UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

THIAGO BELL FELIX DE OLIVEIRA

Applying Bandit Algorithms to the Route Choice Problem

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Prof. Dr. Ana L. C. Bazzan
Coadvisor: Prof. Dr. Bruno Castro da Silva

Porto Alegre
July 2017

"We are a way for the cosmos to know itself."

— Carl Sagan

ACKNOWLEDGMENTS

I would like to thank my advisors for their feedback and contributions that helped in the development of this work. Specially Prof. Dr. Ana Bazzan for the two years of mentorship I had the privilege of having during my bachelor studies. I am also grateful for my colleagues Gabriel de Oliveira Ramos, Jorge Aching Samatelo, Mariana Mendoza and Ricardo Grunitzki for their support during this time, and Arthur Zachow who contributed to the traffic assignment framework used for this work.

The virtue of my education was no less than the result of the effort of my family to provide me with it even if times were not opportune. The support of friends and family during my studies were outstandingly important and it is something which I'm thankful for.

Acknowledgments are also owed to the European Commission and the Technical University of Munich for, respectively, granting me a scholarship for exchange studies and receiving me as a student, and to the CNPq science promotion agency for two year-long undergraduate research scholarships.

ABSTRACT

Traffic infrastructure in major cities must be able to handle increasing demand. Building this infrastructure is expensive and not something that is done in a short time frame. Bottlenecks in the network and potential improvements must be identified for further planning and expansion. However, changes to the network are not always beneficial as shown by the Braess paradox. Therefore, it is ever more important to be able to understand the demand of its users and how it affects the network, and predict the effect changes in the network will cause.

Traffic demand is normally defined in origin destination studies but this information is incomplete. It shows only the endpoints of trips but not the routes they take. To understand how each link in the network is used, a route allocation must be calculated.

The problem of allocation of routes to trips is called the traffic assignment problem. We compare the performance of selected bandit algorithms with that of Q-learning in the context of the traffic assignment problem in a multi-agent reinforcement learning scenario. Specifically, non-stationary bandit algorithms were the main focus due to the non-stationarity of the problem. These algorithms were evaluated in their ability to provide traffic allocations that minimize the average travel time on a traffic network. Through experimentation, we found that bandit algorithms show potential. However, they did not perform better than Q-learning. Further study is required to better ascertain their capabilities for the problem.

Keywords: Reinforcement learning. Q-learning. bandit algorithms. traffic assignment. route choice.

Aplicando Algoritmos de Bandits ao *Route Choice Problem*

RESUMO

Infraestrutura de tráfego em grandes cidades deve ser capaz de lidar com demanda crescente. Construção de infraestrutura tem altos custos e requer longo tempo de construção. Gargalos e melhorias em potencial em redes de tráfego devem ser identificados para planejamento e expansão. No entanto, mudanças na rede não são sempre benéficas como mostrado pelo paradoxo de Braess. Portanto, é cada vez mais importante entender a demanda dos usuários e como ela afeta a rede, e prever os efeitos que mudanças na rede vão causar.

Demanda de tráfego é normalmente definida em estudos de origem e destino mas essa informação é incompleta. Ela mostra apenas o inicio e fim de cada viagem mas não as rotas seguidas. para compreender como cada aresta na rede é usada uma alocação de rotas deve ser calculada.

O problema de alocação de rotas a viagens é chamado de *Traffic Assignment Problem*. Nós comparamos a performance de algoritmos de bandits selecionados com aquela do Q-learning no contexto do *Traffic Assignment Problem* em um cenário multi-agente com aprendizagem por reforço. Especificamente, algoritmos de bandits não estacionários foram o principal foco devido a não-estacionaridade do problema. Esse algoritmos foram avaliados nas suas habilidades de prover alocações de tráfego que minimizem o tempo médio de viagem. Através de experimentação, foi descoberto que os algoritmos de bandits mostram potencial. No entanto, eles não tem performance melhor que a do Q-learning. Mais estudos são necessários para melhor determinar as capacidades desses algoritmos para o problema.

Palavras-chave: Aprendizagem por reforço,Q-learning, algoritmos de bandits,*traffic assignment*, *route choice*.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

TAP	Traffic Assignment Problem
UCB	Upper Confidence Bounds
MSA	Method of Successive Averages
SF	Sioux Falls network

CONTENTS

1 INTRODUCTION

With rising urbanization and the growth of cities to the order of several million inhabitants, road networks suffer from significant pressure. The cost associated with building road infrastructure makes their adequate planning very important. It is a waste of taxpayer money to build infrastructure where it is not needed. Furthermore, changes in the road infrastructure can make what is currently a congested area even worse. An example is building a road which then depicts symptoms of the Braess Paradox (BRAESS, 1968).

Planners must understand how the network is used to know what to build and where to do it. They use surveys to obtain data such as the origin and destinations of trips inside the system (e.g. a person lives in the neighborhood A and commutes to work every day in the neighborhood B). This data, however, is not complete. It says nothing about which route the users take to work but its endpoints.

The traffic assignment problem deals with allocating trips to routes on a network seeking to minimize a cost defined by the instance of the problem (normally travel time is used). It is useful in two cases. The first would be a solution to the problem mentioned above: A realistic allocation could serve as a form of study for planners looking for potential improvements on the network. Another use case is to allocate trips in the real world in a scenario of automated vehicles, for example.

Traditional methods of solving the TAP can be very efficient and solve large instances. The TAP treats traffic allocation from the perspective of a traffic manager trying to understand a network. Its algorithms, therefore, are mostly centralized. The same problem, however, can be seen from the perspective of the driver where they have to choose a route. Notice that with this variant, the problem becomes distributed. Each driver must allocate its route instead of a central authority. We call this the *route choice problem*.

We use reinforcement learning in a distributed multi-agent formulation to solve the route choice problem. On previous work such as (STEFANELLO; SILVA; BAZZAN, 2016), Q-learning is the reinforcement learning algorithm of choice. However, this algorithm is stateful while the route choice problem is stateless. We study how stateless bandit algorithms compare with Q-learning for the problem and how they deal with its non-stationarity.

This work is organized as follows Chapter 2 presents some important information about the TAP and learning methods used. Chapter 3 presents related work; Chapter 4 our proposal for this problem; Chapter 5 experimental results and Chapter 6 the conclusion.

2 THEORETICAL BACKGROUND

Traffic Networks

A road network can be modeled as a graph $G(V, L)$ where V is the set of vertices and L the set of links. The vertices represent points on the network, and links, the roads between those points. In real world networks, every road has a capacity (i.e. a maximum flow of vehicles that can cross it simultaneously). If too many vehicles are on that road, the more congested it is and the longer it will take to cross it according to the cost-flow relationship as described on (ORTÚZAR; WILLUMSEN, 2011). Road models can have costs that represent different information according to the interest of its designer. For example, they can represent the fuel cost of using it or, usually, the travel time between its endpoints.

