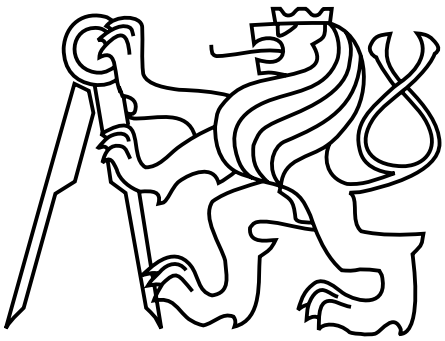
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science



Bachelor's Thesis

Demand-aware routing for ridesharing

Zdeněk Bouša

Supervisor: Ing. Martin Schaefer

January, 2020

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Bouša Zdeněk** Personal ID number: **456986**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Software Engineering and Technology**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Demand-aware routing for ridesharing

Bachelor's thesis title in Czech:

Navigování vozidel pro ridesharing zohledňující poptávku

Guidelines:

Ridesharing is a way to reduce the number of vehicles needed for transportation. The practical online ridesharing depends on the routing of vehicles. The routing should consider the demand that is usually not known in advance. Recently, the approaches that make use of machine learning for demand prediction to solve hard routing problems appeared in the literature.
1) Research the related literature on the ridesharing problem with a focus on the related problems solved by using neural networks.
2) Design and implement a solution to generate demand-aware routing with ridesharing.
3) Demonstrate the contribution of the demand-aware routing, discuss the scalability of the approach with the size of the problem.

Bibliography / sources:

[1] Abubakr Alabbasi, Arnob Ghosh, and Vaneet Aggarwal. Deeppool: Distributed model-free algorithm for ride-sharing using deep reinforcement learning. CoRR, abs/1903.03882, 2019.
[2] Qiulin Lin, Lei Deng, Jingzhou Sun, and Minghua Chen. Optimal demand-aware ride-sharing routing. IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, 2018. doi: 10.1109/MDM.2016.34
[3] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In International Conference on Learning Representations, 2019.

Name and workplace of bachelor's thesis supervisor:

Ing. Martin Schaefer, Artificial Intelligence Center, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **12.08.2019** Deadline for bachelor thesis submission: **07.01.2020**

Assignment valid until: **19.02.2021**

Ing. Martin Schaefer Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I would like to thank my supervisor, Martin Schaefer, for countless advice and steering me in the right direction while working on my thesis.

Declaration

I declare that I elaborated this thesis on my own and that I mentioned all the information sources and literature that have been used in accordance with the Guideline for adhering to ethical principles in the course of elaborating an academic final thesis.

In Prague on January 6, 2020 .....

Abstract

In this thesis, we focus on ridesharing and improving the pick-up delay. In densely populated cities, it is not sustainable for each citizen to own a car. Therefore, we can observe the rise of ridesharing services such as UberPool that utilize a vehicle capacity by sharing it among passengers. For real-time routing of the vehicle, it is common to use the shortest path planner and insertion heuristic, which takes care of vehicle-request assignment and it is also responsible for the appropriate order of pick-ups and drop-offs. We propose a new heuristic path planning algorithm that takes into account the chance of future demand in the vehicle surroundings while following the planned path. We simulate our approach to the problem on demo data as well as on data based on Prague demand model. The results confirm that the proposed algorithm leads to lowering of the time needed for pick-up of a request, compared to using the shortest path planner. The created framework and proposed path planner can later serve as a baseline for further improvements in the description of surrounding for the vehicle.

Key words: ridesharing, A^*, heuristics, demand-aware, agent simulation

Abstrakt

V této práci se zaměřujeme na spolujízdu a zlepšení čekacích dob na obsloužení. V hustě obydlených městech není výhodné pro každého obyvatele vlastnit auto. Je to jeden z důvodů, který vedl k rychlému růstu služeb jako je UberPool. Tyto služby využívají kapacity vozidla a jejího sdílení více cestujícími. Pro navigování vozidla v režimu spolujízdy se obvykle využivá plánovač, jehož výstupem je nejkratší cesta, a algoritmus pro přiřazovaní požadavků k vozidlům, který se navíc stará o naplánování vhodného pořadí vyzvednutí a vyřízení poptávek. Navrhli jsme nový heuristický plánovací algoritmus, který zohledňuje pravděpodobnost výskytu žádosti o spolujízdu v nejbližším okolí vozidla v průběhu jeho navigování. Navržený přístup jsme simulovali na demo datech a také na datech z pražského modelu poptávky. Na základě výsledků jsme ověřili, že námi navržený algoritmus vede ke snížení čekacích dob na obsloužení žádostí v porovnání s využitím plánovače nejkratších cest. Vytvořený testovací framework a navržený algoritmus může v budoucím výzkumu posloužit jako základní kámen pro další vylepšení popisu situace v okolí vozidla.

Klíčová slova: ridesharing, A^*, heuristiky, zohledňující poptávku, agentní simulace

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This work deals with demand-aware routing in the road network that improves the level of quality in ridesharing systems. In densely populated cities, it is not sustainable for each citizen to own a car. Thus, we can observe the rise of services such as UberPool, Liftago, and many more. Such services use state-of-art technologies to provide faster pick-ups and cheaper transport for passengers by sharing a vehicle's capacity, also known as car-pooling. The passengers might seek faster travel times and lowering the cost of their trip; the same applies to the service providers. Operators are looking forward to lowering their traveled distance, overall operating costs, and to attracting customers, for example, by shorter waiting times. Current approaches focus more on combinatorial optimization of known-ahead demand and fleet delegation. After that, they usually plan using the shortest paths. Therefore, the goal of the thesis is to describe a system that estimates future demand based on historical demand and diverts from the shortest path to a more economical path to become a real-time ridesharing path planner. Such a route will provide an increased chance of another request to be assigned to the routed vehicle, shorter response time to new requests in exposed locations (shopping centers, train stations, downtown), and economic benefits for both providers and customers.

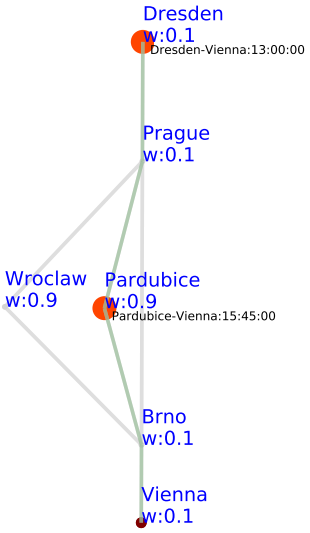


Figure 1.1: Synthetic graph with an actual (green) path of an agent. The graph consists of multiple points (nodes) that are pick-up and drop-off points for demands. In the picture, there are two demands at two different times. First demand request from $Dresden \rightarrow Vienna$ made at 13:00 and second from $Pardubice \rightarrow Vienna$ made at 15:45. The same agent completes both requests. In this case, the agent diverts at point $Prague$ from the shortest path before he is assigned with the demand $Pardubice \rightarrow Vienna$.

1.1 Motivation

Imagine a passenger traveling from point $Dresden$ to point $Vienna$ using a taxi, as shown in Figure 1.1. To lower a total trip cost, the customer allows the taxi driver to pick up another passenger and take a small detour. This situation would be called ride-sharing. In the best-case scenario, the taxi driver would have information about other trip requests from point $Pardubice$ to point $Vienna$, and it is up to him to decide whether to pick that demand or whether not to. However, it is not uncommon that the driver would not know about the future demand $Pardubice \rightarrow Vienna$ at the point $Prague$ when he is traveling with a passenger $Dresden \rightarrow Vienna$. How would he decide at point $Prague$? Will he use the shortest path, or is it cost-optimal to try a longer path with a chance of another passenger traveling in a similar direction? This work focuses on demand-aware routing that will provide an answer to these questions.

It is essential to keep in mind that the online ridesharing is not aware of exact demand in the future. Therefore, we would not know about the demand before the request is made. Take, for example, the driver when he is at point $Prague$ at 15:00:00 and there is no knowledge of the demand $Pardubice \rightarrow Vienna$. The demand will pop-up in the system at

15:45:00. However, the driver or dispatcher has been observing the road graph for a more extended period and knows that at point *Pardubice* and there is a larger city and, therefore, a higher chance of a starting demand at that point. So then, the driver is advised to detour through point *Pardubice*.

