Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Online k-Taxi Problem

Theoretical

Patrick Stäuble
patricst@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Georg Bachmeier, Yuyi Wang
Prof. Dr. Roger Wattenhofer

October 3, 2017

Abstract

In the k-server and the k-taxi problem the distance that the entities cover is the main and sole cost source. However, the time a request is waiting for its execution is very important as well and is excluded from the current cost model. In this thesis we propose a new approach that takes both the driving distance as well as the waiting time into consideration. For this new cost function we will present an algorithm and discuss its performance.

Contents

Chapter 1

Introduction

Imagine a city with vehicles, pedestrians, roads and street corners. A taxi company receives requests from customers who want to drive from one corner of the city to another but, the company has only a finite number of taxis. This describes the task of the original k-taxi problem, which aims at minimizing the total distance the taxis travel. It belongs into the category of online problems.
An online algorithm receives its input in chunks of data, parts of the information needed are not available at the beginning. The algorithm constantly has to react to new data packages and adapt its decisions accordingly. We use competitive analysis to compare the performance of different online algorithms, and focus on how well the algorithms adapt to the input rather than on their runtime. One of these online problems is the famous k-server problem.
In the k-server problem, an online algorithm has to move k servers to satisfy requests. The servers are distributed over the vertices of a graph. As input progresses, requests appear on the vertices of the graph. To satisfy requests an algorithm has to move the servers along the edges to their locations. The goal of the k-server problem is to minimize the sum of distances over all servers. The original k-taxi problem is a generalization of the k-server problem.
In the original k-taxi problem, an algorithm has control over k taxis on the graph instead of k servers and the requests are customers who want to drive from their current vertex to another vertex. As new customer requests appear, an algorithm has to schedule the taxis in such a way that every customer gets delivered to his destination. Similar to the k-server problem, the aim is to minimize the total driving distance of all taxis.
Figure 1.1 shows an instance of the online 2-taxi problem. At the start we can see the 2 taxis marked green on the lower 2 vertices. A customer appears on the middle vertex and wants to drive to the upper left vertex, marked orange. The algorithm now chooses one of the taxis and the taxi picks up the customer. The taxi is now occupied, and is not available for another customer. As soon as the taxi arrives at the destination, the customer request is satisfied and the taxi is free for a new customer.
In the cost model of the original k-taxi problem, a customer could be waiting a long time until his request can be scheduled. We want to propose a new cost

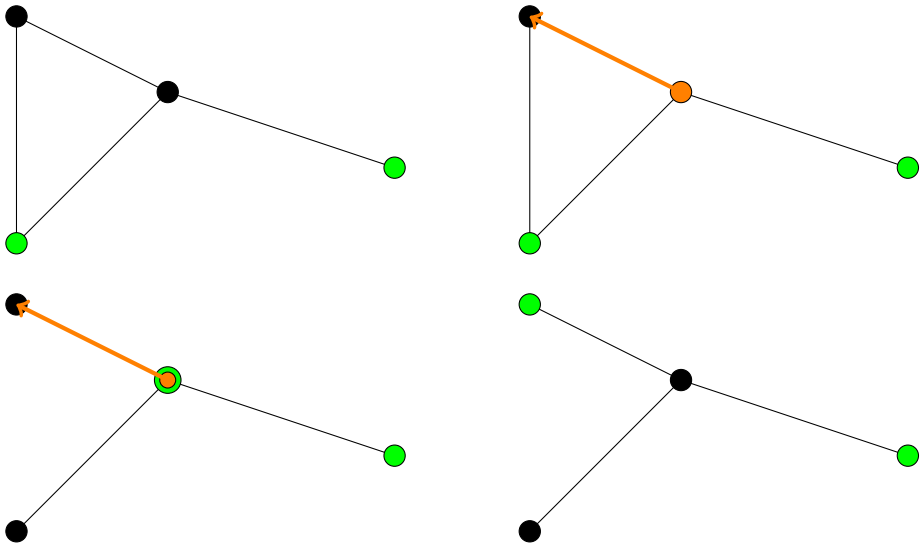

Figure 1.1: Example of a 2-taxi problem

model which punishes algorithms that keep customers waiting. To achieve this, we measure the time all taxis spend driving and the time all customers have to wait instead of calculating the total distance all taxis drive.
We start with the formal description of the problem environment and the cost model that we will use in this thesis. Then we realize that only the subset of lazy online algorithms is relevant for this problem, where "lazy" means that taxis only move while customers are waiting. Then we present an online algorithm that runs optimally on a restricted subset of the problem. Finally, we present results for the general problem.

