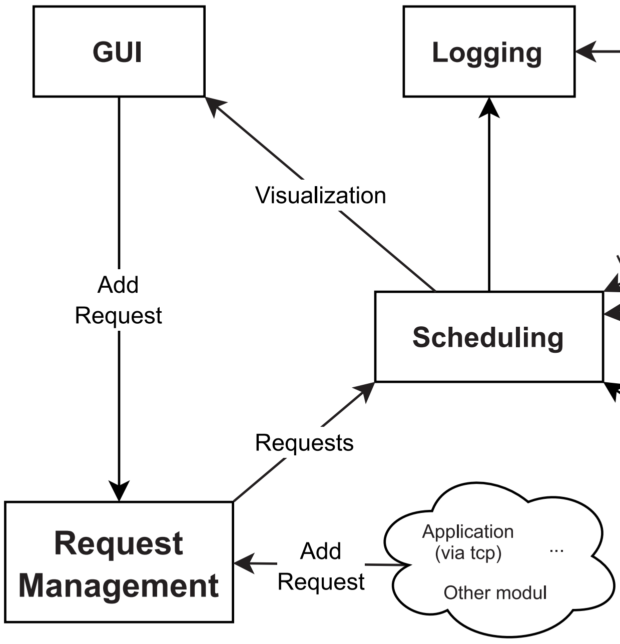


Fig. 2. Overview of the system components and their communication.

individual buses, which are then executing them. During execution, the buses send their current state information to the scheduling to be used in the following cycles.

3.1. *Representation of street network*

The precise representation of the road network is of high importance for a correct scheduling procedure. For a good representation of the street network, the used graph generated based on the free-to-use database of Openstreetmap[2] [4]. The information is saved in topological nodes, which are extracted and saved in a graph structure [13]. The created network generally consists of two types of nodes, geographical and topological. Topological nodes represent nodes with a degree different from two and, by that, either crossings or dead ends. The geographical nodes describe the road course more precisely between the topological node. In addition to the location, semantic information is saved in the nodes. Traversable roads for the bus can be determined based on the information regarding the street types. Fig. 3(a) shows the full graph generated from the OSM information, while Fig. 3(b) shows a cleaned-up version in which prohibited and unreachable regions are removed from the network. Common path-finding algorithms such as A* are implemented on the graph structure, which is later used in the scheduling process to calculate the fastest paths for the requests. It is possible to add and remove edges or change their properties during runtime. By that, it is possible to adjust the scheduling procedure based on environmental changes, for instance, blocked roads or large crowds. Since the calculation of the fastest paths between two points is based on the possible driving speed of the different street segments, the scheduling is automatically taking these changes into account. By generating the graph for the supported area from OSM it is simply possible to change the area for the scheduling to any arbitrary location covered by OSM.

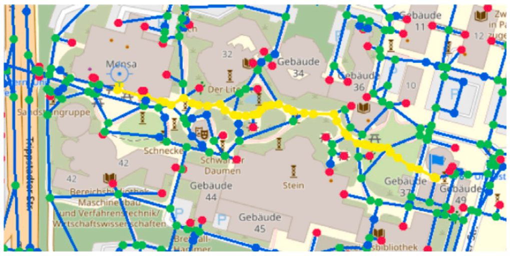
3.2. *Scheduling procedure*

Before starting the scheduling procedure, the newly received requests need to be processed. Since it is possible to send requests from arbitrary positions, while the bus has to stick to the edges of the street network, the requests are mapped to their closest point of the network. Additionally, to further reduce the complexity of the scheduling, the nodes are moved to the closest topological node of the network. This simplification is unproblematic in the chosen environment because of the small distances between topological nodes. In environments where adjacent topological nodes can have large distances, additional geographical nodes can be defined as possible starting nodes for requests. The whole scheduling procedure is overviewed in Fig. 4. Then the shortest connection between the start and target point for each request is calculated. This travel time is later used to calculate the requests' maximum travel delay. Before starting the scheduling, the request pool is cleaned from requests for which the maximum waiting time is exceeded. Then, if there are requests left in the pool, the scheduling procedure is started. Details on the assignment procedure are presented in the following sections. Generally, a close to optimal assignment of the requests in the pool to the busses is calculated. While doing so, the current position and the already assigned requests are considered. In every scheduling cycle, all requests in the request pool are either new requests or non-assigned ones from the previous scheduling cycles. Due to the fact that it is desired to give the customers feedback on the acceptance of their requests, it is not possible to give trips back from a bus to the request pool. It is, though, possible to reschedule the order of the tasks within each bus if new tasks are assigned, as long as the constraints of the already assigned tasks are met. After completing the assignment, buses may remain without any assigned trips. In this case, it is possible to create virtual requests, to position them in areas where requests are expected in the future. By doing so, the average waiting times for future requests can be reduced. To do this rebalancing in a meaningful way, it is required to have information on the typical distribution of requests. Finally, the requests and their execution order are sent to the vehicle and performed by them autonomously. The following sections give more insights into the graph generation and general scheduling procedure.

3.3. *Graph generation*

The graph generation is performed to calculate all possible combinations of buses and requests, which are later used to formulate the *integer linear programming* (ILP) problem. A request sent by a passenger

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


(a)