When using travel time as the cost metric, it can be a fixed value, or it can vary. The advantage of using the latter is the possibility to model congestion. The more vehicles there are on the link, the costlier it will be to cross it. A common example is the volume-delay function. It has a constant component which represents the time to traverse the link under free flow conditions, (i.e. no congestion), and a variable component dependent on the flow. This last component models the delay on the travel time caused by how congested the link is. An example function is shown on Equation 2.1. t_a is the free flow travel time on link a and the factor 0.1 is the increase in travel time caused by each trip allocated to the edge a.

$$c_a(flow) = t_a + 0.1 * flow \tag{2.1}$$

A route on the network is a series of links connecting an origin and a destination. This route has a cost defined as the sum of the costs of each individual links. Traffic planners study the demand of traffic networks through origin-destination surveys. They identify the major demands of a transportation network and represent this information with origin - destination (OD) pairs and their respective demand values. These values specify how many trips are made between each of the origin-destination pairs on the network. Planners use this information for insight into how the road system is used and how to improve it. Figure 2.1 shows the OW network from (ORTÚZAR; WILLUMSEN, 2011). Table 2.1 shows the OD pairs of the network and their associated demand values in the number of trips. We consider the number of trips going through a link as its flow and use it for calculating its cost with a function such as that of Equation 2.1.

Figure 2.1: The OW network.
Origin nodes shown in light blue; destination ones in dark blue.

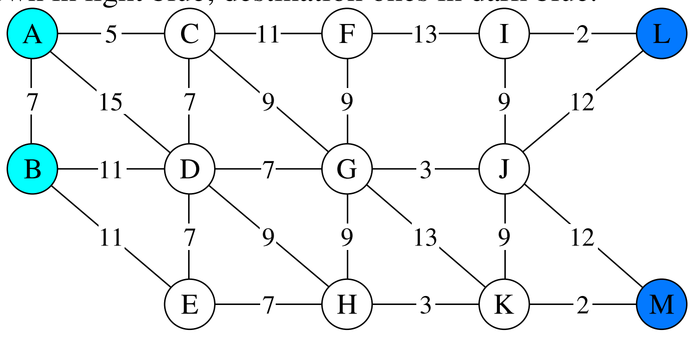


Table 2.1: OD pairs of the OW network

origin	destination	number of trips
A	L	600
A	M	400
B	L	300
B	M	400

However, OD pairs are not enough to accurately understand the network and its use. Traffic assignment algorithms allocate trips to routes on the network. If the cost of the links on the network varies according to the flow, then, assigning trips to routes is not enough. Congestion and its effects on the cost of travel must be taken into account. (WARDROP, 1952) defines what is known as Wardrop's first equilibrium. It states that the distribution of trips across the network reaches equilibrium when no vehicle can change its route and decrease its travel time. It is also known as user equilibrium. It assumes that users are rational and seek to minimize their own travel cost. In this way, users are selfish in that they do not care about their impact on other user's travel time. Reaching the user equilibrium does not guarantee that the average travel cost of the network is the minimum. In contrast to that is social equilibrium also known as system optimum or Wardrop's second equilibrium, defined by the same author, where no change in the route of any trip can decrease the average travel time of the network. This is seen as a more egalitarian allocation because it seeks what is best for society and not any specific individual. However, in this scenario, some users may be negatively impacted compared to the user equilibrium because individual travel times are not taken into account, only the average of all users. In this way, a user may be allocated to a route which is not the best for him but improves the system. One example could be a user being forced to take a detour to avoid roads that are critical to others which is an allocation that is less likely to function in real world conditions because drivers may not adhere to the route assigned to them and take a shorter one in detriment of the system. As such, the user equilibrium model is the one that is closer to the real world since drivers seek to minimize their travel time.

Reinforcement Learning and Bandit Algorithms

Reinforcement Learning is a branch of machine learning that deals with learning in a way inspired by our own. The learning agent is not taught what is correct or not, or what it should or should not do. It receives no form of structured knowledge. It must then learn to explore and make choices based exclusively on feedback from the environment, the only form of information it receives. If its choice was a good one, it receives a better reward. If not, a worse one. It is important to notice that there are no absolute references to determine if that reward is good or not. It must be compared with other rewards received. An agent has no way of knowing when it reached an optimal reward only considering its value. Therefore, it must explore different alternatives it does not yet know. However, it must also exploit the knowledge it has learned. If it only exploits, it will never learn new information and if it only explores it will learn new information but will never use it.

Q-learning is a reinforcement learning algorithm based on Markov decision processes. In it, a player is in a specific state and must choose one out of a set of available actions to play. When it plays, it will receive a reward and may change state. The objective is to maximize the reward received. The rewards are specific for each action state pair. Q-learning estimates the utility of playing each pair (i.e. the usefulness of the reward it is expected to learn). The algorithm stores this information in a table called the Q-table which it uses to track these utility values. Each cell in this table is called a Q-value. For a player with two states and two actions, Table 2.2 shows an example. For state A action 2 has a larger expected utility while for B it is action 1.

The algorithm is iterative and updates the Q-table as the simulation progresses. Normally, the Q-values are initialized with random values or are all equal to the same value (e.g. zero). The Q-table is updated according to Equation 2.2 where r_t is the reward for the action a_t taken at state s_t on time step t. α is the learning rate which measures the importance given to new information, and γ is the discount factor which is the weight given to the estimate of future rewards to be obtained following this state-action pair. The algorithm must use the information in the Q-table for exploitation. For exploration, a heuristic is the $\epsilon - greedy$ strategy where the player will explore a random action with probability ϵ and the one with the maximum Q-value with probability $1 - \epsilon$. At every iteration of the algorithm, ϵ is decayed with a $\epsilon\ decay$ value. Algorithm 1 shows Q-learning with a $\epsilon - greedy$ strategy.

Q-learning can also be used in a single state or stateless manner. In this case, there is a single state, and every action will lead to a transition from and to it. The Q-table is then a list of the Q-values of every action.