Chapter 2

Definition of the Problem

2.1 Problem Specification

Definition 2.1. An **instance** of the k-taxi problem is a tuple (G,L,K). G is an undirected, connected and weighted graph with a set of vertices V, a set of edges E and a function w that denotes the weight of an edge:

$$\forall e \in E, \exists w(e) \in \mathbb{R}^+ \tag{2.1}$$

L is a customer request list(see Def. 2.3) and K is a list of vertices that holds the starting positions of the k taxis.

Note that the property connected is just included for simplicity. An instance with a separated graph can be split up into sub-graphs and solved independently. The weight of each edge represents the time a taxi needs to travel from one end to the other. In order to measure the waiting cost any customer might have, we introduce an additional variable.

Definition 2.2. A **time point** $t \in \mathbb{R}_0^+$ represents the time passed after the start of an algorithm's execution.

Definition 2.3. A **customer request** is a tuple $c = (v_{arr}, v_{dest}, t_{arr})$ and the three components are defined as follows:

v_{arr}: The **arriving vertex** is a vertex in G where the customer will appear and where the taxi has to drive to and pick him up.

v_{dest}: The **destination vertex** is a vertex in G which represents the destination the customer wants to drive to.

t_{arr}: The **arrival time** is the time point at which the customer request becomes visible for the algorithm.

The customer request list is the set of all customers, and the cardinality of this set is n. In the offline problem, the complete customer request list is available to the algorithm, but in the online problem only the customer requests that have an arrival time that is smaller or equal to the current time point are visible.

Definition 2.4. A **configuration** is the state of an algorithm's execution on a given instance of the k-taxi problem at a certain time point. A configuration consists of the taxi positions, the information which customer is currently in which taxi and a list of the satisfied customer requests. The configuration is called **initial configuration** if the value of the time point is 0. If an online algorithm is currently in a configuration with time point t, the algorithm cannot reach a configuration with a time point value smaller than t.

If an online algorithm reaches time point t, the algorithm cannot change decisions made at time points smaller than t, and information given at time point t is not available at time points smaller than t. If $e \in E$ is an edge and e connects the vertices $v_1, v_2 \in V$, then $w(e)$ is the time it takes for a taxi to drive from the vertex v_1 to v_2 or from the vertex v_2 to v_1. The problem starts at time point 0. When a taxi drives on e starting at time point t, then it is going to arrive at v_2 at time point $t + w(e)$.
When executed, an algorithm can control up to k taxis. Every taxi is either on a vertex or driving on an edge. A taxi is not able to turn on an edge and go back to the vertex it started from. It has to drive to the end of the edge and there it is able to turn around if needed.

Definition 2.5. If in the current configuration a taxi carries no customer, it is called **free**. If the taxi carries a customer, it is called **occupied**.

An algorithm executed on an instance of the k-taxi problem can execute the following commands in any configuration:

Pickup(Taxi): If the taxi from the argument is located on a vertex as well as a customer this taxi is then occupied by this customer.

Drop(Taxi): If the taxi is occupied and if the customer's destination is the current location of the taxi, the taxi has delivered the customer and is now free.

Move(Taxi,Vertex): This sets a destination vertex for the taxi. The taxi now moves to this vertex, while taking the most efficient path. This command can be used even when the taxi is moving. In that case the taxi just drives to the new destination vertex.

If a taxi does not have active commands, it does not move and stays on its vertex. All these commands, as well as any other calculation the algorithm might perform, are not a part of the cost model.

Definition 2.6. The **costs** of an online algorithm execution on an instance of the online k-taxi problem is defined as the sum of the driving time over all taxis and the sum of the waiting times over all customers. The waiting time of a single customer request is the difference between the time point the customer is picked up by the taxi and t_{arr} of that customer request.

$$cost := \sum_{i=1}^{k} t_{driving} + \sum_{j=1}^{n} t_{waiting} \tag{2.2}$$

To compare different online algorithms, we use the concept of competitive ratio. The competitive ratio compares the cost incurred by a given online algorithm to the optimal offline cost. An offline algorithm has the complete information about the incoming customer requests and, with that, it can compute an optimal solution. The competitive ratio of an online algorithm is the maximal ratio between the costs of that online algorithm executed on a given instance and the cost of an optimal solution of that instance. Note that this ratio has to be at least 1.

Definition 2.7. An online algorithm is called an **optimal online algorithm** for the k-taxi problem, if the competitive ratio of all online algorithms is equal or higher.