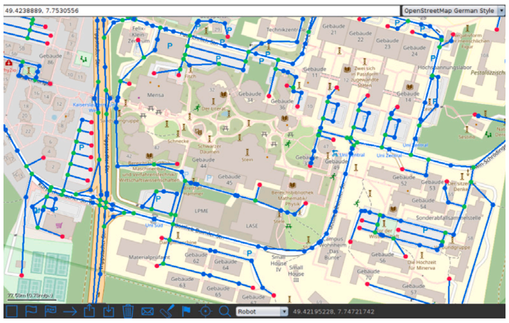
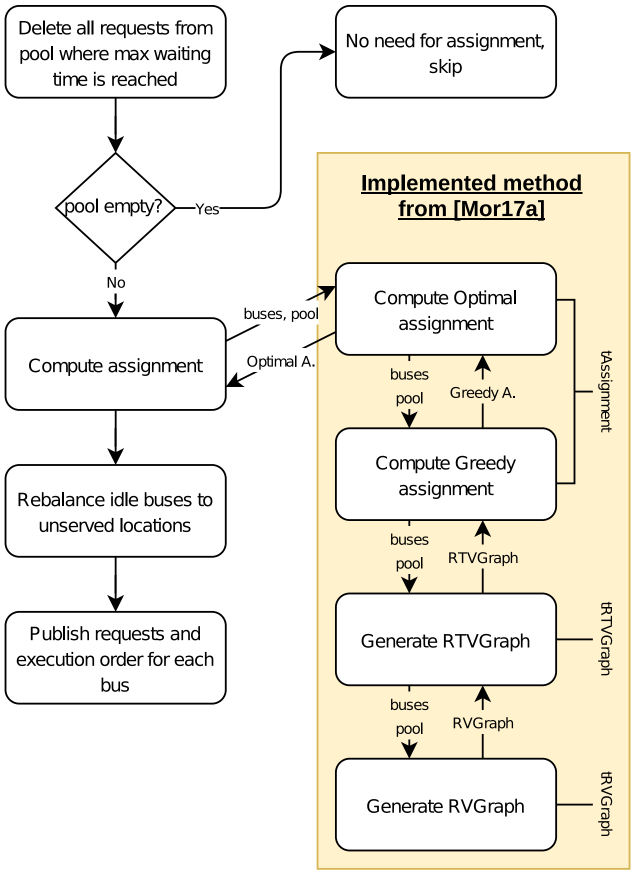

(b)

Fig. 3. Fig. 3(a) shows a zoomed-in part of the complete osm graph. Nodes are visualizing topological nodes (crossings (green), dead-ends (red), intermediate nodes of a street (blue)), and edges show the paths. The yellow path is the calculated shortest path between two nodes. Fig. 3(b) shows the reduced graph after removing edges that are not drivable by the bus. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



Delete all requests from pool where max waiting time is reached
No need for assignment, skip
pool empty?
Yes
No
Implemented method from [Mor17a]
Compute Optimal assignment
buses, pool
Optimal A.
Compute assignment
buses
pool
Greedy A.
tAssignment
Compute Greedy assignment
Rebalance idle buses to unserved locations
buses
pool
RTVGraph
Generate RTVGraph
tRTVGraph
Publish requests and execution order for each bus
buses
pool
RVGraph
Generate RVGraph
tRVGraph

Fig. 4. Overview of the scheduling procedure.

is defined by the desired pick-up and drop-off location and the time of the request. To ensure their uniqueness, the request management provides each request with an exclusive identifier. Furthermore, for simplicity, the pick-up and drop-off locations are mapped to the closed topological node in the open street map graph. Based on the parameters of the vehicle and the OSM graph, the travel duration of the request is calculated, and it is added to the request pool.

As introduced by Alonso-Mora et al. [3], the scheduling process is periodically triggered and is based on solving an *integer linear programming* problem using a greedy assignment as initial distribution to speed up the convergence. To formulate this problem, all possible trips for each bus are created using two graphs, the *Request-Vehicle-Graph* (RVGraph) and the *Request-Trip-Vehicle-Graph* (RTVGraph) as visualized in Fig. 5. The graphs include all unassigned requests and the buses. The RVGraph is created based on the current pool of unassigned requests and the states of the buses. In the graph, two requests r_i, r_j are connected if both requests can be served by a virtual bus starting at the pick-up location of one of the requests. The edge $e(r_i, r_j)$ is weighted with the sum of travel delays of the two requests. An edge $e(r_i, b_j)$ between a request r_j and a bus b_i exists if the bus b_i is able to satisfy r_j while keeping the constraints of this and all currently picked-up requests, representing all possible trip of size one. Using the RVGraph, the RTVGraph is calculated, which is used to calculate trips starting with trips with two requests. A trip of size k is defined as a set of requests $T = \{r_0, \dots, r_n\}$ with $|T| = k$. A subgraph of the RVGraph with k nodes is a trip of size k when it is complete. By that, the problem of finding all trips of a specific size corresponds to finding all cliques of the same specific size. Using the sub-feasibility lemma [14, p. 7], this problem can be solved incrementally. T_1 is created by adding all requests r servable by at least one bus and by that are part of an edge $e = (r, b)$. Next, the trips of size two up to the maximum seating capacity are calculated. For the calculation of T_k, all combinations of two trips from T_{k-1} with size k are calculated. For each valid combination, it is checked whether all subsets of the combined set of size $k-1$ are part of T_{k-1}. If this is the case, the combined set might form a feasible trip of size k. Then it is checked if there exists a valid route given the current bus states. If this route exists, the trip will be added to T_k. After calculating all T_k for all buses, they are merged into a trip set used for the assignment process.

3.4. Optimization of bus trips using simulated annealing

To formulate the *integer linear programming* problem, the costs for each trip have to be known. Optimal solutions for each trip are calculated using simulated annealing [15]. To deliver valid solutions for the bus application, some requirements have to be defined. Firstly, the bus location has always to be the first location of the trip. Secondly, the position of the pick-up of a request r_{i_pu} has to be before the drop-off position of the same request r_{i_do}. Thirdly, if the pick-up of r_{i_pu} is not in the list of requests, the passenger is already picked up, and by that, r_{i_do} can be placed at an arbitrary position in the request order. The last requirement is that constraints of maximum waiting time and the maximum travel delay are fulfilled.

The generator function considers the first three constraints. As an initial solution, a sorted list with the bus at the first position and each pick-up r_{i_pu} directly followed by its drop-off r_{i_do} is used. By that, the constraints 1–3 are initially met. The generator function creates new samples based on the current solution. This is done by randomly selecting one of the request locations of the trip and placing it at a new position as shown in Fig. 6. By ensuring to satisfy the second constraint, the possible range of the request is limited by the position of the corresponding request.