1.2 Goals and organization of the work

There are three main goals that are divided into multiple sections.

The first goal is to research the current state of knowledge. Chapter 2 focuses on theoretical context, approaches of other researchers and summarizes their work. Appendix B is a brief overview of all commonly used methods.

The second goal is to propose a new approach based on the research of current state. The problem formulation is in Chapter 3 and is later used in Chapter 4, which focuses on description of proposed solution and its algorithms for demand-aware routing.

The third goals is to show what the impacts of the proposed solution to online ridesharing are. There is Chapter 5 with experiments to serve this purpose. The summary in Chapter 6 discusses achieved results. In addition, there is a brief thought of future work.

Chapter 2

Work context

In this chapter we focus on the surrounding of the ridesharing problem and its definition. We also investigate related works from other researchers in order to understand the problem fully and be aware of dead-ends when proposing a new approach.

2.1 Ridesharing

The ridesharing is an increasing mode of transport that improves the economy and permeability of travelers by sharing and splitting the travel cost among passengers. It is often promoted as the utilization of empty seats in a vehicle. It does not matter whether that is a personal car or a cab. Modern approach allows to match travelers even though they do not share the same pick-up and drop-off points. [18]

A solution to the ridesharing problem is quite complicated, and in recent years various techniques to solve it have been proposed. Many of them try to solve the problem for known pick-ups ahead, which is not suitable for a dynamic and changing environment such as transportation is. Lately, some of the proposed techniques focus on less traditional techniques that have roots in the field of machine learning. In many cases, this approach balances shortcomings of algorithms for the offline setting, and mainly it counterbalances a lack of prediction of the future demand. [22]

2.1.1 Problem division

Based on literature and articles, there are three main directions to solving the ridesharing problem. The first, known to be NP-hard, is an offline variant and accounts only for known demand. According to the article [21] the proposed solution is based on the 2.5-approximation algorithm.

The second is demand-oblivious, assuming no knowledge of future demand. This approach is more suitable for the online decision, where we do not know precisely a future demand. However, this direction does not adapt very well to new patterns[1].

The third is demand-aware that benefits from machine learning development, demand can be learned. It best fulfills the assumptions to have overtime adaptable, demand prediction model, and algorithm for ridesharing. Assumptions of future travel distribution only support decisions. [17]

2.2 Related problems

To be able to dive into the current state and suggest a new approach, it is vital to understand related problems.

Pick-up and delivery problem and its variant with time windows (PDPTW). In this problem, every request wishes to be served as soon as possible. The main goal is usually to minimize a combination of the time need and satisfaction with the service (also known as reject rate). Usually, the main algorithm of the solution is constructed as TSP, and then is the result just randomly picked up. More can be found in [19].

Dial-A-Ride problem is a pick-up and delivery problem where the destination of the request is uncertain. The problem is very close to the online ridesharing problem, and the only main difference lays in the knowledge of a destination. [7]

[1] By that we mean pattern that is created by input data, such as demand variation in time in different days, and so forth.

Traveling salesman problem can be imagined as the TSP is modeled as a fully connected weighted graph $F(V_f, E_f)$ above graph $G(V, E)$ that represents road network, such that paths are edges E_f, points are graph's nodes E_f and distance between nodes is the weight of the belonging edge. Vertices $V_f \subseteq V$, therefore they can be connected to create a fully connected graph F. Instance is complete undirected graph $(K_n >= 3)$ and weights $c : E(K_n) \to R^+$. The goal is to find a Hamiltonian cycle of G with minimum cost. [9]

The graph can be transferred to a directed graph as long as it remains a fully connected graph; the goal remains the same. In this situation, we would deal with an asymmetric TSP. [2]

L is strongly NP-hard if there is a polynomial p such that L_p is NP-hard. If L is strongly NP-hard, then L cannot be solved by a pseudo-polynomial time algorithm unless $P = NP$. [2]

2.2.1 Shortcuts and problem derivations

For easier orientation in the topic, a list of related problems that can be more or less[2] substituted is provided below.

- DART - Demand-aware routing, does not describe how to assign individual requests.
- DRS - Dynamic ridesharing service, often referencing a system that assigns individual requests on short basis. This method is usually used for short-trips in cities. [13].
- DARP - Dial-a-ride problem - more can be found in Section 2.2.
- PDP - Pick-up and delivery problem - more can be found in Section 2.2
- VRP - agent routing problem - is a PDP in which all the origins or destinations are located in a default vertex.
- PDPTW - Pick-up and delivery problem with time windows.
- CVRP - Capacitated agent routing problem, derivation of the TSP. A limited number of agents with limited capacity.

[2]Some of problems provide basic ideas on how to approach the possible solution and often it is possible to use this a basic guideline for the implementation.

- VRPTW - agent routing problem with time windows, derivation of the TSP. It is a defined series of time windows when the customer would accept a drop-off.
- PCTSP - Prize collecting TSP and it is stochastic version, that introduces uncertainty.

2.3 Related works

The ride-sharing problem poses many challenges, therefore it has been investigated by many articles. The main articles, which directly address the ride-sharing problem, are:

- DeepPool [1] is a decentralized version of the DART/ridesharing problem, thus each vehicle decides independently, the decision is supported by its own deep Q-network (Section B.2) which predicts a future demand. Each vehicle learns based on its own actions. The proposed model-free approach means that there is no accurately modeled system and it rather uses reinforced learning to improve its dispatch and routing. However, this requires constructing a simulation environment.
- Ridesharing Problem with Flexible Pickup and Delivery, described by the paper [22], proposed a hybrid method based on a greedy algorithm and dynamic programming for the knapsack problem, therefore the representation of the PDPTW is more spatial. *"To serve passengers 1 and 2, vehicle h_1 is sent out to travel according to a feasible routing plan, which is denoted by the space-time path marked as the green line in Figure 2.1. We can see that initially picks up passengers 1 and 2 at space-time vertexes (2) and (4) and then delivers passenger 1 when passing (10) and eventually delivers passenger 2 at (13)."* [22] This work focuses on offline version of ridesharing.
- The simulation case study of the ridesharing impact in mobility on demand system [10] focuses on a impact of a dynamic DARP with large fleet of agents, such as a change from current situation to fully functional ridesharing in Prague. The article adopts insertion heuristic for request-agent matching and for routing uses shortest A^* path planner. In order to simulate demand and impacts credibly they use multi-agent simulation. Based on their results there is an increasing relation between average occupancy of the agent and maximum delay time.

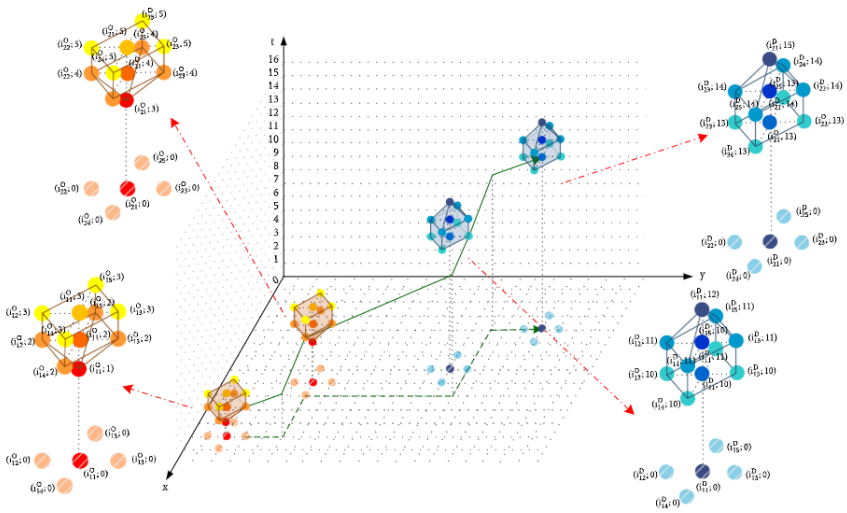


Figure 2.1: Spatial time-route representation. Source: [22]

There are also studies addressing the more general topic of the TSP and routing-optimization problems in general. Very promising study [14] is using neural network to solve TSP and some other related problems. Their approach is mainly focused on the acceleration of the algorithm and to achieve this, they focus on bringing a new combination of models into the solution. The key factor of their solution and improved learning is the change from the Pointer network during training to the Attention model described in Section B.2. The final path is then produced by a decoder that resolves path from selected starting vertex and in each vertex, it decides based on computed probabilities by the neural network, therefore it predicts the best route based on encoded data. The encoding can contain various data in multiple layers that are then passed to the multi-head attention model. They provide a source code under MIT license[3]. This approach can be reused for the ride-sharing problem, however, it has to be used as a centralized dispatch heuristic to choose the best path and predict future demand, due to being a graph-based method.