2.2 Related Work

Sleator and Tarjan[1] suggested to compare online algorithms to an offline algorithm. This was the beginning of a systematic analysis of online problems. One of the basic problems in competitive analysis is the k-server problem, which was proposed by Manasse[2]. Another important problem is the paging problem[1], where we have a fixed number of fast-memory pages to hold data, and we have to decide in an online fashion which pages to keep there.
The k-taxi problem was originally proposed by Xu and Wang[3], and some important results have already been shown. Additionally a lot of papers suggested variations of the k-taxi problem. The k-taxi problem on a real line[4] restricts the graph to one line, pictured as a very busy main street. The weighted k-taxi problem[5] is based on the weighted k-server problem, where each server has an additional weight. In the weighted k-taxi problem, the taxis have different costs for moving.

The k-truck[6] is a different generalization of the k-server problem and originally presented by Ma,Xu,Wang. In this problem trucks move on a graph, and similar to the k-taxi problem, deliver load from point a to b. Additionally, the weights of the loads are different and have an impact on the cost function. Many other variations of the k-server problem can be found in the literature (for an overview, see [7]).

Chapter 3

Laziness and Probabilistic Algorithms

3.1 The Instability Issue

Example 3.1 shows how unstable the instances of the online k-taxi problem can be. A small addition to the customer request list can change the complete optimal solution.

Example 3.1. Figure 3.1 displays two instances of the k-taxi problem. Again the taxis are marked in a green tone and the customers in orange. We additionally marked the first step of the optimal solution with a green arrow. All the edges to the top node of the graph have high cost to make sure the taxis do not drive unnecessarily through the top vertex. The main difference between the two instances is the customer request appearing at the right node. In the first instance the best possible solution is to assign every taxi with the customer that appears at its left node. In this instance the left most taxi is not assigned to any customer, because there is one more customer than taxis in this first instance. If we include the customer on the right end of the graph, as in the second graph, we end up with a completely different assignment. In this instance, the taxis deliver the customers that are to their right, because this assignment has lower costs than just assigning the taxi on the left with the customer on the right.

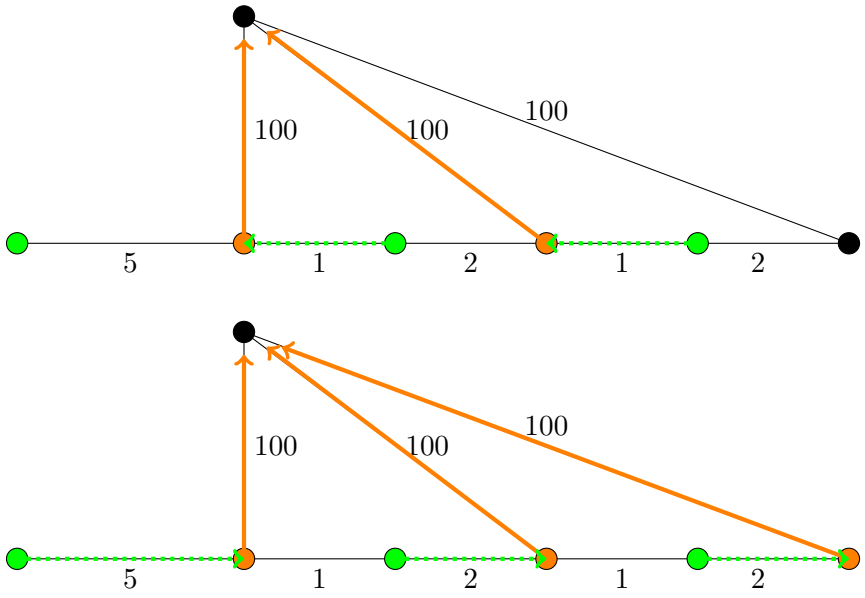

Figure 3.1: The unstable 3-taxi problem.

Note that this graph can be extended for much larger graphs with more taxis and customers. For each additional customer request over a new 100 cost edge we add a node for an additional taxi and connect with edges accordingly.

3.2 Lower Bound

In this thesis we only consider lazy algorithms for the online problem.

Definition 3.2. Lazy algorithms are algorithms that only move taxis while there are active requests.

Theorem 3.3. *Lazy algorithms have a competitive ratio of at least 2.*

Proof. We prove Theorem 3.3 by constructing a problem instance that shows a competitive ratio of at least 2 for all lazy algorithms. In Figure 3.2 the single customer request marked orange appears at time point 1, therefore in the offline solution it is possible to move to the customer with the taxi marked green in time and deliver him right away.

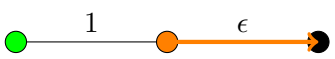

Figure 3.2: Lower Bound for all online algorithms; $0 < \epsilon \ll 1$.