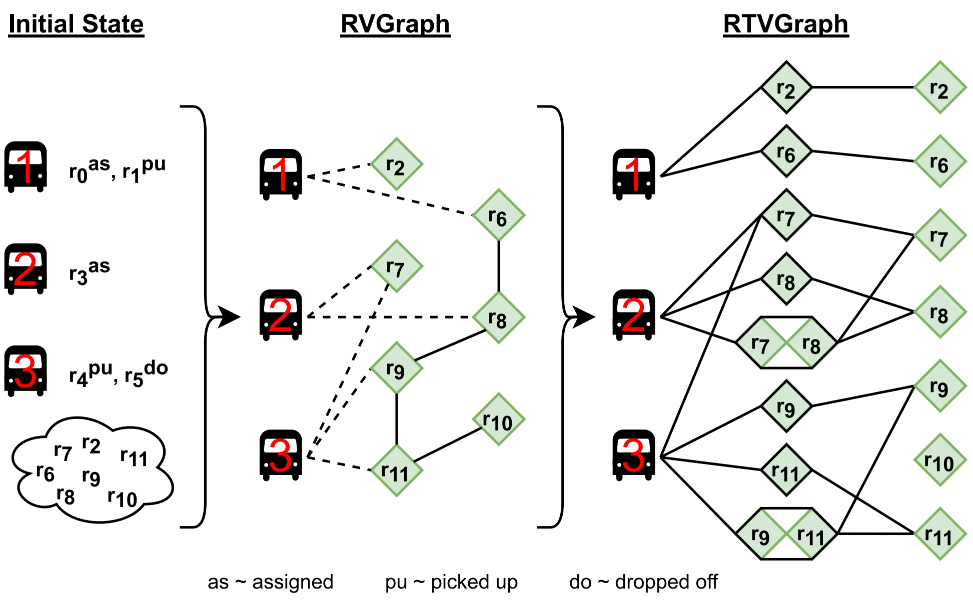


Fig. 5. Construction of the RTVGraph using the initial state and the RVGraph. RVGraph: dotted lines show which requests a bus can fulfill considering the already assigned tasks, and full lines show which requests can be combined independently of the bus. RTVGraph: framed requests are pick-up tasks, and non-framed are drop-off tasks.

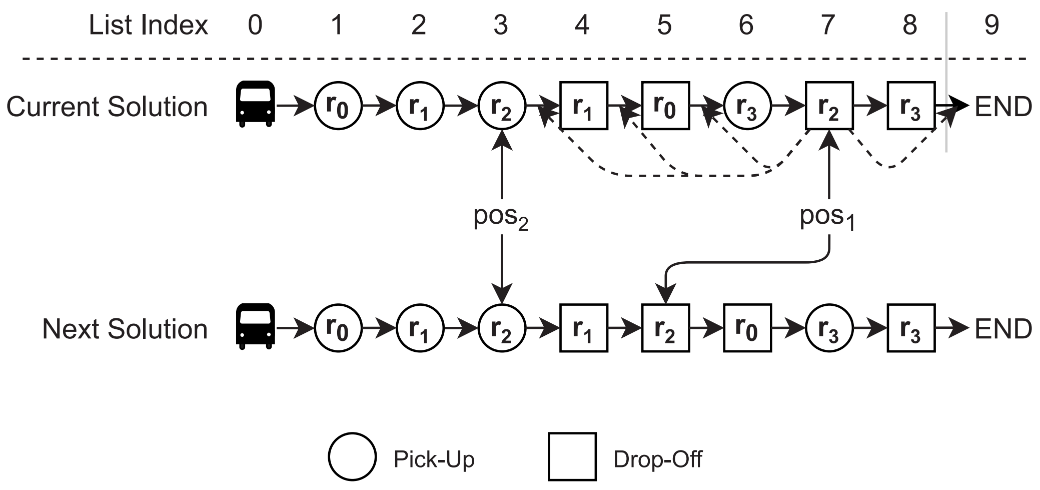


Fig. 6. Generation of the next valid solution based on the current solution. The drop-off of r_2 has been selected to change its position of the current solution, and pos_2 limits the possible positions for r_2 since it is a drop-off and cannot be placed in front of its corresponding pick-up.

Next, the energy function calculates the travel duration of the generated solution. The travel duration is calculated by adding the distances between adjacent requests, which are calculated by a path-finding algorithm on the open street map graph. In addition, the constraints for all requests are checked as visualized in Fig. 7. Using the distance between the requests, an expected timestamp for the completion of each request is calculated. Next, it is checked whether the constraints are met. For the pick-ups, the maximum waiting delay is checked $t_{req} - t_{pu} \leq d_{wt\_max}$ using the starting time of the request t_{req} and the expected time for finishing the request t_{pu}. For the drop-off tasks, compliance with the maximum travel delay is checked. This delay describes the maximal time the trip is allowed to exceed the expected driving time $t_{do} - t_{pu} - d_{travel\_time} \leq d_{td\_max}$. If the delays are exceeded, a high penalty is added to the travel delay calculated in the energy function. By that, a solution satisfying the constraints is always preferred.

3.5. Construction and solution of the ILP problem

Finally, the trips and their calculated delays are used to find an optimal assignment of requests to the buses. For this, the trips represented in the RTV graph are transferred to the decision variables in the ILP [3]. Linear programming is a method of mathematical optimization. It can be used for the maximization and minimization of linear object functions while satisfying defined constraints [16]. Integer linear programming, as used in this approach, is a special case of linear programming, including the restriction that all variables are integer values $x \in \mathbb{Z}^n$. Each edge e between a trip (T) and a bus (b) $e(T_i, b_j)$ is represented as a binary variable $\epsilon_{i,j}$ which equals 1 if the trip T_i is assigned to the vehicle v_j. The set $\mathcal{E}_{TV}$ is defined as the set of indexes for all edged $e(T_i, v_j)$.

Further, for each request $r \in \mathcal{R}$, a binary variable x_k is defined and set to 1 if the corresponding request r_k is not served. The cost function is defined in Eq. (1), where $c_{i,j}$ is the cost for the trip and c_{ko} is a penalty added if the trip is not assigned, which leads to avoiding unassigned trips if possible.

$$C(\mathcal{X}) := \sum_{i,j \in \mathcal{E}_{TV}} c_{i,j}\epsilon_{i,j} + \sum_{k \in \{0,\ldots,n\}} c_{ko}x_k \qquad (1)$$

Additionally, to get a valid solution, the following constraints are defined. First, each bus can only be assigned to at most one trip Eq. (2).