[3] <https://github.com/wouterkool/attention-learn-to-route>

Chapter 3

Problem formulation

This chapter focuses on problem statement. We describe the required road graph, demand and ridesharing. However, in order to be able to build definitions in a logical sequence, the definition of ridesharing is at the end of this chapter in Section 3.3.1.

3.1 Graph definition

For further work, we formalize the graph as $G(V, E)$ where:

- V represents a set of all vertices (also called nodes).
- E is a set of all oriented edges $e(v_1, v_2)$ where $v_1, v_2 \in V$ outgoing from vertex v_1 to vertex v_2.
- Graph G does not contain parallel edges.

Each edge e may contain additional data, such as curvature, maximum allowed speed, length, that would be called weight (or price) of the edge. Also, each vertex v can have extra data, such as a note of point of interest [1] or any other useful data for routing (lanes configuration, and so forth.). [9]

[1] Terminus station, shopping center, and so on.

3.2 Demand definition

The definition is based on model presented in [21]. Request d_i consists of the creation time τ_i, an origin node $s_i \in V$ and destination node $t_i \in V$. Reject time, reward and other attributes are defined by the scenario as variables. We assume that travel requests are independent on each another and an agent can pick up requests up to its capacity limit. Requests can be fulfilled in numerous ways as shown in Figure 3.1. Depending on the scenario and criteria it is possible for an agent to reject the request and go through vertex s_i without picking up a request. We also assume that a request can be canceled, unless reject time is achieved, also it is not possible to change the location of origin or destination and it is not possible for a request to be processed by more than one agent.

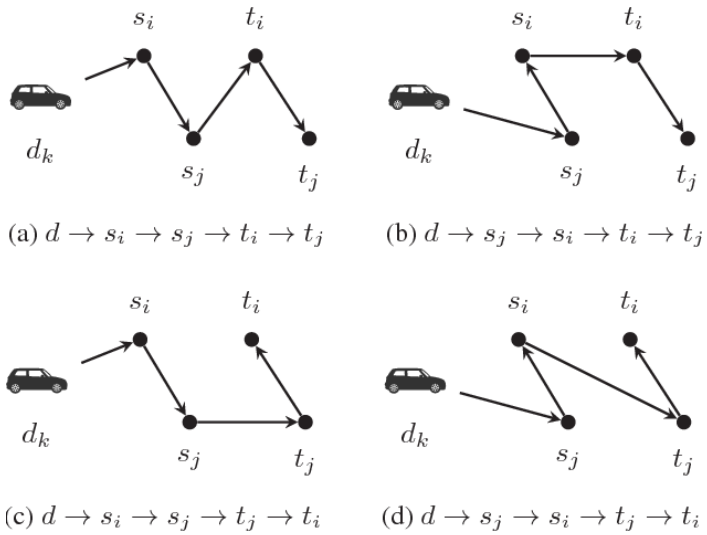


Figure 3.1: Demand definition. Example of requests and many ways on how to fulfill. Source: [21]

3.2.1 Model definition

For our proposed solution, we extend the definition of demand. Vertices' ids "from_id", "to_id" are required to be part of graph G, therefore $from\_id, to\_id \in V$. The request will be shown at time at that is represented by an integer as the Unix Timestamp [2], hereafter referred to as τ.

[2]The timestamp value is the number of seconds since January 1st,1970 at UTC.

The transportation requests in the testing environment are represented as a set of requests $D = \{d_0, d_1, ...d_i\}$ and the i-th request is revealed at time τ_i; $i < M, i \geq 0$; M is total number of served requests. Assigned request d to agent a can be only picked up when $\tau_d > \omega$, where ω represents current time in transportation system. Additionally, request d can be picked up by agent a when the starting point of request s_d equals the position of agent v_a. Request d can be assigned to only one agent from set A.

For each request, d the shortest path is computed using a simple A^* algorithm, and such path is then converted to travel time κ_d using maximum allowed speeds on edges. For simplification each request occupies one seat in the agent.

Request can be in following states:

- New - immediately after the request is shown in the system at τ.
- Not assigned - immediately after the request is shown in the system at τ and in case when the request parameter exceeds limit set by scenario during assignment to the agent[3].
- Rejected - in case of scenario with pick-up limit, the request is rejected or canceled if it is not picked up in limit.
- Assigned - the request is assigned to the agent that has to pick it up. The request cannot be reassigned.
- On-board - the request has been picked up by the agent and occupies one seat.
- Finished - the request was dropped off at the destination.

3.2.2 Distribution of demand

The request has to be assigned to an agent, either by the dispatcher for centralized systems or by the agent itself for a decentralized system [6].

[3]For example the system is overloaded and total travel time will be higher than the request requires.

Dispatcher knows all positions of agents and decides which request will be served by which agent. It is often that the dispatcher also inserts the request into an agent's plan and sets the order of vertices. This approach benefits from overall knowledge, the cooperation of agents, and is preferred for fleet optimization. However, it might disadvantage individual requests.

Decentralized system is often based on the competitive environment, first to show wins, or on an auction system, best offer wins (best total travel time or cost). This is beneficial for individual requests, but on the other hand, it may lead to increased pick-up times and to non-optimal fleet management. The request is picked up by the agent, assigned to it by broadcast and it is up to the agent to pick the best strategy to insert the request into its drive plan.

For both methods, it is often used distance-minimizing request-agent assignment. It is attempted to incorporate requests into each agent's plan. Operational costs and prolongation of other already assigned requests are checked then. For description of such an algorithm check paper [10] with the insertion heuristic based on the shortest distance.

3.3 Ridesharing definition

Formal definition of ridesharing. Introduction to ridesharing is in Section 2.1.

3.3.1 Model definition

Let $A = \{1, ..., n\}$ be the set of all agents available to fulfill request d. Each agent $a \in A$ has its position at any vertex $v \in V$ at a specified time[4].

Agent $a \in A$ has its plan p_a that can be changed either when the agent is assigned with a new request d or when it is at any vertex v. Plan p_a contains all demands assigned to agent a. However, the order of $s_1, t_1, ...s_x, t_x$ in plan p_a is uncertain and is subject to proper path planning; x is the number of assigned requests to the agent. Based on the scenario, the agent cannot pick more requests that it can carry on-board.

[4]Such time is pre-computed by the planner in the simulation.

The agent's plan p_a contains a subset of vertices $R, R \subset V$ that creates an order of vertices in which the agent has to appear. Each vertex v can repeat itself multiple times in the p_a, implementation details in Section 4.1.1.

3.3.2 Quality criteria

Values are used as arguments to quality functions, and provide us valuable insight into the whole designed ridesharing system. These commonly used values were picked up based on the study of materials in Section 2.3.

- Reject rate - provides necessary feedback from customers when picking them up. The measuring function is defined as a time tracker for each request, starting when the request is created, and stopping once the request is satisfied or preset time expires. Based on whether the request is satisfied or not, the rate is calculated. Measured in %.
- Idle time of an unoccupied agent - is time when the agent has to wait for a pick-up. Measured in seconds.
- Idle time of an occupied agent - is time when the agent has to wait for a pick-up while it has other passengers on-board. Measured in seconds.
- Time extension above minimal trip time - (also known as in-car delay) describes how much the trip-time has extended for each passenger in the agent. Measured in seconds.
- The maximum capacity of an agent - is a limitation of how many passengers can occupy an agent at the same time. For average-sized car, the value would be 3.
- Not-occupied trips time - is a value that provides information about economics effectiveness. Measured in seconds and compared to total moving times.
- Cost of travel - it is often a small market-oriented economic model that is focused on a balance between customer and service provider. Based on the quality of the service, a provider is rewarded, and a customer is given a discount.
- Ratio between average occupancy of an agent and average delay time of request.