Because we only consider lazy algorithms, all online algorithms will not move their taxis until time point 1. The best an online algorithm ALG can do is to pick up the customer with the taxi.

$$competitiveratio \geq \frac{t_{driving_{ALG}} + t_{waiting_{ALG}}}{t_{driving_{OPT}} + t_{waiting_{OPT}}} = \frac{(1+\epsilon)+1}{(1+\epsilon)+0} \tag{3.1}$$

□

Looking at the proof of Theorem 3.2 one could argue that an algorithm that moves the taxi to the middle of the graph and violating the lazy restriction can achieve better results.

Theorem 3.4. *An online algorithm either controls the taxis like a lazy online algorithm or the online algorithm has a competitive ratio of at least 4.*

Proof. We observe the behavior of the taxi in Figure 3.2 until the first time point. An online algorithm either moves the taxi to the middle vertex or not. If the taxi does not move until the customer request appears, then the behavior of the online algorithm is lazy. If the online algorithm moves the taxi, we change the customer request list that the customer request arrives at the left vertex. Figure 3.3 shows the graph at the time point the taxi arrives at the middle vertex and the customer request appears.

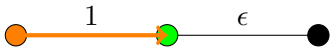

Figure 3.3: Adapted version of Figure 3.2; $0 < \epsilon \ll 1$.

$$competitiveratio \geq \frac{t_{driving_{ALG}} + t_{waiting_{ALG}}}{t_{driving_{OPT}} + t_{waiting_{OPT}}} = \frac{3+1}{1+0} \tag{3.2}$$

□

A different approach to achieve a better lower bound could be to make the algorithm probabilistic. Instead of the decisions in the previous examples, in this chapter, the algorithm stays or moves according to a probability distribution.

Theorem 3.5. *The probability, that a probabilistic algorithm has lower costs than a lazy algorithm, is arbitrarily low.*

Proof. To show Theorem 3.5 we will construct a class of instances with decreasing probability of success for a probabilistic algorithm. Figure 3.2 is the first instance

of the class and Figure 3.4 shows the fifth instance of the class. We just add another graph piece to the existing one and end up with a higher instance.

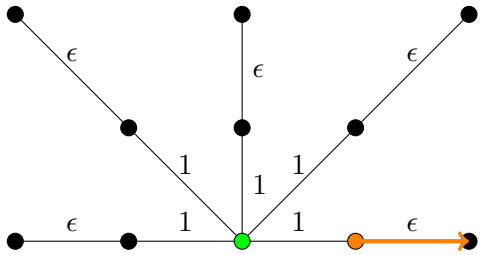

Figure 3.4: Adapted version for probabilistic algorithms; $0 < \epsilon \ll 1$.

If there are more options to move the taxi at time point 0, then the probability that the taxi is moved to the correct vertex is smaller. □

Chapter 4

Results

In this chapter we first take a closer look at a slightly adapted version of the problem model and present an online algorithm.

4.1 Model

We look at a restricted case by adapting the model in the following way. The graph is new completely connected and every edge has the same weight:

Definition 4.1. An instance of the **unit distance** k-taxi problem is a tuple (G,L,K). G is a undirected, connected and weighted graph with a set of vertices V, a set of edges E and the function w that denotes the weight of an edge. The following holds for all graphs:

$$\forall v_1, v_2 \in V : \{v_1, v_2\} =: e \in E, w(e) = 1 \tag{4.1}$$

L is a customer request list and K is a list of vertices that holds the starting positions of the k taxis.

In Figure 4.1 we show an example of this restricted k-taxi problem with 5 vertices.

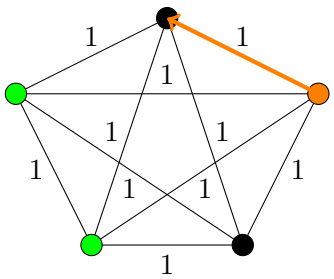

Figure 4.1: A unit distance 2-taxi problem with 5 nodes and 1 customer.

4.2 A First Approach

Let us consider the following first-in-first-out or short FIFO algorithm:

```
Input: Graph G and the initial configuration of the taxis K as well
       as the list L of customer requests.
Output: Commands where the taxis should drive.
while True do
    for Taxi t do
        if t.isFree() then
            Choose remaining customer c with smallest arrival time,
             break ties arbitrarily
            move(t,c.v_arr)
            pickup(t)
            move(t,c.v_dest)
            drop(t)
        end
    end
end
```

Algorithm 1: Pseudo-code of FIFO Algorithm

The above described FIFO algorithm seems to be a good first approach, but Theorem 4.2 shows that the lower bound of the competitive ratio grows with the number of customers.