Table 2.2: Example of Q-table with two actions and two states.

state	action	
	1	2
A	10	20
B	8	5

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{2.2}$$

Algorithm 1: Q-learning Algorithm

```
1  $Q[|states|][|actions|]$;
2  $s_t = state\ at\ time\ step\ t$;
3  **foreach** *t < NUM_EPISODES* **do**
4      **if** $random(0,1) < \epsilon$ **then**
5          $a_t = choose\ action\ randomly$;
6      **else**
7          $a_t = \arg\max_{a=1..k} Q(s_t, a)$;
8      **end**
9      $r, s_{t+1} = play(s_t, a_t)$;
10     $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(r + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$;
11     $t++$;
12     $\epsilon = \epsilon \times \epsilon\ decay$;
13 **end**
```

The Bandit class of algorithms was created to solve the several variations of the multi-armed bandit problem. It describes a scenario where a player can choose from a set of slot machines' arms to play with. Each arm has a different reward distribution.

The player must then learn which machine to play to maximize its received reward. In comparison to the Q-learning algorithm and the Markov decision process it is based on, the problem of the multi-armed bandit is stateless.

As mentioned previously, bandit algorithms must maximize their profit when playing *arms* with different reward distributions. Several algorithms have been proposed for this problem. (KULESHOV; PRECUP, 2010) describes some of them. Bandit algorithms are known for their applications on online advertising (CHAPELLE; LI, 2011). Their use in clinical trials was one of the original ideas behind the development of these algorithms. However, this real world application has not been widely studied (KULESHOV; PRECUP, 2010).

Classical bandits algorithms include UCB (Upper Confidence Bound) which explores each arm initially and later chooses which arm to play based on previous rewards, and Thompson Sampling that approximates each machine's reward distribution using one of its own, with parameters calculated from the rewards obtained from that arm on previous plays.

In the multi-armed non stationary bandit problem, the reward distribution of each machine changes through time. For the example of online advertising, the changing trends in society and individuals mean that there are no guarantees that what someone finds interesting today will remain so in times to come. Therefore, these algorithms must adapt to these changes. This is a problem for reinforcement learning algorithms because they must not only be able to learn information but also to forget them when they are no longer useful.

Algorithms that solve the stationary multi-armed bandit problem may not be able to handle the changes in rewards effectively and relearn the task. For this reason, some non-stationary algorithms were proposed.(GARIVIER; MOULINES, 2011) studies two modified versions of the UCB algorithm called discounted UCB and sliding-window UCB designed to deal with non-stationarity. (GUR; ZEEVI; BESBES, 2014) presents a modified algorithm called Rexp3 for the nonstationary multi armed bandit inspired by a softmax action selection.

The Upper Confidence Bounds (UCB1) Algorithms

The UCB1 algorithm is proposed as being optimistic in the face of uncertainty as described in (KULESHOV; PRECUP, 2010). The algorithm selects arms that maximize the upper bound on the rewards on each of them. This bound is calculated using the mean of past rewards and also a padding function that quantifies how unknown the reward distribution of that arm is. An arm that is played very often will have a smaller padding value than that of one played only once. The padding function would allow the agent to select less known options even though its rewards until now were not the best. One interesting aspect of this algorithm is that its regret is bounded. Regret is the difference between the cumulative rewards of always playing the best arms and the plays executed by the algorithm. A more detailed description of this algorithm can be found on (GARIVIER; MOULINES, 2011)

The UCB1 algorithm, however, was not designed to handle a nonstationary environment. Once each arm has been played a reasonable number of times, it will consider its arms as well-known even if its reward distributions have changed. Two nonstationary bandit algorithms are presented by (GARIVIER; MOULINES, 2011). The first is a discounted version of UCB1 (*Discounted UCB*) which weights the average of rewards based on their novelty: newer observations have larger weights. The authors propose a variation of this algorithm called *Sliding Window UCB* where the same discount is applied but only the last τ observations are considered.

Thompson Sampling

Thompson Sampling models each arm as a distribution with parameters taken from past observations (e.g. a normal distribution with mean and standard deviation calculated from the rewards). Its action choice is then probabilistic (CHAPELLE; LI, 2011). A random value is drawn from each arms respective distributions. The arm with the largest corresponding value is selected. For a normal distribution, the higher the average reward of an arm, the higher the values drawn will tend to be. Similarly, the smaller the standard deviation, the more certain and predictable will the same values will be.

The Rexp3 Algorithm

The Rexp3 algorithm is proposed in (GUR; ZEEVI; BESBES, 2014). It is an algorithm that explores with some probability ϵ and takes the action with the maximum expected reward otherwise. To deal with non-stationarity, the algorithm *forgets* everything it has learned previously and starts from the beginning at periods in time called epochs at predefined intervals.

3 RELATED WORK

Traffic Assignment Problem

The traffic assignment problem is a well-studied topic. Several algorithms were proposed to solve it. Some of them are very efficient. Normally, it is solved using assignment algorithms which try to allocate trips to routes in a centralized fashion as described in (ORTÚZAR; WILLUMSEN, 2011). We describe some of them. Most of them are described in the previously referenced work.

All or Nothing Assignment

This method does not consider congestion nor does it varies the cost of links based on their flow meaning that they are fixed. All trips are allocated to their respective shortest route. This assignment can easily congest the network excessively and is, therefore, not very useful in regards to optimization.

Stochastic Methods

In the All or Nothing assignment, all trips of an OD pair follow one single route which they all see as best. However, using a single route for an OD pair is not very realistic, nor efficient. Finding alternative routes is a challenge that stochastic methods seek to solve

In real world conditions, the perception of what is a good route varies from driver to driver, affecting their decisions. In *simulation based stochastic methods*, the cost of each link is determined by a random variable drawn from a distribution (e.g. normal) with the mean being the cost value determined in the network's definition. This represents the different perceptions that each driver has. The trips are then allocated into routes using All or Nothing assignment but considering the cost as seen by each driver. This method presents an improvement to the All or Nothing assignment due to the variability of routes. However, it still does not consider congestion.

Proportional stochastic methods work by going backward from the destination node to the origin. At each node, it splits the flow of trips into every inbound link which has as a source a node closer to the origin node than the current one. The proportion of flow directed into each link is defined through a *splitting factor* which is determined by the implementation and varies.

Congested Assignment

The previous methods did not take into account possible congestions on the network and their effect on the cost of choosing a certain route. They only considered fixed costs on links such as travel time under free flow causing the cost of traversing a link not to be dependent on the amount of traffic. However, on real scenarios, the amount of traffic on the link of a network affects the time needed to cross per the cost-flow relationship mentioned in the previous chapter and illustrated on (ORTÚZAR; WILLUMSEN, 2011).

The All or Nothing Assignment method allocates all the trips of each OD pair one after the other not worrying about congestion. The *Incremental Assignment* method splits the trips into groups. Before each group is allocated, the link costs on the network are updated based on previous allocations following a cost function designed to take into account the flow of vehicles. Then, the group is allocated based on the shortest paths calculated with the updated values. This is repeated until all groups were added. This method, while it does not allow the deallocation of trips, allocates trips gradually observing the effects on the network. The way trips are split into groups is defined by the implementation and there are no guarantees that the result will converge to the user's equilibrium.