Chapter 4

Solution design

Pursuant to the study of the theory and the context we concluded that it is beneficial to propose a solution which is:

- Demand aware - is obvious that some locations (airports, shopping centers) are more eager for service than others. Thus, it is better to route agents to busier places based on historical demand.
- Quality guaranteeing - is important for customers to limit excesses in routing and provide a comparative function for different variants of the approach, more in Section 5.2. Also, take into account a market-oriented economics. Such function is defined as the passenger cannot spend more time aboard of the vehicle than time ζ, for example, a $\zeta = 2 \times \eta$, where η is travel time of the shortest path for the passenger's request.
- Centralized - is an approach that provides a better overview of demand and can dispatch agents effectively. Thus, agents, instead of competing, cooperate.
- Adaptable - for the cost-effectiveness of provided services, it is important to be able to adapt over time to new demands' patterns[1].

This will allow us to achieve lower pick-up times while maintaining the quality of service (trip prolongation) from the perspective of passengers, and in case of high demand density, it will lower the total traveled distance of agents.

[1] By that we mean new routes to new important places with higher demand.

Our solution is, therefore, delimited as an agent-request assignment and agent's routing in an uncertain environment, also known as DART. Thus, we propose a solution where we use adjusted Insertion Heuristic for agent-request assignment and a modified A^* path planner. We called the planner a situation-aware A^* planner because it takes probability of demand along the planned route and adjusts its route accordingly. The likelihood of demand is pre-learned and is further trained while the system is in operation. That will allow to adjust routing for behavioral changes.

To be able to keep up with periodical changes in demand, we fix the calculated probability of demand to epoch of 3 hours. Thus, to simulate the whole day, it is necessary to split it into multiple epochs. Furthermore, we simplify our approach by not taking into account the capacity of roads and waiting for spots.

We test our proposed solution with scenarios described in Section 5.2 and show results in Section 6.1.

4.1 Environment

For all of the proposed testing scenarios, we need to implement an agent simulation that will provide routing, time computing, and quality scoring. The simulator is implemented from the very beginning. For such implementation, we need to describe a modified road graph, an agent simulator, and how to plan routes.

4.1.1 Agent simulator

The simulation is based on paradigm of Discrete Event Simulation (more in Appendix B.3). The step is represented by a variable that can be adjusted, and the default value is 1 second.

Each of the critical areas is implemented in such a way that allows easy replacement and thus gains the advantage of heredity. Proposed interfaces and their implementation:

- Path planner - shortest, situation-aware A^* planner
- Distance approximation - heuristic function such as the Euclidean distance (Section B.1)

- Agent-request matching - simple, random, insertion-heuristics

Environment of the simulation is created based on road graph. The file structure of the simulator is pictured in Appendix C. The road graph, node weights, and requests can be saved in JSON for reuse.

Agent Each agent's task is to fulfill assigned requests and follow its drive plan. Also, each agent supports *pick-up* and *drop-off* action for each assigned request d in its plan p_a. The *pick-up(d)* is only possible when the agent is at a vertex, which is the same at the request origin point. The *drop-off(d)* is possible once the agent reaches the request destination point. Such actions are recorded in the history of the agent for later research and statistics.

Agent states - For each agent, we observe the following states:

- empty, sitting at one spot and available for pick-up
- empty, roaming towards a pre-selected spot and available for pick-up
- fulfilling one or more requests and available for another pick-up
- fulfilling one or more requests and not available for another pick-up

Planning data structure is composed of multiple *planning nodes* that are in an agent's *drive plan*.

The *planning node* is composed of its position at vertex v, agent's time of arrival to this node, and time of departure from this node. This time gap allows the agent to pick up, to drop off a request, and to wait for some time at the vertex. The arrival time to node v_n cannot be smaller than a sum of travel time and time of departure from previous node v_{n-1}.

The *drive plan* of an agent keeps the order of planning nodes in which order the agent moves. In addition, it provides a function to process the correct movement of the agent. Each time an agent is assigned with a new drive plan, the previous is stored in history.

Each time the agent is due to departure from the current planning node, the current planning node is removed and replaced with a next planning node from the agent's drive

plan. Drop-off and pick-up are processed once the agent is time-wise available in the current planning node.

4.2 Algorithms

In this section, we focus on the core of the proposed solution.

4.2.1 Agent-request assignment

Insertion heuristic is used for assignment of request d to agent a, where $d \in D, a \in A, A$ is a set of all agents and D is a set of all requests [10]. The request is tentatively added to plan p of each agent a. The *dropoff(d)* point has to be inserted after *pickup(d)*. Additionally, the order of previously added requests to agent's plan remains the same. The function δ measures the operation cost based on selected testing scenario. The χ provides additional restrictions for the new plan to be valid. For the pseudocode see the Algorithm 1.

Proposed extension of insertion heuristic is as follows. The total drive plan p travel time of vehicle a, $\delta_{a,p}$ is defined in Equation 4.2, this is then used in function to obtain ξ. The ξ function is defined as shown in Equation 4.4. Additionally, for each $\Delta_{a,p,d}$ we check prolongation constraint ζ defined by scenario, the computation of it is shown in Equation 4.1.

In case of scenario that allows quality function for pick-up time, we allow measuring λ_d that equals to the difference between the time of pick-up and the occurrence of request d. If it is not allowed by scenario, then $\lambda_d = 0$.

For path planning between vertices in plan p_a the proposed situation-aware A^* path planner (Section 4.2.3) or the shortest A^* path planner (Appendix B.1) is used.

Algorithm 1 Finding an agent a for request d. [10]

function ON NEW REQUEST(d)
  for all $a \in A$ **do**
    let p_a be the current plan of a;
    let ψ be the number of vertices in current p_a;
    for $i \in 1, ..., \psi + 1$ **do**
      for $j \in i + 1, ..., \psi + 2$ **do**
        $p_a^i \leftarrow$ insert *pickup(d)* order before position i in plan p_a;
        $p_a^{ij} \leftarrow$ insert *dropoff(d)* order before position j in plan p_a^i;
        $a^*, i^*, j^* \leftarrow argmin(\xi)$; ▷ Example of $\xi = \delta_a(p_a^{ij}) - \delta_a(p_a)$, request d is assigned to agent a^* and is subject to additional restrictions χ;
      end for
    end for
  end for
  request d is assigned to agent a^*
  agent a^* follows plan $p_{a^*}^{i^*j^*}$;
end function

$$\zeta_d = (\zeta \kappa_d) + \lambda_d \tag{4.1}$$

$$\delta_{a,p} = \sum_{d=1}^{d} \Delta_{a,p,d} \tag{4.2}$$

subj. to

$$\{\Delta_{a,p,d} \leq \zeta_d\} \tag{4.3}$$

$$\xi = \delta_a(p'^{ij}_a) - \delta_a(p_a) \tag{4.4}$$

$\xi-$ quality of new plan compared to current feasible plan

$\delta_{a,p}-$ is the total travel time of requests in drive plan p_a

$\Delta_{a,p,d}-$ is travel time of request d in plan p_a for agent a

ζ_d- travel time limit of request d

$\zeta-$ scenario constraint, for example 2 $\times$ travel time of the shortest path for request

κ_d- the shortest path travel time of request d

λ_d- the time between occurrence of request d and its pick-up

4.2.2 Demand probabilities and learning

Definition - In a given vertex the request occurs $x = 1$, does not occur $x = 0$, with probability P and $1 - P$ respectively. Where P is a parameter of alternative distribution $Alt(P); 0 < P < 1$. For simplicity, we choose sample mean which is described in Equation 4.5. Thus, we assume that values are identically divided[2] and we support this approach by slicing time-periods into epochs.

$$P_v = \frac{1}{n} \sum_{i=1}^{n} x_{i,v} \tag{4.5}$$

$V-$ set of vertices

P_v- probability of demand at vertex $v \in V$

$n-$ sum of historical demands $d \in D$ starting at any $v \in V$

$x_{i,v}-$ represents if the request occurs or not at vertex v.

$D-$ all requests in the system, starting at any $v \in V$

[2]We do not expect fluctuations in values

Learning of probability P_v at vertex v is updated after the run of the epoch. Thus, we allow to add all new requests to the D and increase the number n. Therefore, we recalculate the probability for each node using Equation 4.5.