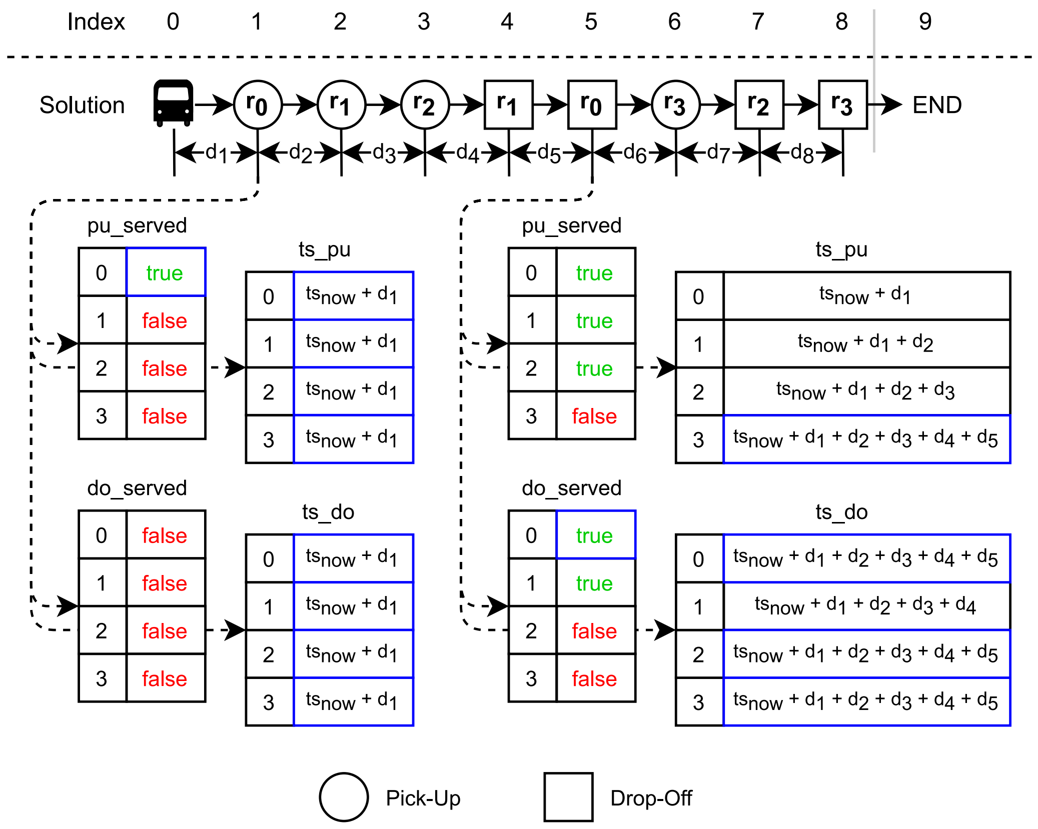


Fig. 7. Check of waiting time and travel time constraints during the optimization of each trip. The numbers in the first column represent the identifiers for the four tasks. The boolean in the second column shows if the action for the corresponding task is already executed (pick up (pu), drop off (do)). The next graph shows the stepwise calculation of the expected time for the task to be finish. Based on these timestamps the fulfillment of the constraints can be checked. The calculation is based on the current time and the expected time for the distance between two subtasks based on the current order of tasks.

$$\sum_{i\in\mathcal{I}_j^V} \epsilon_{i,j} \leq 1, \quad \forall v_j \in V \tag{2}$$

Secondly, each request can either be assigned to a bus ($X_k = 0$) or ignored ($X_k = 1$) if there is no way to schedule it while satisfying the constraints Eq. (3).

$$\sum_{i\in\mathcal{I}_k^R} \sum_{j\in\mathcal{I}_i^T} \epsilon_{i,j} + \mathcal{X}_k = 1, \quad \forall r_k \in \mathcal{R} \tag{3}$$

As an initial solution for the ILP, a greedy solution is generated by looping over the trips trying to assign the largest trips with the lowest costs to the vehicles. Then the ILP is solved using the open source lp_solve library.[3] While solving the ILP problem, the initial solution is replaced as soon as a better one is calculated. The process ends if an optimal solution is calculated or if there is a timeout. The timeout is triggered when the next scheduling cycle is started. Then the current best solution is used for the assignment. This solution is at least as good as the initial greedy solution by construction.

Finally, after the calculation of the solution, each bus is assigned its associated trip and can start to execute it. If an old trip is assigned, it is replaced with the new one. Then the new request execution order is checked for validity since, due to the calculation time, it is possible that some of the requests assigned are already executed. This is done by erasing finished tasks from the request execution order. The execution order is processed by calculating the route to the next request based on the open street map graph, which is then followed by the bus.

[3] https://CRAN.R-project.org/package=lpSolve

3.6. Interaction between scheduling and busses

The implementation of the concept is done using the robotic control framework *Finroc* [17,18]. *Finroc* is a modular framework designed for robot control systems. The data flow is established using ports that can be connected between various modules. A module is a component in the framework, which, among other things, contains a cyclically called Update function. This function contains the primary implementation of the module. It typically accesses information from its input ports, which provide data calculated in other modules. Ports are used for the data flow. They connect different modules to send data between them. Based on the dependencies of the data flow, the modules are scheduled, and their Update function is executed in each cycle. Furthermore, groups are used as a container for modules and other groups. In general, there is a large number of different components supporting the development of robotic applications. The structure of the implementation done for this work is shown in Fig. 8. It consists of two main groups, the *gScheduling* and *gAutonomousBusControl*. The first is executed on the server responsible for the scheduling, while one instance of the second group runs on each bus. *mScheduling* is the core part of the scheduling group. The module contains the scheduling procedure, which has been described in the previous chapters. The module *mRequestManagement* collects the requests and forwards them to the scheduling whenever a new cycle is started. In the following experiments, the requests are sent based on pre-computed datasets. In general, it is possible to send requests via a GUI or to receive requests via TCP messages, which can later be used to send them by mobile devices. The remote interfaces are used to send information between the buses and the scheduling. The ports visualized in *gScheduling* are vectors of ports, which contain one port for each bus of the fleet. *gAutonomousBusControl* is responsible for managing the requests sent by the scheduling group. It identifies the