A major problem of the previous methods is that it is not possible to deallocate trips to decrease congestion. The *Method of Successive Averages (MSA)* is an iterative algorithm where the allocation of all trips is updated at each iteration. At each step, the costs of all links are calculated according to the current flows and a new All or Nothing assignment is calculated. Its result updates the previous flow value according to the equation 3.1. Where n is the current iteration, F_l^n the flow on link l and G_l the flow on link l determined by the All or Nothing algorithm at the current iteration. ϕ is the weight given to new information. Each implementation of the algorithm may use a different value for it. With $\phi = 1/n$, the algorithm converges to the Wardrop's first equilibrium (SHEFFI, 1985).

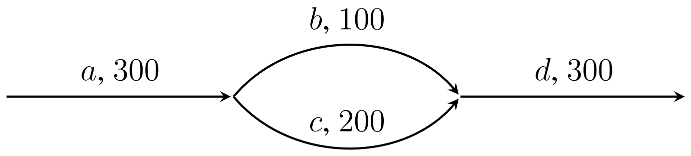
$$F_l^n = (1 - \phi)F_l^{n-1} + \phi \, G_l \tag{3.1}$$

TAPAS

Normally, allocation algorithms seeking the user's equilibrium provide consistent link flow values (i.e. the flow passing through each link). To properly understand the demand for the network it is necessary to know which routes are taken by each vehicle and not only the level of flow on each link. However, algorithms do not usually provide this information.

In the case of a network where trips of OD pairs are split and then merged back again as shown in Figure 3.1, the traffic passing through link a is split between links b and c, and back into link d. If there are trips from more than one OD pair crossing this links, how can we distribute them between b and c realistically? There are many different solutions (infinite if you consider flow a real number). Having many different possibilities to choose from is not very tractable, so the algorithm should be consistent on which it chooses. (BAR-GERA, 2010) proposes the TAPAS algorithm which seeks to split trips with equal proportions: in the example of Figure 3.1, two-thirds of the trips of each OD pair going through link c and the rest through b. The algorithm seeks to provide good solutions to the Wardrop's Equilibrium and to the route flows. It allocates the flows of vehicles from every OD pair into routes and associated flows following the equal proportions rule mentioned above.

Figure 3.1: Example of a section of a network.
Values accompanying each link represent their flow.



Route Choice

As mentioned earlier, route choice is similar to the TAP but focus on the drivers and their decisions. Algorithms for the TAP are mostly centralized while route choice allows for the use of decentralized algorithms. This problem can be modeled as a multi-agent system where each agent is a vehicle and it must choose which route to follow.

Multi-agent Reinforcement Learning-based Assignment

A decentralized approach to the Route Choice method with multi-agent reinforcement learning uses Q-Learning for each agent on the network such as those of (STEFANELLO; SILVA; BAZZAN, 2016). A major difference from centralized methods for the TAP is that they consider the k shortest routes for each OD pair in the network using the k shortest path algorithm by (YEN, 1971).

The problem is modeled in a multi-agent scenario where each vehicle is an agent. Each agent, then, is a stateless Q-learning instance which must choose between the k routes generated. The reward it receives is its own travel time. Thus, the system's goal is the User Equilibrium. It is important to notice that, here, the allocation of trips to *pre determined* routes that is optimized and not to links.

This method has some limitations regarding the non-stationarity of the environment. The assignment of trips to routes changes at each iteration as does the flow on links and their respective costs. These changes affect the costs of the links and effectively changes the problem: what an agent learned some iterations before may be useless because the environment changed.

Genetic Algorithms Based Assignment

A proposal to approximate the System Optimum is that of (BAZZAN; CAGARA; SCHEUERMANN, 2014). The authors propose the use of a centralized genetic algorithm where each individual is a complete allocation of all trips to routes to minimize the average travel time of the network.

An expansion of this work combining multi-agent reinforcement learning and genetic algorithms is presented in (BAZZAN; CHIRA, 2015). In this work, the worst individual in the population of the genetic algorithm is replaced by a solution from the Q-Learning algorithm. While seeking a System Optimum, the algorithm uses solutions from the Q-learning algorithm which represents what drivers would like to do individually.

4 PROPOSAL

We consider the Route Choice Problem in a multi-agent reinforcement learning scenario where each trip in the network (or vehicle) is represented as an agent. These agents learn and act through the use of a reinforcement learning algorithm. Q-learning has been successfully applied to the route choice problem with good results (RAMOS; SILVA; BAZZAN, 2017). However, Q-learning is stateful while our problem is often modeled as stateless. That is, the agents are always in a single state for the whole duration of the simulation.

Bandit algorithms seem adequate for the problem at least in a modeling perspective since they are simpler than Q-learning and provide for the action selection capability that is needed. This work seeks to evaluate the performance of different Bandits algorithms when applied to the Route Choice Problem and how they compare to Q-learning.

Non-stationarity in the Route Choice Problem

Problems such as route choice are heavily nonstationary. Since agents must share scarce resources, (i.e. the road network), the route taken by one agent can easily affect the travel time of the others. Imagine an agent that must take a specific highway to fulfill its trip independent of which route it may take. If other agents can avoid this highway, congestion on this link will decrease and so will the travel time of the first agent. Notice how the travel time of the first can change as a function of the other agents' actions. Each agent must adapt to a changing environment. This causes non-stationarity. A greater challenge, in this case, is that what changes is not the environment itself but the other agents. As was mentioned on section 3.2.1, the nonstationarity of route choice is a challenge to the multi-agent reinforcement learning approach. To solve this problem efficiently, the learning algorithm must be able to deal with this sort of agent-caused non-stationarity instead of those caused by outside elements.

Bandits Algorithms Used

The route choice problem was modeled with bandits algorithms by representing each of the available routes of each agent as an arm. The agent must learn to choose which route to take. It is important to notice that it is not common to see Bandit algorithms applied in a multi-agent system. The algorithms used are described in the following sections.

UCB1

Traditional UCB1 algorithm. It needs to be initialized to an initial knowledge of each of the available routes (arms). Normally, the algorithm tries each one of its arms sequentially. The problem with using this method on a multi-agent system is that all agents will be doing the same: They will all try arm number 1 simultaneously then number 2 and so on. Since it is a problem with non-stationarity caused by other agents actions, this initial exploration may provide unrealistic rewards since agents of each OD pair will always be concentrated on the same route.