Clustering of close proximity nodes on real roads' graph. Because such a phenomenon of close proximity nodes is not uncommon, we use DBSCAN clustering on real roads' graphs. We preset the algorithm with $eps = 50$ meters and $minPts = 3$. Then, the demand probability is computed in such a way that we treat all clustered core nodes as one and compute the probability using Equation 4.5. Then, the result is a probability (weight) of each core node in the cluster.

Normalization coefficient is used in case a vertex with probability $P_v \geq 0.9$ does not exist. Then the coefficient is computed as follows. We find the vertex with maximal P_v and thereafter the coefficient is equal to $0.9/max(P_v)$. Next, probability P_v for every vertex $v \in V$ is multiplied by this coefficient.

4.2.3 Situation-aware A* path planner

We modify A^* path finding algorithm in such a way that takes weights at vertices into account. We did this by recalculating the edge weight in the planning algorithm. The proposed recalculation is described in the Equation 4.6 and is performed in each step of the algorithm in which a weight of an edge is requested.

$$e'_w = e_w - (e_w v_c) \tag{4.6}$$

e_w– weight of $e \in E$, also known as probability

e'_w– recalculated weight of e for the path planning algorithm

v_c– probability weight at neighbour vertex $v_d \in V$, v_d is also target vertex of the edge e, $0 < v_c < 1$

Chapter 5

Experiments and results

This section is dedicated to the testing of our proposed solution. We describe how we approached experiments, what the scenarios for tests are, and what variable we used.

5.1 Testing environment

The environment variables influence the outputs greatly, so here we describe the acquisition of the test demand, road network, and simulation tools.

5.1.1 Road graph processing

The road network is represented by a connected directed graph that does not have to be complete and contains multiple components, thus can contain one-ways. Snapshot from OpenStreetMap (OSM) [1], that provides free open map data, also usually contains dead-ends, and therefore it creates a disconnected graph. To eliminate these shortcomings, it is necessary to process the whole graph and delete dead-ends and correctly process boundaries of the graph at the nearest vertex with degree bigger than two (crossroad).

To do so, there are suitable tools for map processing that are provided by AIC FEL CTU called RoadMap-Processing[2]. After necessary processing, a suitable graph might look like

[1] <https://www.openstreetmap.org/>

[2] <https://github.com/aicenter/roadmap-processing>

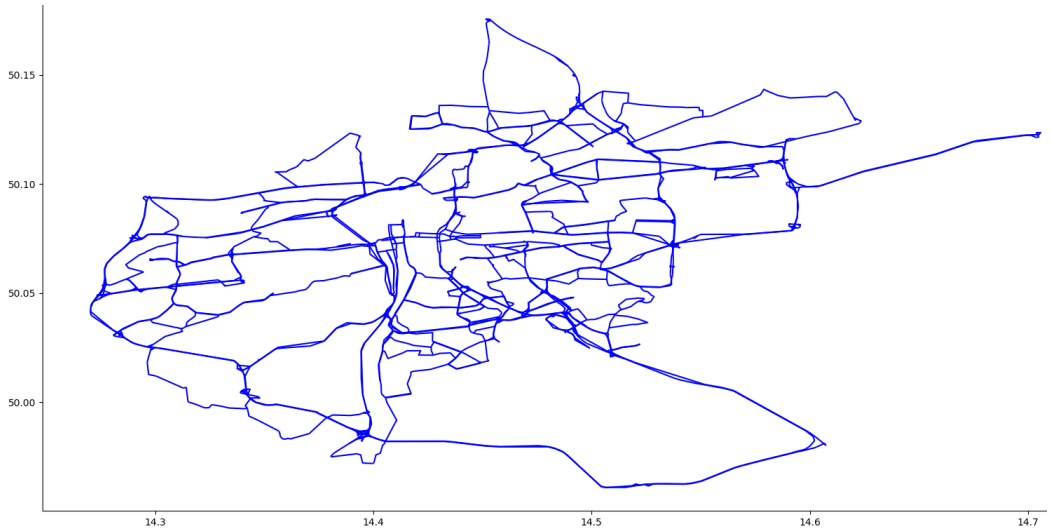


Figure 5.1: Simplified Prague's map.

the one in Figure 5.1 and has to contain exactly one strongly connected component. Thus, there exists a directed path for every pair of vertices. [8]

The structure of graph is represented by JSON in following format:

```
{
   "type":"FeatureCollection",
   "name":"demo",
   "crs":{
      "type":"name",
      "properties":{
         "name":"urn:ogc:def:crs:OGC:1.3:CRS84"
      }
   },
   "features":[
      {
         "type":"Feature",
         "geometry":{ "type":"LineString",
            "coordinates":[
            [ 14.434614, 50.191457],
            [ 14.435089, 50.120986]]
         },
         "properties":{
            "id":1,
            "osm_id":1,
            "type":"highway",
            "name":"1",
            "tunnel":0,
            "bridge":0,
            "oneway":1,
            "class":"highway",
            "utm_coords":[
               [ 45964170, 55600710 ],
               [ 45961629, 555223540]
            ],
            "length":783573, # in cm
            "maxspeed":50, # in kmph
            "lanes":1
         }
      }, # And more features (edges)
  ...
  }
```

Demo graph is used for demonstration of the approach and contains vertices $V = \{A, B, C, D, E\}$. Its structure is pictured in Figure 5.2, and edges are directed in both ways. Each node has to be assigned to some location in real GPS coordinates, because the location is used later for distance computing.

Prague graph is based on real Prague road network, which is pruned to contain only primary roads. The road graph is strictly directional. The graph consists of 1385 vertices and is shown in Figure 5.3.

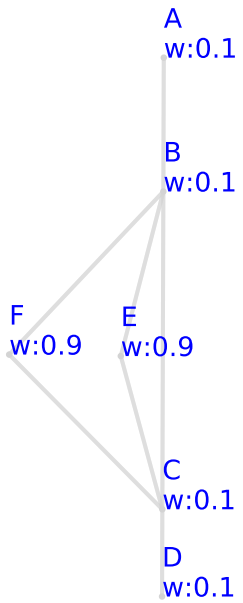


Figure 5.2: Simple graph, w represent probability at vertex p_v

The original OpenStreetMap (OSM) [3] source file consists of data both for edges and nodes. The obtained data usually represents rectangular cutouts from maps and therefore contains multiple dead-ends, abandoned, and unrelated elements. Thus, it needs to be processed for strongly connected components, pruning of unrelated data and edge simplification[4].

Clustering and further simplification In many instances even the simplified road network may remain excessive[5], therefore, it needs further simplification

[3] <https://www.openstreetmap.org/>

[4] This means, that multiple edges that create just a line without intersection can be joined into a single edge.

[5] For example, the biggest optimally solved TSP has an instance of 85900 vertexes and the Prague's simplified map can contain twice that.

Figure 5.3: Prague graph with nodes that are presented by dark blue dots. The probabilities in nodes are not clustered.

One of the approaches consists of pruning of less important edges. If we look into the code sample in JSON format, we can see that in key "properties" there is "type". This key can take several values - highway, primary, secondary, tertiary, pedestrian, and these values are specified by the source map. For ridesharing, it is not vital to account for each road, it is more important to be able to distinguish important places with high demand for the starting of trips, and therefore we can prune away some of the edges. Other approach would be clustering of the graph into the Voronoi diagram.

5.1.2 Demand processing

For testing our approach, it is crucial to have the most real data sample. On the other hand, it is useful to have data which is simple enough and comprehensible for implementation and testing.

Requests's JSON format:

```
{
  "requests":[
    {
      "from_id":4596416985560071088, # id of node in graph G
      "to_id":4595404605528520538,
      "at":1571825800, # time when the request is created
      ... # and other data such as penalty for delay and so on
    },
    ...
  ]
}
```

Demo demand is manually preset to demonstrate the goal of the proposed solution and it is simulated on graph from Figure 5.2. We set demand from A to D and, later in time, demand from E to D. This is used for illustrative demonstration of tested algorithms.

Synthetic demand is needed for implementation and testing, and it is overall more straightforward to obtain it than a real demand for a specific city. For our case, we would generate demand that is based on the probabilistic value assigned to each vertex[6].