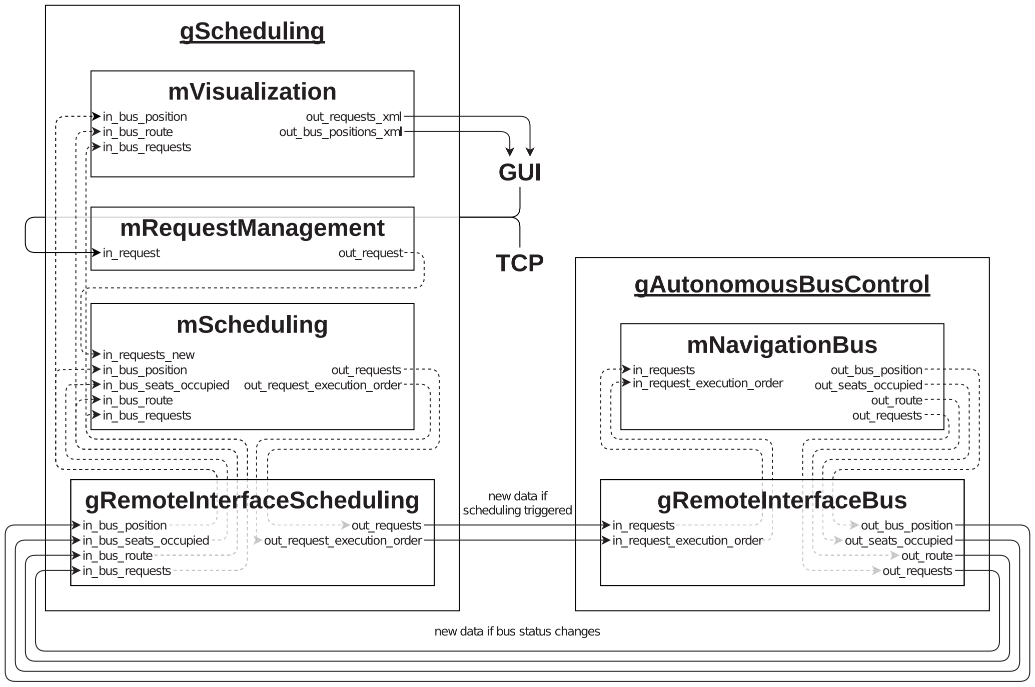


Fig. 8. Overview of the modules in groups created in Finroc for the scheduling part and the control of the bus. The boxes represent modules and groups. The arrows are connections established using ports.

next target node and forwards the information to the control of the real bus or the simulation used in this work.

4. Experiments

The experiments have been simulated on the street network of the TU Kaiserslautern and its surroundings. The buses are simulated on the OSM network traveling at a constant velocity. Kinematic constraints and delays are not considered, but non-traversable paths are removed from the OSM graph. The seating capacity of the buses is set to a maximum of six passengers per bus. Each bus is simulated self-sufficiently, receiving new trips from the scheduling algorithm and sending its status information using the same interface as a real bus. Owing to this, it is possible to replace one or all buses with real vehicles without the need for adjustments to the scheduling algorithm. Besides testing the general functionality of the approach, it was also targeted to analyze the behavior under heavy load. During the experiments, different parameters are varied to analyze their effect on the scheduling. To analyze the influence of the constraints maximum waiting time d_{wt_max} and maximum travel delay d_{td_max} used in the scheduling algorithm, they are set on values from three to seven minutes. Regarding the request, two parameters are set: their occurrence rate and their distribution. Each experiment uses a set of 500 requests, which are spawned based on the occurrence rate $r_{req} \in \{2, 4, 6\}$ $[requests/min]$. The cycle time used to pool new requests is 30 s. Since there are no real-world data to derive the locations of the request, data sets for the spawning of requests are created. One set is constructed using random positions on the campus area. For the second data set, five points of interest (POI) are defined, marking locations where a larger occurrence of requests is expected. Requests are placed based on a Gaussian distribution around these points. In addition to these two sets, combinations of both using different distributions between them are created.

A snapshot of an experiment is shown in Fig. 9. It shows a part of the available area and how the busses distribute on it. The picture shows a state after about 30 min, which is why the buses have now entirely spread over the map depending on the requests. The yellow edges in the graph show the paths the buses are taking. The triangles indicate the positions of pick-up tasks, and the stars show the places of the drop-offs. Grey icons are tasks that are finished, and finished requests are removed after starting the following scheduling procedure. First tests with at most three buses showed that the scheduling manages to reach serviced request rates close to 100%. After these results, a more extensive test setup with a fleet of ten buses is prepared. Based on the occurrence rate of the request, the experiments took between 1.5 and 4.5 h. During the tests, timestamps for assignment, pick-up, and drop-off are logged as well as the internal states of the buses.

During the scenarios, the size of the request pool is dependent on the chosen metrics. Nevertheless, the progression of the request pool more or less follows the same pattern. An example of the progression is shown in Fig. 10. In the beginning, there is a start-up phase in which the vehicles start to distribute on the map based on the first spawned requests and the rebalancing. During this phase, the request pool grows up to the maximum possible size, which results from the maximum waiting delay and the spawn rate of the requests. In the example in Fig. 10, a request is spawned every 15 s, and the maximum waiting delay is set to 300 s, which limits the request pool size to 20. This is the case since requests exceeding the maximum waiting time are removed from the request pool. In general, it can be observed that the pool size is closer to its limit in the experiments with high spawning rates. Furthermore, the experiments using a large proportion of random requests are closer to the maximum pool size since the chance to form larger trips is smaller. At the end of the scenarios, there is an end phase in which no new requests are spawned, and the remaining ones are either assigned to buses or removed due to violations of constraints.

The most important result is the percentage of serviced requests shown in Fig. 11. As expected, the service rate benefits from a lower occurrence rate of requests and more generous constraints. Especially in the experiments with the $r_{req} = 6$, the service rates are significantly less. This service rate is explainable by monitoring the utilization of the buses. Even though the buses are active the whole time, they can only serve some of the requests. Primarily in cases with strict constraints, the utilization of the buses is poor due to the reduced possibilities of shared rides, further decreasing the service rate. The scheduling failed due to many possible bus-trip combinations in the trial with $r_{req} = 6$ and high constraints. To bypass this problem, timeouts are required to stop the graph construction, leading to not calculating large trip sizes