Because of the problem caused by the order of initialization, we tested two variants of this algorithm (as well as Discounted UCB and Sliding Window UCB): One where the initialization is done sequentially and another where it is done randomly.

The action selection of the UCB1 algorithm considers the average mean of past observations as well as a padding function with value inversely proportional to the certainty the algorithm has of its knowledge of each arm. If an arm was played several times, then the mean of past rewards is considered accurate and the padding function value is small. If it has been played only a few times, the accuracy of using the mean value as an estimate for the arms reward distribution is low, and the latter may be better than the former. Algorithm 2 shows the pseudo-code for UCB1.

Algorithm 2: UCB1

```
Result: Arm to play
foreach episode < NUM_EPISODES do
    if episode < K then
        if init == sequential then
            $i_t = t$;
        else
            $i_t = choose\ randomly\ arm\ not\ yet\ played$;
        end
        $n_{i_t} = 1$
    else
        foreach k in K do
            $padding_k = \sqrt{\frac{\xi\ lnt}{n_k}}$;
        end
        $i_t = \arg\max_{i=1..k}\ \mu_i + padding_i$;
    end
    reward = play($i_t$);
    $\mu_i = \frac{\mu_i * n_{i_t} + reward}{n_{i_t} + 1}$;
    $n_{i_t} = n_{i_t} + 1$;
end
```

Discounted UCB

This is a variation of the previous algorithm where old observations are discounted based on decay values. With this modification, the algorithms then gives greater weight to more recent observations when calculating the mean reward of each arm. The algorithm is initialized in the same way as UCB1. The algorithm is presented below as in (GARIVIER; MOULINES, 2011). γ is the discount factor and B is an upper bound on rewards.

Algorithm 3: Discounted UCB1

```
Result: Arm to play
rewards = array of rewards;
episode = 0;
while episode < NUM_EPISODES do
    if episode < K then
        if init == sequential then
            $i = t$;
        else
            $i = choose\ randomly\ arm\ not\ yet\ played$;
        end
    else
        N = array of size |K|;
        M = array of size |K|;
        padding = array of size |K|;
        foreach k in K do
            foreach j in 1..episode do
                if arm k was played at time step j then
                    $N[k] = N[k] + \gamma^{episode-j}$;
                    $M[k] = M[k] + \gamma^{episode-j} * rewards[j]$;
                end
            end
            $M[k] = M[k]/N[k]$;
            $padding[k] = 2B\sqrt{\frac{\xi ln(sum(N))}{N[k]}}$;
        end
        $i = \arg\max_{a=1..k} M[a] + padding[a]$;
    end
    episode = episode +1;
    rewards[episode] = play($i$);
end
```

Sliding Window UCB

The sliding window UCB builds upon the Discounted UCB. Its only difference is that it considers at most the τ last observations. Creating a window with maximum width τ that moves across time. It is also defined in (GARIVIER; MOULINES, 2011).

Thompson Sampling

An optimization was done to our implementation of Thompson Sampling where the parameters for every distribution used by the algorithm to draw samples is updated only at the end of predefined intervals. In this way, the algorithm avoids having to recalculate them at every iteration.

Algorithm 4: Thompson Sampling

```
   Result: Arm to play
1  foreach episode < NUM_EPISODES do
2      if episode < K*2 then
3          $i_t = episode\ mod\ K$
4      else
5          foreach k in K do
6              $average_k = average(past\_rewards(k))$;
7              $std\_dev_k = standard\_dev(past\_rewards(k))$;
8              $distribution_k = normal(average\_k, std\_dev\_k)$;
9          end
10         $i_t = \arg\max_{k=1..K} draw(distribution_k)$
11     end
12 end
```

Rexp3

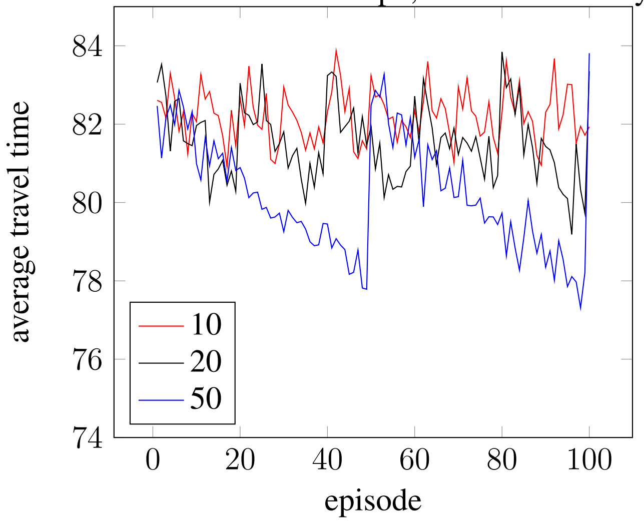
The problem with applying this algorithm to a multi-agent learning scenario is that when all agents *forget* simultaneously, all knowledge about the solution to the problem is lost. This can be seen on Figure 4.1. Notice the periodicity on the plotted lines denoting different frequencies of forgetfulness. When the agents forget, the travel time jumps upwards.

This algorithm was designed to deal with a different type of non-stationarity: the one caused by the environment. One example would be if the layout of the network changed or the online advertising selection that was mentioned earlier. For this reason, we attempted modifying this algorithm. Instead of all agents simultaneously *forgetting* at each epoch, there is a probability of an agent forgetting at any given time. This probability is decayed with a decay rate as the simulation progresses. We call it Rexp3MA. This modification provided only marginal improvement at best in relation to the original. To be able to compare these two algorithms, all simulations with Rexp3 did not include epochs with the exception of that of Figure 4.1.

Algorithm 5: Rexp3 (without forgetting epochs)

```
Result: Arm to play
1  foreach k in K do
2      $w_k = 1$;
3  end
4  foreach episode < NUM_EPISODES do
5      foreach k in K do
6          //p is a probability distribution;
7          $p_k = (1-\epsilon)\frac{w_k}{\sum_{i=1}^{K} w_i} + \frac{\epsilon}{K}$;
8      end
9      $i_t = draw\ from\ distribution\ p$;
10     $reward = get\_reward(i_t)$;
11     foreach k in K do
12         if $k == i_t$ then
13             x = reward;
14         else
15             x = 0;
16         end
17         $w_k = w_k e^{\frac{\epsilon x}{K}}$;
18     end
19 end
```

Figure 4.1: Average Travel time on OW with Rexp3, $\epsilon = 0.01$ and varying epoch intervals



5 EXPERIMENTAL RESULTS

The algorithms mentioned in Chapter 4 were tested in two traffic networks: The OW network from (ORTÚZAR; WILLUMSEN, 2011) and Sioux Falls from the Bar-Gera repository [1].