We propose Equation 5.1 that allows us to generate synthetic demand for each vertex (starting node of request) for a requested count of requests. We generate a required number of requests starting at each selected vertex. Then, we use a pseudo-random[7] method[8] with current time seed to generate request time τ (reveal time) and destination vertex.

$$\lceil D_v \rceil = \frac{p_v D_c}{\sum_{i=1}^{V}(p_i D_c)} D_c \tag{5.1}$$

$V-$ set of vertices

p_v- probability of demand at vertex v

D_v- count of requests that should appear at vertex $v \in V$

D_c- count of requests

[6]The value is usually computed from historical data of demand at the node; for our demonstration scenarios it can be set manually.

[7]With uniform distribution

[8]Python random method: <https://docs.python.org/3/library/random.html>

Realistic sample of demand was generated based on the real-world based demand model provided by the AIC FEL CTU[9]. Such a sample contains over 100,000 trips with origin and destination. For our testing purposes, we use 1000 trips for training purposes and other 300 trips to run the simulation.

However, the generated demand had to be matched to our nodes in Prague graph. For origin and destination points of the request, we found closest nodes and assign them respectively. If the pair origin and destination points are the same node, we would not use this request in the simulation.

The weights p_v built on the basis of this demand are shown in Figure 5.4. The demand was additionally clustered using DBSCAN and normalized (more in Section 4.2.2). The reason for that is the graph represents a real road network. Thus, junctions consist of multiple vertices and unclustered weights at close vertices may vary too much.

Figure 5.4: Prague graph with demand represented by the size of blue dots, the bigger it is the higher the demand is.

[9] <http://sum.fel.cvut.cz/>

5.2 Experimental scenarios

Scenarios and criteria will provide crucial metrics and guidance for the algorithm and help us to compare each solution.

Criteria are the following:

- Pick-up delay - this is the time between the occurrence of demand d and pick-up time. This is one of the two leading performance indicators, and the main goal of the proposed solution was to minimize the time.
- On-board travel time - this is the second main quality of service measurement, and it also describes the behavior of the agents. This number is affected by setting limit $\zeta = 2$. Thus, it cannot be bigger than $2 \times \eta$ (travel time of the shortest path for each request).
- Prolongation - this describes the relationship between on-board travel time and average travel time of the shortest path for requests. Thus, it provides more in-depth insight into the behavior of agents, such as the length of detour.

Scenarios are test-cases on which we demonstrate the functionality of the proposed solution. Scenarios allow us to compare combinations of criteria, strategies, and algorithmic approaches across each implementation and allow us to look inside the algorithm. Each scenario is managed by the dispatcher that computes agent-request matching and also provides the agent's drive plan (path planning). The simulation runs for a longer time than the epoch's duration is.

Bellow are concluded scenarios:

1. Simple comparison between shortest routing with the insertion heuristics and the proposed weighted routing with the insertion heuristics. This is a proof of concept test case and thus it is achieved on a simple graph, as shown at Figure 5.2 with one agent and two manually entered requests (demo demand). Epoch time was set to 30 minutes and simulation time to 60 minutes. The $\zeta = 2 \times \eta$ and $\lambda = 0$. For better overview, we

added Shortest path planner with simple request-agent matching, this acts as a basic taxi service with no ridesharing allowed.

Results are shown in Table 6.1.

2. Comparison on Prague road graph between shortest routing with the insertion heuristics and the proposed weighted routing with the insertion heuristics. We also tested an update of probability by re-running of Weighted routing with insertion heuristic (IH). In all cases it was concluded with two agents, 20 synthetic requests, that were generated pursuant to weights on graph[10], and initial probabilities at vertices that were generated randomly on the interval of $\langle 0, 1 \rangle$. The $\zeta = 2 \times \eta$, $\lambda = 0$ and epoch time was 60 minutes with simulation lasting 2 hours.

3. Complex comparison between shortest routing with the insertion heuristics and the proposed weighted routing with the insertion heuristics. This is a case that is run on simplified Prague map. The total number of requests was 300, and they were based on a real sample (more in Section 4.2.2 - Real sample).

   In order to run the simulation, we limited, compared to other scenarios, the time to assign the request to an agent to 900 seconds, and after this limit, the request was rejected. The epoch time was set to 2 hours, and the simulation time was set to 4 hours. This helps us to analyze the behavior of agents with assigned requests.

   We ran multiple cases to compare the behavior of two different algorithms, Shortest path planner with insertion heuristic (IH) and Weighted path planner with insertion heuristic (IH), under different variables. We focused on a number of agents in the system (30, 25, 20, 15, 10) while the number of requests remained the same. The $\zeta = 2 \times \eta$ and the probabilities at each vertex p_v were pre-learned on sample of 1000 requests. Additionally, for each combination of passengers' numbers and algorithms, we watched the impact of λ. This takes into account the pick-up time when finding the best agent-request pair, and we labeled this as "p" in results in the Table 6.3.

Before running scenarios, we completed a test ran on the following baselines to check the

[10]For the first instance, those are generated with Python's pseudo-random function with uniform distribution, for the seed we used current time.

functionality of the agent simulation and the implemented algorithms:

1. The shortest routing and simple matching - We use this as our baseline to prove that the simulation works correctly and show how a taxi provider operates without a ridesharing.

2. The shortest routing and insertion heuristics - We use A^* focused on the shortest paths and provide insertion heuristics. This shows us how a normal ridesharing service would operate. Additionally, it proves that the insertion heuristic works correctly.

Figure 5.5: Agent routing using shortest path planner insertion heuristic on synthetic demand.

Figure 5.6: Agent routing using weighted path planner and insertion heuristic on synthetic demand. Because of closer routing to the the requests (orange), it became suitable to pick some of them by the agent.

Chapter 6

Summary

This section is dedicated to the discussion of the results, their impact on future work, and an overall summary.

6.1 Results and comparison

Below are results for proposed test scenarios in Section 5.2.

1. Simple comparison between shortest routing with the insertion heuristics and the proposed weighted routing with the insertion heuristics. Results are shown in Table 6.1.

	Name	Avg. pick-up delay [min]	Avg. on-board time [%]	Prolongation [%]
0	Shortest simple	19.216	27.550000	0.000000
1	Shortest IH	19.216	27.550000	0.000000
2	Weighted IH	2.333	46.566667	23.083700

Table 6.1: Comparison between the Shortest routing with simple agent-matching, Shortest routing with IH and the Weighted routing with IH on simple graph

2. Comparison on Prague road graph between shortest routing with the insertion heuristics and the proposed weighted routing with the insertion heuristics. Results are shown in Table 6.3.

	Name	Avg. pick-up delay [min]	Avg. on-board time [%]	Prolongation [%]
0	Shortest IH	26.042	4.537500	13.240914
1	Weighted IH	20.583	4.720791	17.815248
2	Weighted IH - 2nd epoch	20.018	4.918555	20.939826

Table 6.2: Comparison between the Shortest routing with IH, the Weighted routing with IH, and its second epoch with updated weights on Prague graph. Number of requests was 20, and only two agents were available.

3. Complex comparison between shortest routing with the insertion heuristics and the proposed weighted routing with the insertion heuristics. Results are shown in Table 6.3.

	Name	Avg. pick-up delay [min]	Avg. on-board time [%]	Prolongation [%]
0	30S	11.345511	4.656222	10.752396
1	30W	8.416471	5.586825	27.668088
2	30Sp	3.345691	4.807096	10.123837
3	30Wp	2.860016	5.673841	27.532240
4	25S	14.141808	4.676280	11.589169
5	25W	11.410177	5.508276	27.602298
6	25Sp	3.255690	5.017596	15.960011
7	25Wp	3.176217	5.657179	27.189570
8	20S	18.304834	4.706233	12.515064
9	20W	18.611343	5.508052	27.776565
10	20Sp	4.440944	5.105572	17.939508
11	20Wp	3.941194	5.524310	27.105907
12	15S	32.710053	4.905922	14.350867
13	15W	34.789851	5.436693	28.130464
14	15Sp	5.950302	5.387031	24.881334
15	15Wp	5.265629	5.731985	31.293716
16	10S	69.179247	4.934068	15.762521
17	10W	67.857447	5.562501	27.970217
18	10Sp	12.049666	5.877388	35.936059
19	10Wp	9.075326	5.917524	33.945516

Table 6.3: Comparison between proposed approaches with different variables on Prague road network. Number denotes the count of agents in simulation, S stands for Shortest planner, W stands for Weighted planner, and p stand for baseline that takes a λ (pick-up delay) into account, all cases have $\zeta = 2 \times \eta$. The total number of requests was 300, and the epoch time was 2 hours. Requests samples are based on real-life demand model.