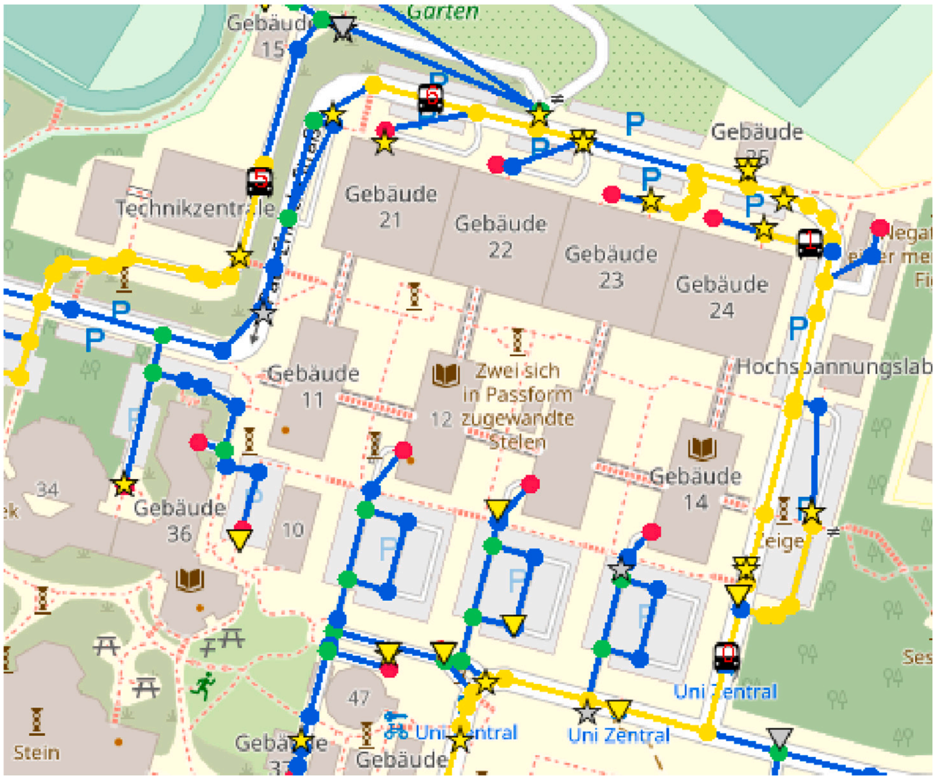


Fig. 9. Snapshot of an experiment after roughly 30 min. Triangles mark the pick-up location of requests; stars mark the target locations. Grey icons are finished tasks. Yellow-colored street segments are currently scheduled for at least one bus. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

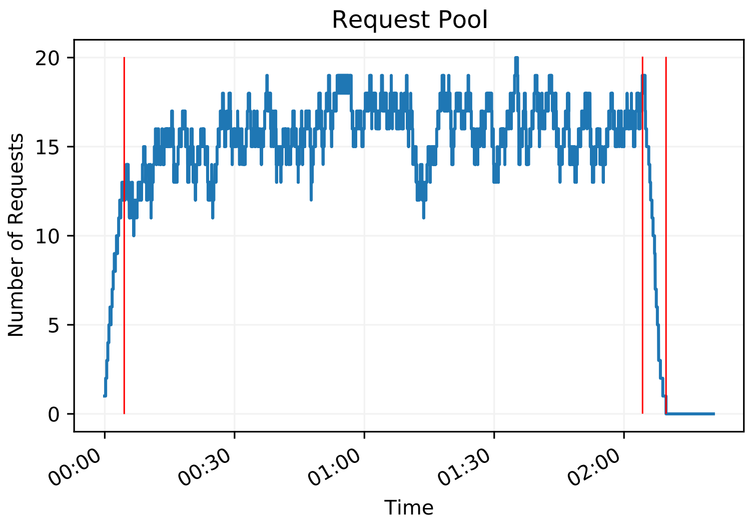


Fig. 10. Request pool size during an experiment with ($d_{wt\_max} = d_{td\_max} = 300$ s, $t_{req\_gen} = 15$ s).

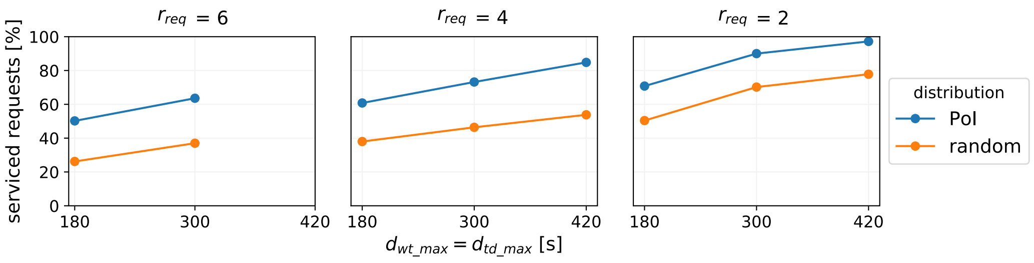
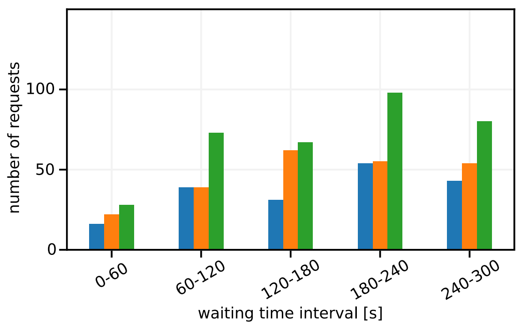
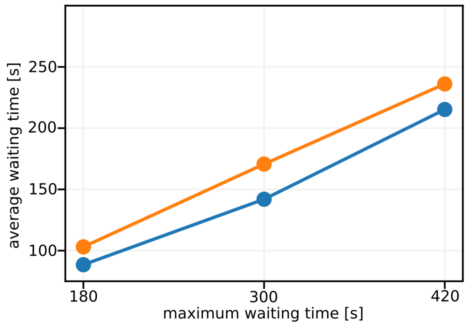


Fig. 11. Ratio of serviced request depending on request occurrence rate r_{req}, constraints d_{wt_max} and d_{td_max}, and the distribution (random/points of interest) of requests.