Theorem 4.2. *The FIFO algorithm executed on any instance of the unit distance k-taxi problem has at least a competitive ratio of* $\frac{n}{2}$.

Proof. We prove Theorem 4.2 by constructing an example and show that the FIFO algorithm has a competitive ration of $\frac{n}{2}$:
The graph shown in Figure 4.2 has four nodes, two of them are painted blue and the other two are painted red. One of the taxis starts at the first blue node B1 and one taxi at the first red node R1, both are marked green.
The adversary places the taxis such that the optimal solution can pick up the customers on the blue nodes with one taxi and the customers on the red nodes with the other. In contrast the FIFO algorithm has to switch taxis between the blue nodes and the red nodes. We construct the following family of instances: $\forall n = 4 * i$, $i \in N$ we have four groups of customers:

- (A) Customer# $1 + i * 4$: Arrives at B2 and wants to drive to B1. Appears at time stamp $i * 2 + 1$.
- (B) Customer# $2 + i * 4$: Arrives at R2 and wants to drive to R1. Appears at time stamp $i * 2 + 1 + \epsilon$.

- (C) Customer# $3+i*4$: Arrives at R1 and wants to drive to R2. Appears at time stamp $i*2+2$.
- (D) Customer# $4+i*4$: Arrives at B1 and wants to drive to B2. Appears at time stamp $i*2+2+\epsilon$.

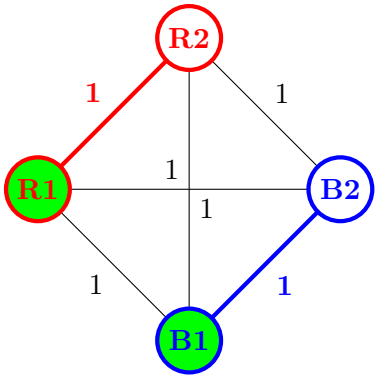

Figure 4.2: Competitive ratio lower bound for the FIFO algorithm.

The optimal path to deliver the customer request groups A to D is to schedule the taxi that starts on B1 with the customers A and D. The taxi picks up A as soon as D is delivered and picks up D after finishing A. The taxi that starts on R1 takes the customers B and C accordingly. This leaves us with $2+n$ driving costs and $n/2*\epsilon$ of waiting costs, where n is the number of customers.
The path of the FIFO algorithm is not nearly as efficient as the optimal solution. For each taxi we just execute the customer who arrives first. This means that the taxi starting on B1 executes customer groups A and C and the taxi starting on R1 executes customer groups B and D. This leads us to a total driving cost of $2*n$. The waiting cost accumulates as more customers appear. The first 2 customers wait 1 time unit, the second 2 customers wait 2 time units and the j'th 2 customers wait j time units. This results in a waiting cost of $\frac{(n*(n+2))}{2}$. All of these costs leave us with a lower bound of the competitive ratio of approximately $\frac{n}{2}$, for a small ϵ. □

4.3 Online Algorithm

Algorithm 2 is the pseudo-code of our algorithm UNITALG. UNITALG is designed to calculate the best possible solution with the available information and controls the taxis according to this solution. If UNITALG receives additional customer requests, the solution is recalculated and the taxis receive new destinations.

Input: Graph G and the initial configuration of the taxis K as well as the list L of customer requests.
Output: Commands where the taxis should drive.
while *True* **do**
  if *new customer requests are visible to UNITALG or a taxi finishes a customer request* **then**
    solution = calculateBestSolution(input)
    `// In the following we execute the calculated solution:`
    for *Taxi t* **do**
      move(t,solution[t].v_{arr})
      pickup(t)
      move(t,solution[t].v_{dest})
      drop(t)
    end
  end
end

Algorithm 2: Pseudo-code of UNITALG

UNITALG is designed in a way that every time the customer request list L changes, which means that there is a new customer request visible to UNITALG, or if any taxi finishes a customer request the algorithm changes the destination of each taxi. Whenever one of these conditions are true the algorithm executes the following steps:

- In the first step the algorithm calculates the best result from the input that is currently available. The best result is the most cost efficient and optimal solution without considering future requests. If there are multiple optimal solutions the algorithm outputs the first calculated.
- In the second step the algorithm executes the calculated solution until the input changes such that the execution order could change. This is only possible, if a new customer appears, so we execute until a new customer request arrives.

Especially the calculation of the best solution with the current input is very inefficient with a large number of customers and taxis, but in this thesis we focus on the cost efficiency of the solution and not on runtime.