Based on the results the proposed situation-aware A^* planner achieved an overall improvement in average pick-up delay. However, in some situations, the pick-up delay worsened, and it was in a situation when the ration between requests/agent/hour was between 7.5 to 10. Otherwise, the pick-up times improved on the interval of 15% to 25%. The downside is the average prolongation for passengers on board, which on average, increased up to 27% (peak 34%) while the insertion heuristic with only shortest path planner achieved prolongation around 12% with peak 36%. The repercussion of focusing both on pick-up delay (λ) and travel times led to lower pick-up delays when using insertion heuristic for agent-request matching.

The impact of proposed learning (Section 4.2.2) is not significant. The reason for that is the demand has not changed a lot. Therefore, it led just to small adjustments of weights at vertices.

6.2 Conclusion

We have designed, implemented, and successfully tested an algorithm that focuses on demand-aware routing, and we have achieved overall improvements in pick-up times. However, improved pick-up times are redeemed by increased on-board time. The main goal of the proposed algorithm is to route the agent to vertices, where is a higher chance of picking-up a request.

The proposed algorithm does not take occupancy of an agent into account. Therefore, the proposed situation-aware A^* planner continues to route the agent as it still can pick up more passengers. Thus, it increases on-board time for all requests that are served. It can be dealt by using the Shortest path planner in such cases.

It also does not take into account the lowering probability of demand at some vertex by the end of the epoch. As well as, it does not take into account the positions of other agents, even though the dispatcher is aware of their positions and is able to route them effectively.

Aside from the primary implementation of the algorithm, we have implemented a testing environment that would enable us more profound research on demand-aware routing while focusing on context, such as the position of agents relative to each other. This testing

framework can be used as a Q-function for neural networks in future work, and they can influence the routing by adjusting weights at vertices. The environment does not take into account a limited space on roads/edges and at vertices. In order to observe stable demand, we have introduced epochs that can be imagined like morning/evening rush hours and other parts of the day.

Scalability As we stated, the solution is divided into epochs. Thus it is possible to scale it from repeating a few hours to multiple days or weeks. Such an approach enables learning of specific demand for workdays, weekends, and holidays. In the case of a larger number of requests, we have allowed rejecting requests.

It is also important to draw attention to ratio request/agent/hour, because it leads to increased number of points in agent's plan. Therefore, it increases computation time because the insertion heuristic is $O(A \times (2n)^2)$, where n number of assigned requests to agent a and A is the total number of agents. Additionally, this leads to increased reject rate. Based on results of scenarios we found out that the breaking point is ration equal to 10 requests/agent/hour.

To enable a faster computation of paths, we have enabled caching of already calculated directed paths (origin-destination). This can be used together with the Shortest path planner and with the Weighted path planner. However, it cannot be used with the Weighted path planner in case of changing weights during the epoch.

6.3 Future work

In future work, we would like to improve some of the downsides of the proposed solution. Mainly it would be an implementation of switch between the shortest path and weighted path planner based on the situation of the agent. Secondly, we would like to test the proposed approach with a more austere environment, such as limited capacities of roads and places, in the AgentPolis event-based simulator[1].

[1] <https://github.com/aicenter/agentpolis>

Thirdly, we would like to incorporate a direction demand-aware routing that takes into account the direction of requests and agents. For example, more requests are headed to the city center than the other way. Thus, it seems to be beneficial for agents already with passengers on-board traveling in a specific direction to know the direction of possible or new requests. This would lead to better decisions on whether to detour to a specific point or not.

In order to improve routing, that would lead to both a decreased number of pick-up delay and prolongation on-board we want to add a Q-learning strategy (Section B.2). Such method will be used together with demand probabilities at vertices and with output from the environment simulation (pick-up delay, prolongation) to adjust them. Additionally, we would like to add the Attention recurrent neural network (Section B.2) that would allow us to use context from both agents (their position and direction) and requests (direction).

Bibliography

[1] ALABBASI, A. – GHOSH, A. – AGGARWAL, V. DeepPool: Distributed Model-free Algorithm for Ride-sharing using Deep Reinforcement Learning. *CoRR*. 2019, abs/1903.03882.

[2] APPLEGATE, D. L. *The traveling salesman problem a computational study*. Princeton University Press, 2006.

[3] BECKER, M. Agent-based and Discrete Event Simulation of Autonomous Logistic Processes. *Unknown*. 2006, s. 566–571.

[4] BELLMAN, R. E. *Dynamic programming*. Dover, 2015.

[5] BURROUGH, P. A. – MCDONNELL, R. – LLOYD, C. D. *Principles of Geographical Information Systems:3rd Revised: edition*. OXFORD University Press, 2013.

[6] CHEN, Q. *PRIMA 2015: principles and practice of multi-agent systems 18th International Conference, Bertinoro, Italy, October 26-30, 2015: proceedings*. Springer, 2015.

[7] CORDEAU, J.-F. – LAPORTE, G. The dial-a-ride problem (DARP): Models and algorithms. *Annals OR*. 06 2007, 153, s. 29–46. doi: 10.1007/s10479-007-0170-8.

[8] DEMEL, J. *Graphs and their application (CZ)*. J. Demel, 2015.

[9] DEMLOVA, M. Logic and graphs. Electronic script, 5 2015.

[10] FIEDLER, D. et al. The Impact of Ridesharing in Mobility-on-Demand Systems: Simulation Case Study in Prague. *CoRR*. 2018, abs/1807.03352. Dostupné z: <http://arxiv.org/abs/1807.03352>.

[11] GOODFELLOW, I. – BENGIO, Y. – COURVILLE, A. *Deep Learning.* MIT Press, 2016. <http://www.deeplearningbook.org>.

[12] JAKOB, M. et al. AgentPolis: towards a platform for fully agent-based modeling of multi-modal transportation (demonstration), 01 2012.

[13] KLEINER, A. – NEBEL, B. – ZIPARO, V. A Mechanism for Dynamic Ride Sharing based on Parallel Auctions. In *22th International Joint Conference on Artificial Intelligence (IJCAI)*, s. 266–272, 2011.

[14] KOOL, W. – HOOF, H. – WELLING, M. Attention, Learn to Solve Routing Problems! In *International Conference on Learning Representations*, 2019. Dostupné z: <https://openreview.net/forum?id=ByxBFsRqYm>.

[15] OLAH, C. – CARTER, S. Attention and Augmented Recurrent Neural Networks, Sep 2016. Dostupné z: <https://distill.pub/2016/augmented-rnns/>.

[16] PEARL, J. *Heuristics: intelligent search strategies for computer problem solving.* Addison-Wesley Pub. Co., 1985.

[17] QIULIN LIN, M. C. X. L. W. X. A Probabilistic Approach for Demand-Aware Ride-Sharing Optimization. *ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc '19), July 2–5, 2019, Catania, Italy. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3323679.3326512.* 2019. Dostupné z: <https://arxiv.org/pdf/1905.00084.pdf>.

[18] RAYLE, S. S. N. C. D. D. L. – CERVERO., R. App-Based, On-Demand Ride Services: Comparing Taxi and Ridesourcing Trips and User Characteristics in San Francisco, Nov 2014. Dostupné z: <https://www.its.dot.gov/itspac/Dec2014/RidesourcingWhitePaper_Nov2014.pdf>.

[19] SAVELSBERGH, M. W. P. – SOL, M. The General Pickup and Delivery Problem. *Transportation Science.* 1995, 29, 1, s. 17–29. ISSN 00411655, 15265447. Dostupné z: <http://www.jstor.org/stable/25768666>.

[20] SUTTON, R. S. – BARTO, A. G. *Reinforcement learning: an introduction*. The MIT Press., 2018.

[21] XIAOHUI BEI, S. Z. Algorithms for Trip-Vehicle Assignment in Ride-Sharing, 2018. Dostupné z: <https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16583>.