Experiment Methodology

All algorithms were compared using the average travel time of all agents or trips. The reward given to each agent at every episode is the negative of the travel time at that moment of the route chosen. The cost functions of each of the networks tested were those in their definitions. Both these network files can be found in <https://github.com/maslab-ufrgs/network-files>. The experiments require a parameter k to determine the number of possible routes for each OD pair. Therefore, each agent will have k actions to choose from corresponding to each route. The results shown for all algorithms with random components are the averages of 30 runs. Due to long running times for the Sioux Falls network, each agent represents 100 trips in the simulation. Therefore, the choice of every agent is replicated 100 times to correspond to the demand on the network. Since the Sioux Falls network has a demand of 360600 trips, decreasing the number of trips by a factor of 100 makes the network demand more similar in size to that of the OW.

Comparison of the Initialization Order of UCB

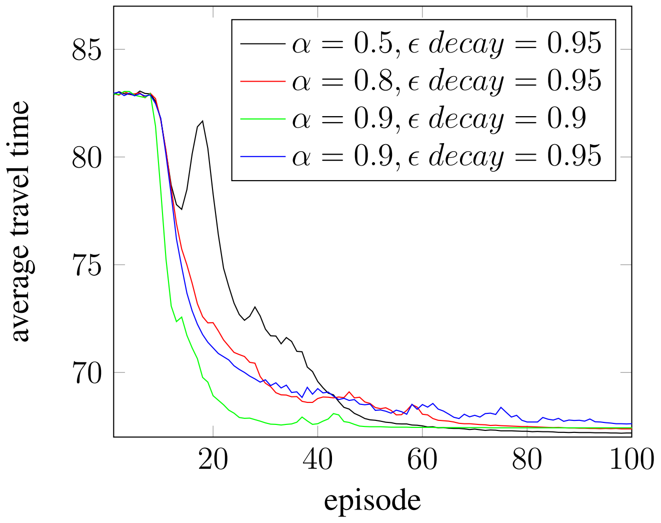
As mentioned in Subsection 4.2.1, the order of initialization of the UCB algorithms can have consequences in a multi-agent environment. To ascertain which variation would fare better, they were tested with both networks in consideration. For the OW network, random initialization provided better results than sequential. However, for the Sioux Falls network, the opposite is true. Figures 5.2 and 5.9 show the comparisons for both networks. Every experiment result presented involving UCB1, Discounted UCB and Sliding Window UCB were executed with the best strategy for each network.

[1]Transportation Networks for Research. <https://github.com/bstabler/TransportationNetworks>

Experiments with the OW network

The OW network is presented by (ORTÚZAR; WILLUMSEN, 2011). It is a quite simple network with few nodes and only four OD pairs. A visualization is shown in Figure 2.1. We benchmark our results with the Q-learning algorithm. To do so, we first simulated with Q-learning to obtain parameters at which good results are found. Some of them are presented in Figure 5.1 all of which had an initial ϵ value of 1. The best average travel time with Q-learning was 67.19 time units with $k = 8$ routes.

Figure 5.1: Average Travel time on OW with Q-learning with varying alphas and decays. Error bars represent standard deviation.



For the UCB1 algorithm, reasonable results were obtained though not close to the ones with Q-learning. The non-stationary variants had better results. The average travel times with the Discounted UCB were good and the Sliding Window UCB was slightly behind it. Figures 5.2, 5.3 and 5.4 show the average travel times for these algorithms.

Figure 5.5 shows the travel times for the Thompson Sampling across time for the OW network. Thompson Sampling did not have as good results, but it was not far behind. The steps on its plot show the episodes where the parameters of the distributions are updated. Increasing the frequency of such updates provided no improvement to the end result and increased run time.

The Rexp3 and Rexp3MA algorithms had the worst results of all algorithms tested. Figure 5.6 shows these results. For Rexp3, we set the epoch interval to be larger than the number of episodes to prevent the effects it causes as shown in Subsection 4.2.5. Even though they explore significantly more than the other bandit algorithms, they couldn't provide good results and the rate at which they decreased their average travel time was quite slow when compared to the others. Furthermore, Rexp3MA showed no significant advantage over Rexp3 with similar parameters.

Figure 5.7 shows how all algorithms compare. For the OW network, some bandit algorithms had reasonable results. Though they did not outperform Q-learning, some of them were able to match it. Nonetheless, the OW network is quite simple and may not be very challenging for these algorithms.

Figure 5.2: Average Travel time on OW with UCB1.
The two lines show the different initialization methods

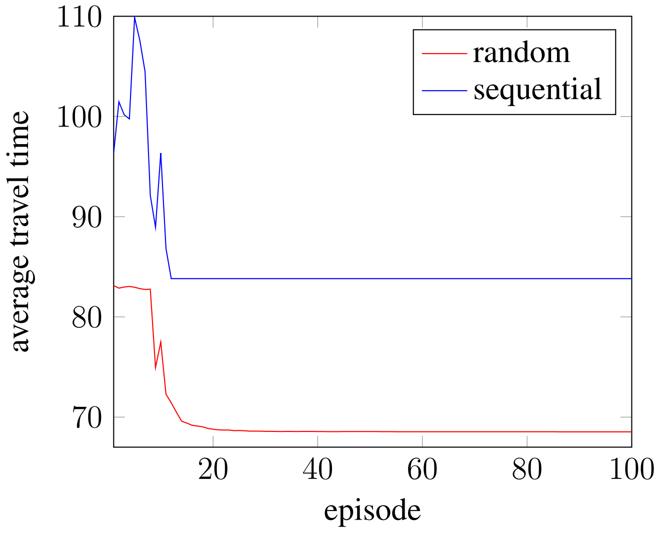


Figure 5.3: Average Travel time on OW with Discounted UCB

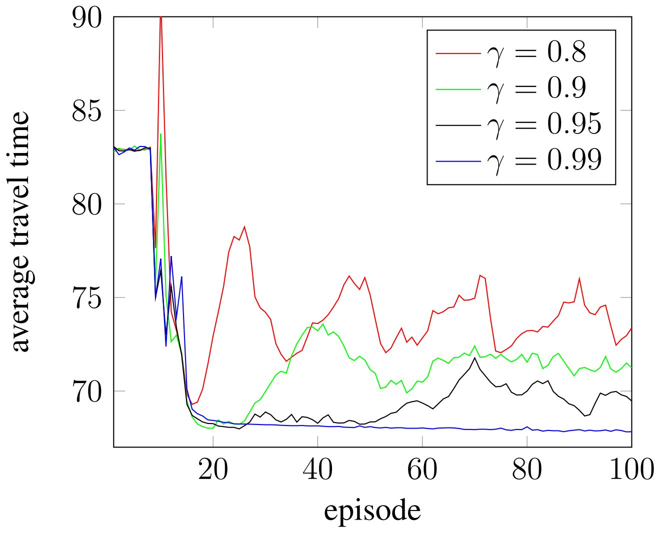


Figure 5.4: Average Travel time on OW with Sliding Window UCB with $gamma = 0.99$ and varying window size