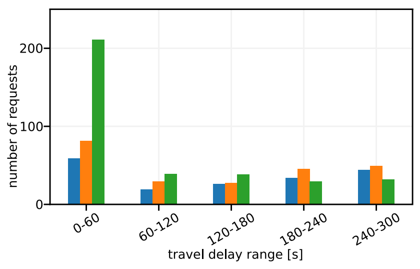


(a) Waiting time distribution for fully random request set. t_{req_gen}: $10s$ (blue), $15s$ (orange), $30s$ (green)

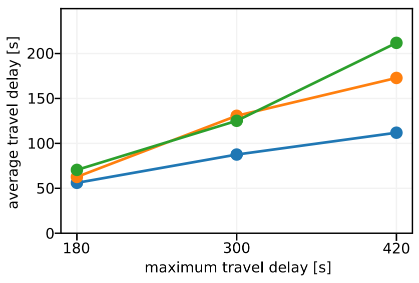


(b) Average waiting time depending on the distribution. random (orange), POI (blue), x-axis: delays, $t_{req\_gen} = 15s$

Fig. 12. Evaluation of waiting time distribution for serviced request. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



(a) distribution of the requests based on their travel delay in experiments with random distribution and max. delays of $300s$



(b) Average travel delay for random distribution based on max. delays (x-axis).

Fig. 13. Evaluation of travel delay for serviced request. t_{req_gen}: 10 s (blue), 15 s (orange), 30 s (green). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

but enabling the scheduling to work under high load. The timeout is chosen in a way that the scheduling is finished before the next cycle for the scheduling is started. If the timeout takes place the simulated annealing is stopped and the currently best solution for each trip is used to calculate the optimal assignment. Another way to reduce the long computation times would be to clear the request pool after each cycle. This would drastically reduce the computation time for the experiments presented in this paper. Since all possible requests are scheduled, it would not affect the number of serviced requests since a request which cannot be scheduled in the current cycle cannot be scheduled in the next cycle. For the real scenario, this would also enable the program to notify passengers if their request can be served after a short period. The removed requests could theoretically be assigned in later cycles for more dynamic scenarios. This can be the case if other requests are canceled, a bus is faster than expected, or new vehicles are added to the fleet.

4.1. Waiting time and travel delay

Another relevant factor of the scheduling is the utilization of the allowed delays. While the main priority is handling the largest possible number of requests, minimizing the waiting time and travel delay is also desired. In general, the waiting times are more or less equally distributed within the defined limitations Fig. 12(a). The low amount of requests with a waiting time below 60 s results from the fact that such a fast pick-up requires an available bus that happens to be in the right place. The probability of this is quite low due to the high load of the fleet in the experiments. It can be observed that the average waiting time has a considerable benefit from the distribution of the request, even though it is slightly lower for the POI requests.

When looking at the travel delays, the distribution looks different. Most requests have travel delays within a minute. Long delays happen but are not the usual case. Since delays caused by the environment are not considered in the experiments, all non-shared trips do automatically have a delay of 0 s. Delays only occur if multiple requests are handled simultaneously by one bus, and there is a significant distance between them, causing additional time in the bus. This explains why especially in the experiments with low request rates, many are handled with a very low travel delay Fig. 13(a). Furthermore, Fig. 13(b) shows that the average travel delay for experiments with low request occurrence and high possible delays is significantly higher than for the other experiments. This can be attributed to the scheduling utilizing the lower request rates and long possible delays to combine more requests, even if this would introduce higher travel delays. In other cases, the request pool is much larger and offers trips with lower travel delays while not serving requests that are harder to schedule. Another observation that can be made is that the travel delay is much lower considering the experiments with POI distribution Fig. 14. Here the request can typically be much easier combined. To further exploit this property to reduce the overall complexity of the scheduling, combining similar requests and scheduling them as a single request could be meaningful. This would lead to reducing the size of the request pool and, by that, reduce the complexity of the scheduling.

4.2. Bus insights

The utilization of the buses gives an indication of the optimal composition of the fleet. The results for a trial with $r_{req} = 4$ are shown in Fig. 15. It can be seen that, except for the start-up and decay phases, the buses are active most of the time. In the case of higher delay

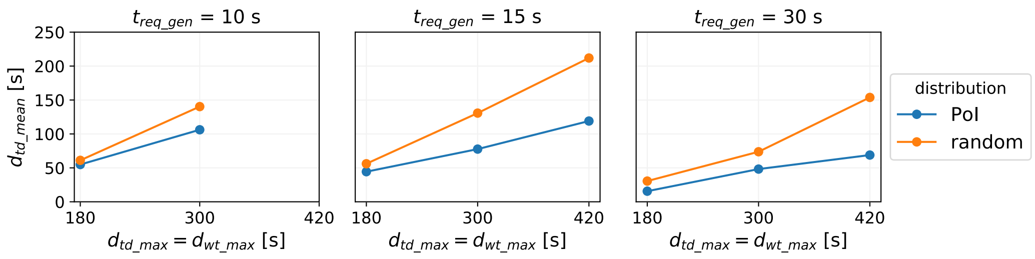


Fig. 14. Average travel delay based on the request distribution.

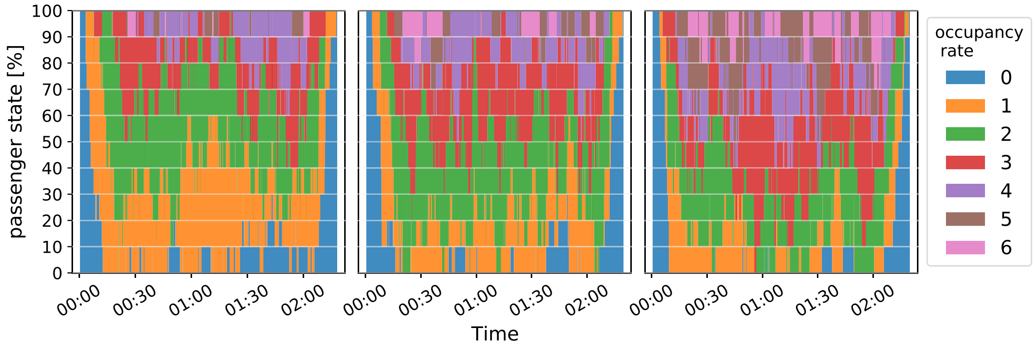


Fig. 15. Occupancy of buses over time for d_{wt_max} and d_{td_max} of 180 s (left), 300 s (center) and 420 s (right), $r_{req} = 4$.