4.4 Competitive Ratio Upper Bound Proof for UNITALG

Theorem 4.3. *The UNITALG algorithm executed on any instance of the unit distance k-taxi problem has at most a competitive ratio of 3.*

Proof. The idea of the proof is to reduce the problem space to a simplified one. For these instances we prove the competitive ratio of 3 and finally we generalize back to the unit distance k-taxi problem model.
There exist instances of the unit distance k-taxi problem where the optimal solution has waiting costs. For example there is a graph with 4 nodes. Two customers arrive at 2 different nodes and want to drive to the other 2 nodes. The 2 customers arrive both at time point 0. This input can't be executed by one taxi without generating waiting costs.
In this proof, instead of looking at all instances, we first consider only instances where the optimal solution has no waiting costs. Any unit distance problem instance can be transformed into the simplified instance with the following steps:

- First we calculate how the optimal solution of the instance. This optimal solution might have waiting costs.
- In the second step we change the customers arriving times such that the optimal solution has no waiting costs. For every customer that waits, we delay the arriving time t_{arr} to the time point the optimal solution reaches the customer.

The optimal solution does not change, when we delay the arriving of the customer requests, because any other execution would have waiting costs.
Now we show that this reduced input space has an upper bound on the competitive ratio for UNITALG. First we look at the case with only 1 taxi. Note that in all of these cases we are at most one time unit behind the optimal solution.

- The customer appears at the node where the optimal solution already has a taxi: In this case the optimal algorithm has cost 1. UNITALG has different costs depending on the situation:
  - It is the startpoint of the taxi: In this case UNITALG schedules the customer immediately, because there is no other customer, and UNITALG has cost 1.
  - It is the destination vertex of a different customer request: Then UNITALG either has already executed that customer and is at the same vertex and UNITALG has cost 1 or UNITALG is delivering this other customer, therefore UNITALG has at most cost 2, because UNITALG can only be 1 time unit behind the optimal solution.

- The customer appears at a node, where the optimal algorithm has to drive 1 edge to get to the start point. OPT has therefore cost 2. In this case UNITALG has no information about the customer until he appears, so UNITALG does not move the taxi and OPT does in the first time unit. In the second the customer appears and UNITALG schedules the customer. Now UNITALG is 1 time unit behind OPT. Note that it could be that UNITALG has to finish another customer in the first time unit, but this takes at most 1 time unit, since UNITALG can only be 1 time unit behind OPT.

This concludes the case distinction for 1 taxi and shows that the upper bound of 3 holds for the 1 taxi case. But we want to show that this also holds for k taxi. In the k taxi case we assign every UNITALG taxi with an OPT taxi. If there is no difference in the execution we can refer to the simple 1 taxi case and have proven the competitive ratio of 3. But what if one of the UNITALG taxis decides to deliver a customer on the route of an OPT taxi that is not assigned to it. This can only happen in one specific scenario:
2 customer appear at the same time and the optimal algorithm has two OPT taxis standing there, because of the restricted problem. 2 UNITALG taxis are 1 time unit behind the OPT taxis(the ones assigned to them). If they are less than 1 time unit behind OPT the algorithm decides to not switch the taxis, because it takes more cost. In this situation it is possible that taxis can switch assignments, but this is not a problem, since the cost stays in the competitive ratio of 3. This switch of taxis can also be extended to $l \leq k$ taxis that interchange, but here as well the cost is in the boundaries. This proves that UNITALG has competitive ratio at most 3 for the simplified input as well as for the original input.
In order to generalize back to the unit distance problem we change the arriving times back one customer after another. This adds cost to the optimal solution, as well as to the solution of UNITALG. If a customer request new arrives earlier than a different customer request, it could be possible that the order of the UNITALG solution changes. This adds additional costs for each customer that follows. In the unit distance problem model this can be at most 3, 1 for driving to the customer, 1 for delivering the customer and 1 until UNITALG is back at the arriving vertex of the next customer. So for each customer a customer request skips, because of the generalization, the optimal solution pays cost of at least 1 and UNITALG of at most 3. This does not effect the competitive ratio and the upper bound also holds for the unit distance problem. □

4.5 Competitive Ratio Lower Bound Proof for UNITALG

Theorem 4.4. *The UNITALG algorithm executed on any instance of the unit distance k-taxi problem has at least a competitive ratio of 3.*

Proof. In order to prove Theorem 4.4 we will construct an instance of the unit distance problem and show that the optimal solution is more cost efficient than the UNITALG solution by a factor of 3.
Given graph G with k taxis and k + 2 nodes $\{v_1, v_2, v_3, ..., v_{k+2}\}$ that is fully connected with unit distance $d = 1$. The initial configuration K for the taxis is $[v_1, v_2, v_3, ..., v_k]$. We now define the customer request list L such that:

- The first customer request arrives at time point $t_1 = 1$ on vertex v_{k+1} with destination vertex v_{k+2}. Now UNITALG calculates the best possible solution to pick up that customer. There are k possible solutions, because the graph is unit distance every taxi can pick up the customer. W.l.o.g., we assume that the taxi on v_1 processes this customer.
- The second customer request arrives at time point $t_2 = 3$, just after UNITALG delivered the first customer request. This customer request arrives at v_1 and wants to drive to the arriving point of the first customer request v_{k+1}. UNITALG again calculates the best possible solution and ends up with k possible ones. Again w.l.o.g., we assume the taxi at v_2 picks up the second customer.
- For $i \in \{3, ..., k\}$, customer i arrives at v_{i-1} with destination v_i at time $t_i = 2 * i - 1$

Before we take a look at the costs we show how an optimal solution OPT moves the taxis:

- In the first time unit OPT moves the taxi that is touched last by UNITALG, which would be v_k to the vertex v_{k+1}.
- At the time this first taxi arrives at node v_{k+1} the customer request appears and can directly be delivered. However, UNITALG has to move the taxi from v_1.
- At the time point t_2 arrives a customer request at v_1 and for UNTIALG there is no taxi there, but OPT has moved a different taxi for the first customer request. So in OPT the taxi just delivers the customer right away.
- At time point t_i the customer request at v_{i-1} arrives and OPT delivers the customer with the taxi on vertex v_{i-1}

After that last customer request, the taxi positions of the solution of UNITALG and OPT are equal. We can also see that the above described solution is optimal. Every customer request is delivered immediately without any waiting and there is only the additional cost of 1 that is necessary anyway.
We calculate the costs by summing up the cost of each individual customer.

UNITALG: UNITALG has costs of 3 for each customer. UNITALG has to drive to the customer, which takes $d = 1$ and an additional 1 for the waiting time of the customer and finally an additional 1 to deliver the customer. If the algorithm uses every taxi once, we have a total cost of: $k * 3$ for $k > 1$

OPT: OPT has costs of 1 to get to the initial configuration and after that, for all the customers OPT only has 1, no waiting cost and no driving cost to the customer request, resulting in a total cost of: $(k) * 1 + 1$ for $k > 1$.

This results in an approximate competitive ratio of 3 for a large number of taxis k:

$$cr = \frac{c_{driving}(UNITALG) + c_{waiting}(UNITALG)}{c_{driving}(OPT) + c_{waiting}(OPT)} = \frac{3 * k}{1 * k + 1} \tag{4.2}$$

□

4.6 General Competitive Ratio Lower Bound for UNITALG

In this chapter we introduced an adapted version of the problem in Definition 4.1 and proved competitive ratio bounds for UNITALG, but how well performs UNITALG with the original problem(see Definition 2.1)?

Theorem 4.5. *The UNITALG algorithm executed on any instance of the general k-taxi problem has at least a competitive ratio of* $n + 1$.

Proof. We prove Theorem 4.5 by constructing an instance of the general k-Taxi problem according to Definition 2.1 and show that UNITALG generates a solution $n + 1$ time the cost of the optimal solution.

Figure 4.3: Competitive ratio lower bound for UNITALG; $0 < \epsilon \ll 1$.

Again $\epsilon \in \mathbb{R}^+$, taxi painted green and customers painted orange. Also note that we still only look at lazy algorithms that only move when there is a customer

request. The customers arriving times are defined as follows when the leftmost customer is customer 1 and from left to right an increment of 1. For customer i $t_{arr} = 1 + \epsilon * (i - 1)$.
The optimal solution is that in the first time unit the taxi moves from the leftmost node to the first customer request start position. The customer request then all arrive such that the taxi can drive and pick up without generating any waiting costs. The total costs of the optimal solution are $c(OPT) = 1 + \epsilon * n$.
UNITALG waits until the first customer appears and then picks up every customer from left to right. Therefore UNITALG has costs of $c(UNITALG) = (1 + \epsilon * n) + (1 * n)$.
This results in a competitive ratio of $1 + n$ for a large number of customers and an ϵ that goes faster to zero than $1/n$. After this presentation one might argue that the competitive ratio is not dependent on the number of customers, but on the size of the graph. Figure 4.4 is an adapted version of the example on a constant size graph:

Figure 4.4: Competitive ratio lower bound for the FIFO algorithm.