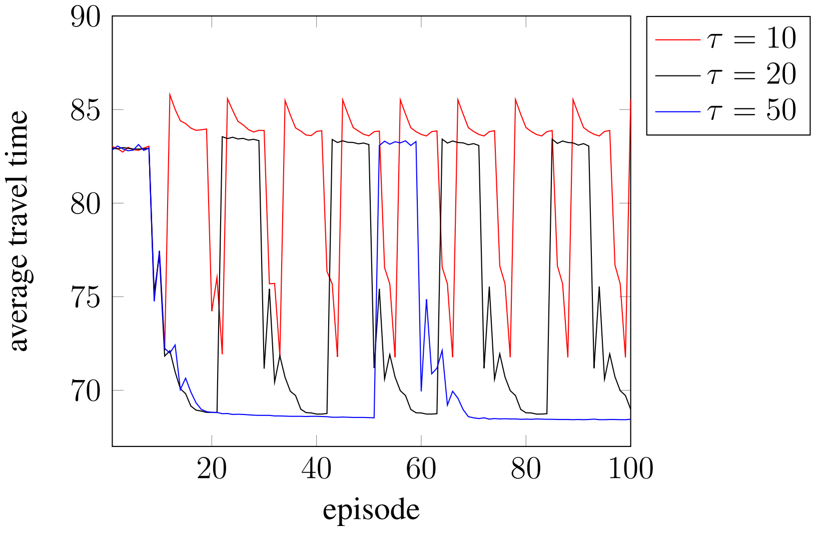


Figure 5.5: Average Travel time on OW with Thompson Sampling

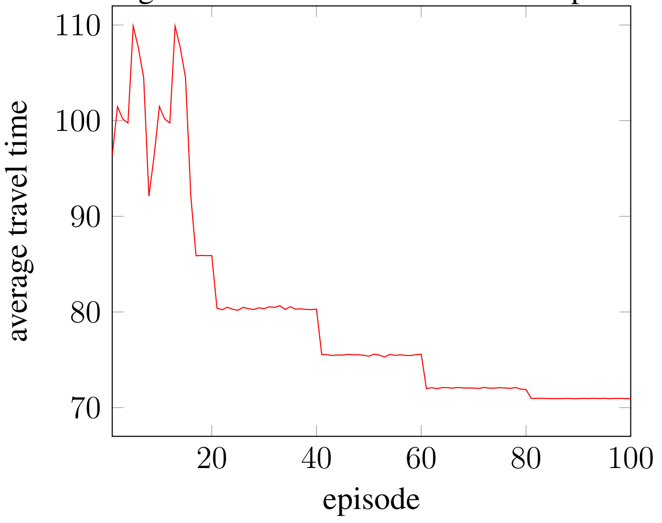


Figure 5.6: Average Travel time on OW with Rexp3 and Rexp3MA.
The Rexp3MA series has additional parameters: $pf = 0.002$ and $\gamma = 0.9$

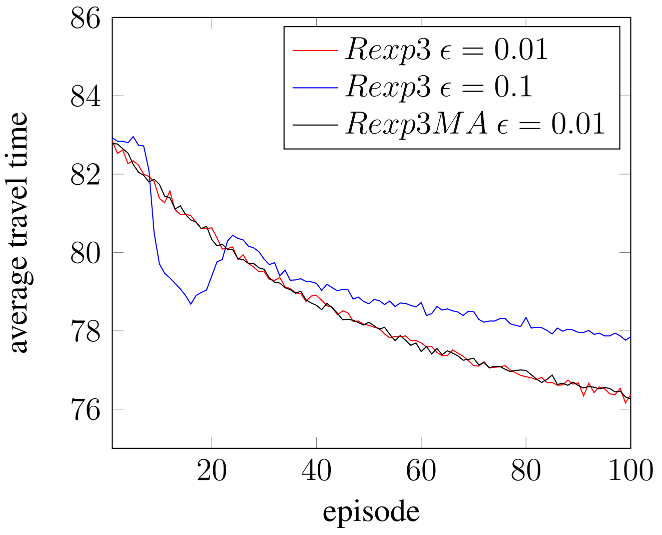
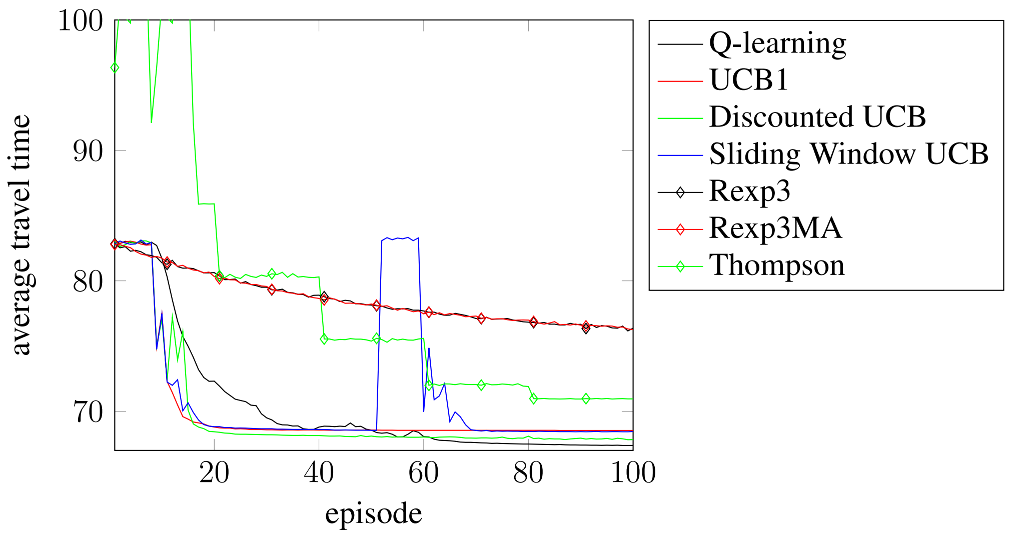


Figure 5.7: Average Travel time on OW with all algorithms



Experiments with the Sioux Falls Network

The Sioux Falls network is much larger. It has 528 OD pairs with a total 360600 trips. It is, therefore, much more complex than the OW network and a greater challenge. With this network, the bandit algorithms had a much worse performance when compared to Q-learning than in the previous. Q-learning had a minimum average travel time of 22.26 time units with $k = 4$ routes. Figure 5.8 shows some of the parameter combinations for the Q-learning algorithm with the best results. Discounted UCB and Sliding Window UCB demonstrated no advantage over UCB1. Rexp3 and Rexp3MA had the best performances excluding Q-learning. These results were in great contradiction with their previous results with the OW network. The results of these two algorithms change quite slowly. We show on Figure 5.14 how Rexp3 approaches the result of Q-learning over a longer period of time. Rexp3 and Rexp3MA had no significant difference in their results with similar parameters.