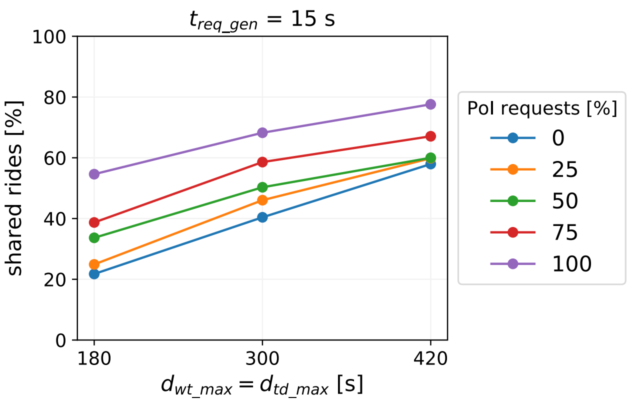


Fig. 16. Proportion of shared rides (trips with at least two customers) based on the distribution and the maximum delays using a request spawn rate of 15 s.

constraints, the utilization of the seating capacity is significantly higher. Another interesting metric is the number of shared rides. A trip is a shared ride if at least two passengers are on the bus during the trip. As shown in Fig. 16, the percentage of shared rides strongly depends on the composition of requests. Even though some buses are close to full capacity, most vehicles are underutilized.

Considering the current structure of requests, this leads to the result that a larger number of vehicles should be preferred over a smaller fleet with increased capacity. Nonetheless, it is also possible that restructuring the request data set would, for example, by considering group requests of multiple people, lead to varying results. Due to that, the use of real passenger data is required to make a precise assumption about the fleet composition.

5. Conclusion

In this paper, a scheduling approach for a mobility-on-demand service of autonomous buses focused on the utilization of shared rides has been demonstrated. The proposed work minimizes waiting time and travel delays while aiming to achieve high service rates. The used road network is extracted from open street map data and transformed into a graph structure. Based on the pool of currently open requests and the positions of the buses, all possible combinations of trips and vehicles are calculated. After optimizing the trips, an integer linear programming problem is formulated and solved, resulting in an optimal distribution of the trips to the buses. The concept was evaluated by performing various experiments using a bus fleet up to size 10, varying different parameters for constraints and the composition of passenger requests. Experiments on data sets of different structures showed a significant impact of the distribution on the scheduling results. This leads to the conclusion that real-world data on the request occurrence is needed to model the bus fleet successfully. Furthermore, in future experiments, the combination of simulated buses with at least one real operating bus should be performed to verify the scheduling procedure and investigate the robustness against delays.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] Qazi Hamza Jan, Jan Markus Arnold Kleen, Karsten Berns, Self-aware pedestrians modeling for testing autonomous vehicles in simulation, in: VEHITS, 2020, pp. 577–584.

[2] Qazi Hamza Jan, Karsten Berns, Safety-configuration of autonomous bus in pedestrian zone, in: VEHITS, 2021, pp. 698–705.

[3] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, Daniela Rus, On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment, Proc. Natl. Acad. Sci. 114 (3) (2017) 462–467.

[4] Mordechai Haklay, Patrick Weber, Openstreetmap: User-generated street maps, IEEE Pervasive Comput. 7 (4) (2008) 12–18.

[5] Hussein Dia, Farid Javanshour, Autonomous shared mobility-on-demand: Melbourne pilot simulation study, Transp. Res. Procedia 22 (2017) 285–296.

[6] Luis Martinez, Philippe Crist, Urban mobility system upgrade–how shared self-driving cars could change city traffic, in: International Transport Forum, Paris, 2015.
[7] Daniel J. Fagnant, Kara M. Kockelman, Dynamic ride-sharing and fleet sizing for a system of shared autonomous vehicles in Austin, Texas, Transportation 45 (1) (2018) 143–158.
[8] Seong-Woo Kim, Gi-Poong Gwon, Woo-Sol Hur, Daejin Hyeon, Dae-Young Kim, Sung-Hyun Kim, Dong-Kyoung Kye, Sang-Hyun Lee, Soomok Lee, Myung-Ok Shin, et al., Autonomous campus mobility services using driverless taxi, IEEE Trans. Intell. Transp. Syst. 18 (12) (2017) 3513–3526.
[9] Michael F. Hyland, Hani S. Mahmassani, Taxonomy of shared autonomous vehicle fleet management problems to inform future transportation mobility, Transp. Res. Rec. 2653 (1) (2017) 26–34.
[10] Paolo Santi, Giovanni Resta, Michael Szell, Stanislav Sobolevsky, Steven H. Strogatz, Carlo Ratti, Quantifying the benefits of vehicle pooling with shareability networks, Proc. Natl. Acad. Sci. 111 (37) (2014) 13290–13294.
[11] Javier Alonso-Mora, Alex Wallar, Daniela Rus, Predictive routing for autonomous mobility-on-demand systems with ride-sharing, in: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2017, pp. 3583–3590.
[12] Andrea Simonetto, Julien Monteil, Claudio Gambella, Real-time city-scale ridesharing via linear assignment problems, Transp. Res. C 101 (2019) 208–232.
[13] Patrick Fleischmann, Thomas Pfister, Moritz Oswald, Karsten Berns, Using OpenStreetMap for autonomous mobile robot navigation, in: Proceedings of the 14th International Conference on Intelligent Autonomous Systems (IAS-14), Shanghai, China, 2016, Best Conference Paper Award - Final List.
[14] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, Daniela Rus, On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment—Supplemental material, 2017.
[15] Emile H.L. Aarts, et al., Simulated annealing: Theory and applications, 1987.
[16] Romesh Saigal, Linear programming: a modern integrated analysis, J. Oper. Res. Soc. 48 (7) (1997) 762.
[17] M. Reichardt, T. Föhst, K. Berns, Introducing finroc: a convenient real-time framework for robotics based on a systematic design approach, Technical report, Robotics Research Lab, Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany, 2012.
[18] Max Reichardt, S. Schütz, Karsten Berns, One fits more-on highly modular quality-driven design of robotic frameworks and middleware, J. Softw. Eng. Robot. (JOSER) 8 (2017) 141–153.

Jörg Husemann studied Computer Science at the Technical University of Kaiserslautern, focusing on embedded systems and robotics. After finishing his studies, he continued his research as a Ph.D. student at the Robotics Research Lab. His research focuses on navigation and cooperation in vehicle fleets.