In this graph there are 2 groups of customers:

- The uneven customers ($i = 1, 3, 5, 7...$) arrive at the middle vertex and want to drive to the right vertex.
- The even customers($i = 2, 4, 6, 8...$) arrive at the right vertex and want to drive to the middle node.
- For all customers the following holds: $t_{arr_i} = 1 + \epsilon * (i - 1)$

After the first pickup the offline solution assigns the taxi such that it drives between the middle and the right node. UNITALG does the same, but with a delay of 1. The cost calculation is analogous to the first and has the same result. □

4.7 Adaption of the Cost Model

In definition 2.6 the driving cost of the taxi is weighted the same as the waiting cost. The total cost is the sum of these two costs. In this chapter we add variables to the cost function and show a more general competitive ratio for that function. Take a look at the following definition of the cost function with these 4 new variables.

Definition 4.6. The **costs** of an online algorithm execution on an instance of the online k-taxi problem is defined as follows with $x, y, v, w \in \mathbb{R}$:

$$cost := x(\sum_{i=1}^{k} t_{driving})^v + y(\sum_{j=1}^{n} t_{waiting})^w \tag{4.3}$$

Now we show how this new cost function influences the previous results.

Theorem 4.7. *The competitive ratio upper bound of 3 for the UNITALG algorithm proven in Theorem 4.3 changes with the additions from Definition 4.6 to:*

$$2^v + \frac{yn^w}{xn^v} \tag{4.4}$$

Proof. In order to prove Theorem 4.7 we need an additional variable that we call *awt*. *awt* represents additional waiting cost that appear, when we transfer from the simplified problem to the unit distance problem(see proof of Theorem 4.3). But they are actually irrelevant, because we can upper bound them.

$$\frac{x * (2 * n)^v + y * ((awt + 1) * n)^w}{x * n^v + y * (awt * n)^w} \leqslant 2^v + \frac{yn^w}{xn^v} \tag{4.5}$$

□

Theorem 4.8. *The competitive ratio lower bound of 3 for the UNITALG algorithm proven in Theorem 4.4 changes with the additions from Definition 4.6 to:*

$$2^v + \frac{yn^w}{xn^v} \tag{4.6}$$

Proof. We fill in the values from Theorem 4.4 and end up with the same result as in Theorem 4.7.

$$\frac{x * (2 * n)^v + y * (1 * n)^w}{x * n^v + 0} \Rightarrow 2^v + \frac{yn^w}{xn^v} \tag{4.7}$$

□

Theorem 4.9. *The general competitive ratio lower bound of 3 for the UNITALG algorithm proven in Theorem changes with the additions from Definition 4.6 to:*

$$1 + \frac{yn^w}{x2^v} \tag{4.8}$$

Proof. The values according to the example in the proof from Theorem 4.5.

$$\frac{x * 2^v + y * n^w}{x * 2^v + 0} \Rightarrow 1 + \frac{yn^w}{x2^v} \tag{4.9}$$

□

Chapter 5

Conclusion

In this thesis we introduced a new cost model for the existing k-taxi problem. For this new problem, we developed an algorithm with a constant competitive ratio on a restricted subset. We also showed how the algorithm would perform in the general problem. However, there is no upper bound proof for UNITALG in the general case. It might be possible to adapt the proof structure from Theorem 4.3 to achieve a general upper bound for the competitive ratio for UNITALG. An additional goal could be to observe the effects the new cost model has on the various k-taxi adaption stated in the community.

Bibliography

[1] Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communications of the ACM **28**(2) (feb 1985) 202–208

[2] Manasse, M.S., McGeoch, L.A., Sleator, D.D.: Competitive algorithms for server problems. Journal of Algorithms **11**(2) (jun 1990) 208–230

[3] Xu, Y., Wang, K.: Scheduling for on-line taxi problem and competitive algorithms. Journal of Xi'an Jiao Tong University (1997)

[4] Chun-lin Xin, Wei-min Ma: Scheduling for on-line taxi problem on a real line and competitive algorithms. In: Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826), IEEE 3078–3083

[5] Ma, W., Wang, K.: On the On-Line Weighted k-Taxi Problem. In: Combinatorics, Algorithms, Probabilistic and Experimental Methodologies. Springer Berlin Heidelberg, Berlin, Heidelberg (2007) 152–162

[6] Ma, W., Xu, Y., Wang, K.: On-line k-Truck Problem and Its Competitive Algorithms. Journal of Global Optimization **21**(1) (2001) 15–25

[7] Koutsoupias, E., Taylor, D.S.: The CNN Problem and Other k-Server Variants. Springer, Berlin, Heidelberg (2000) 581–592