[22] ZHAO et al. Ridesharing Problem with Flexible Pickup and Delivery Locations for App-Based Transportation Service: Mathematical Modeling and Decomposition Methods, Jul 2018. Dostupné z: <https://www.hindawi.com/journals/jat/2018/6430950/>.

Appendix A

Content of attached CD

```
BP
|-- ridesharing/                    - Main directory of our testing
|   |                                  framework
|   |-- experiments/                - Concluded experiments
|   |-- experimetspreparation/      - Preparation of experiments and graphs
|   |-- extools/                    - Modified external tools
|   |-- heuristics/
|   |-- models/                     - Data structures
|   |-- pathcomputation/            - Path finders
|   |-- solvers/                    - Request-vehicle matching
|   \-- tools/                      - Input/output and formatting tools
|-- roadmaptools/                   - External library for graph processing
\-- texts/                          - Source files for the thesis, mainly
                                      TeX files and images
```

Appendix B

Theory background

This part is dedicated to a very brief overview of commonly used and potentially suitable techniques. This provided us with the knowledge that we had to keep in mind when designing the solution.

B.1 Basics of algorithms and theirs heuristics

The typical heuristic approach is based on the solution of a relaxed problem that is technically easier (lower complexity) to solve. A great example would be the A^* algorithm that is used for shortest path planning. Its heuristic function is based on the estimation of the distance to a target node. Heuristics are divided into:

A heuristic function for path finding is admissible if the following statement is true: If we are looking for optimal solution, then heuristic $h(u)$ estimates cost $h^*(u)$ of best possible solution to node u, and $h(u) <= h^*(u)$. In other words, the cost cannot be overestimated. [16]

Euclidean distance is mostly a heuristic function that measures distance d using a straight line between vertex v_1 and v_2. In 2D Euclidean space, which road graph is, it is same as the Pythagorean Formula B.1. [5]

$$d(v_1, v_2) = \sqrt{(x_{v_2} - x_{v_1})^2 + (y_{v_2} - y_{v_1})^2} \tag{B.1}$$

A^* Algorithm is an algorithm for pathfinding in graph G without parallel edges. The approach is ranked as an informed search algorithm. Therefore its main goal is to expand the minimum of vertexes that will not be a part of the final solution. To rank each of the vertexes, we use the heuristic function.

$$f(x) = g(x) + h(x) \tag{B.2}$$

Where $x \in V$, $g(x)$ is cost of current traveled path and $h(x)$ is heuristic function that provides probable cost of remain path. The function mainly affects the behavior and speed of the algorithm. If we put $h(x) = 0$, then it will behave as Dijkstra algorithm. [8]

B.2 Basics of neural networks

A neural network is based on a collection of connected neurons, these connections are called edges and usually contain a weight that is adjusted as learning proceeds. The main components are neurons, connections (with weights and biases), propagation function and learning function/rule. Supervised machine learning is a way where the model is trained in several phases:

- Construction - focuses on adding enough layers and nodes of the network.
- Training - in this stage the model is trained using labeled training data, where the output is adjusted according to the expectations. Epochs indicate how many times the model is trained. For training an approach called backpropagation is used.
- Testing - is an evaluation of the training phase, it is necessary to have another set of labeled data to compare the output with expectations.
- Evaluation - is when a pre-trained model is used to evaluate new data, that are not labeled.

This method is commonly used for pattern recognition (classification) and regression. Unsupervised learning is commonly used for a general estimation. In this approach, the network has to learn without help and find hidden patterns in raw data with help of cost function (i.e. k-means, etc.). A more in-depth view can be found in [11].

Convolution neural network is a special architecture of the neural network that uses series of convolutions, nonlinear pooling layers to classify an image. The convolution layer acts as a filter and is always applied as first, this ensures extracting important data such as vertical structures, etc. in the image. This layer produces a smaller matrix that is the input matrix. The nonlinear layer is an activation function and understands the images and videos. It is done by classification of the picture. [11]

Recurrent neural network has the ability to work with sequences of data like text, especially while working on translation or voice recognition. It understands the language and it can translate. [20]

There are 4 main directions for recurrent networks: neural Turing machines, attentional, adaptive computation time, neural programmers. [15]

Attention recurrent neural network is an extension of RNN B.2 and is suitable as an interface with neural network that has repeating output structure. Attention model is described more in the article [15]

The multi-head attention layer is an extension and can provide additional data and message passing between nodes, therefore connecting all nodes in the representation of the road graph and creating a node-wise fully connected layer. This is used in the encoder and attention layer in [14].

Reinforcement learning is specific because data are not given, but generated by interactions with the environment. At each time-step, the agent performs an action, which is then observed by model and valued by a cost function. The goal is to discover a policy or policies that influence the agent's actions. This can be also modeled as a Markov decision process. [20]

Q-Learning strategy is a specific version of reinforced algorithm (B.2) which cost function is called a quality function and is based on the Bellman Equation. The equation allows rewards (reaction to action) from future states to propagate through states and it also allows not to care about initial values of the Q function. The Q-function is also based on the

assumption that each state is in a direct consequence of the previous state and action.[4] The Bellman Equation B.3:

$$Q(s, c) = r(s, c) + \gamma max Q(s', c) \tag{B.3}$$

Where s is state, c is selected action, then $r(s, c)$ is immediate reward received. Q is quality function based on action and state s'. γ is called discount factor, used to control long-term rewards. [4]

B.3 Agent simulation

An agent-based transportation simulator computes interactions of agents in specified time windows and provides insight into the impacts of individual agents to the whole system. Such a system might be really microscale and describe all important interactions (congestion, acceleration, reaction times, and so on). Or it can be closer to macroscale and focus only on import and specified variables, such as travel times, position of agents. The simulator also might provide visualization and feedback on the quality of transportation. [12]

To be able to really simulate the transportation system and make the visualization, it is necessary to discretize path planning, agent moves, and additional actions in time, collectively called events. For that, it is suitable to use practice called Discrete Event Simulation, which splits time into time slots in predetermined steps. Each event is then inserted into the appropriated time slot and executed in this slot. The processing of slots is sequential to preserver continuity of impacts from previous states. [3]

Each simulator is usually composed of:

- Agents - objects that interact with each other.
- Environment - for example - road graph, and time processing.
- Decision-making heuristics - processing of interaction between agents and their movements.

Appendix C

Agent simulation structure

```
ridesharing
    |-- experiments/
    |   |-- demo_graph.py
    |   |-- demo_ih_shortest_astar.py
    |   |-- demo_ih_weighted_astar.py
    |   |-- demo_simple_shortest_astar.py
    |   |-- demo_simple_weighted_astar.py
    |   |-- prg_real_graph.py
    |   |-- prg_real_ih_pickup_shortest_astar.py
    |   |-- prg_real_ih_pickup_weighted_astar.py
    |   |-- prg_real_ih_shortest_astar.py
    |   |-- prg_real_ih_weighted_astar.py
    |   |-- prg_synthetic_graph.py
    |   |-- prg_synthetic_ih_shortest_astar.py
    |   |-- prg_synthetic_ih_weighted_astar.py
    |   |-- prg_tsk_graph.py
    |   |-- stats.py
    |   \-- zbconfig.py
    |-- experimetspreparation/
    |   |-- process_graph.py
    |   |-- process_nodes_weights.py
    |   \-- process_requests.py
    |-- extools/
    |   |-- compute_edge_parameters.py
    |   \-- simplify_graph.py
    |-- heuristics/
    |   \-- basic.py
    |-- models/
    |   |-- agents.py
    |   |-- drive_plan.py
    |   |-- exceptions.py
    |   |-- graph.py
    |   |-- rconfig.py
    |   |-- request.py
    |   |-- roads.py
    |   |-- simulation.py
    |   |-- trip.py
    |   \-- typing.py
    |-- pathcomputation/
    |   |-- path_tools.py
```

```
    |   |-- shortest_astar.py
    |   |-- time_tools.py
    |   \-- weighted_astar.py
    |-- solvers/
    |   |-- insertion_heuristic.py
    |   \-- simple.py
    \-- tools/
        |-- inout.py
        |-- stats.py
        |-- utils.py
        \-- visualizer.py
```