Figure 5.8: Average Travel time on Sioux Falls with Q-learning

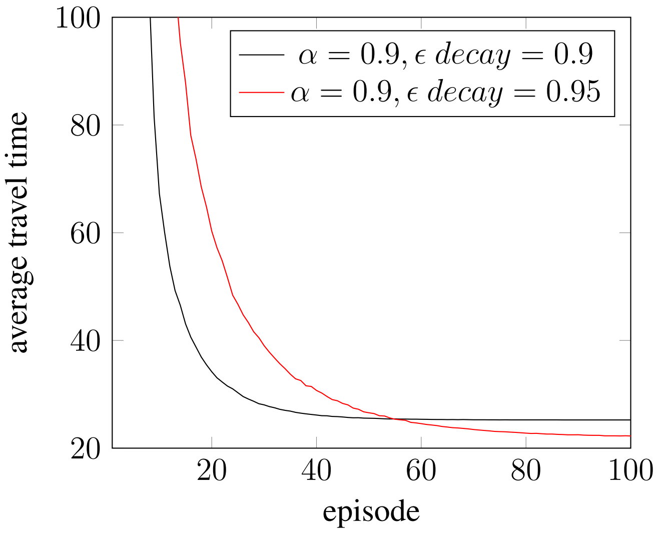


Figure 5.9: Average Travel time on Sioux Falls with UCB1.
The two lines show the different initialization methods

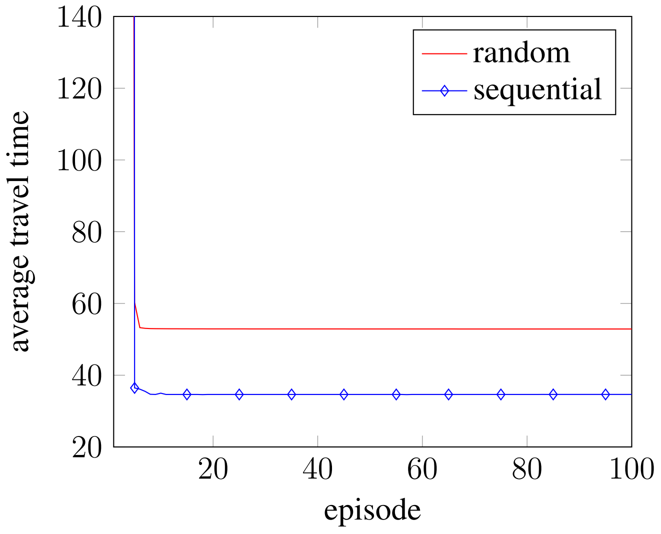


Figure 5.10: Average Travel time on Sioux Falls with Discounted UCB and Sequential Initialization

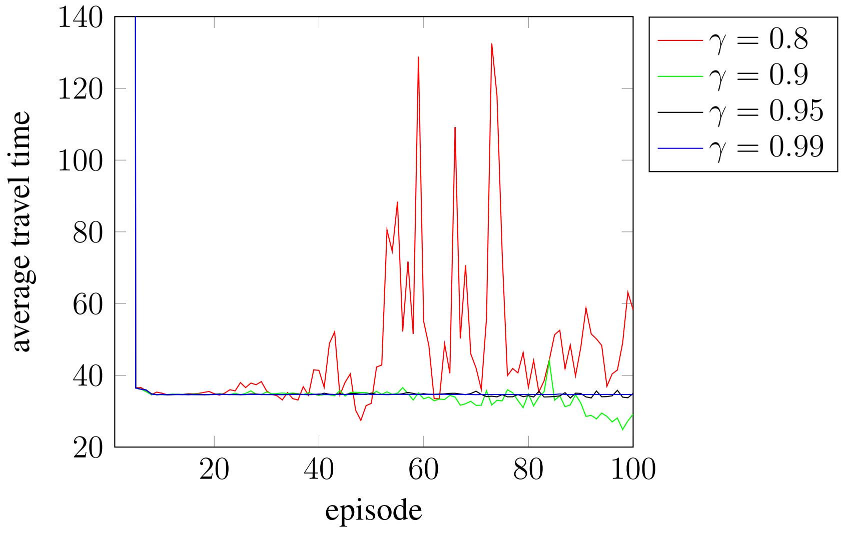


Figure 5.11: Average Travel time on Sioux Falls with Sliding Window UCB using $\gamma = 0.99$ and Sequential Initialization

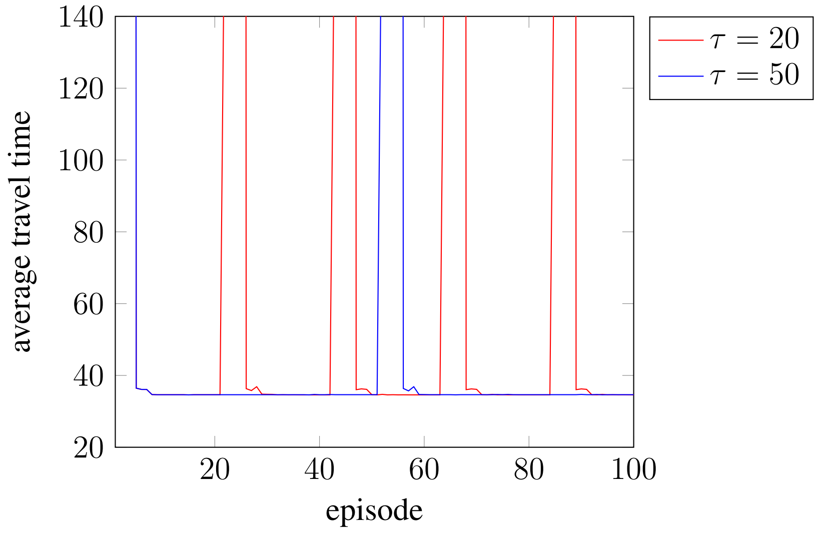


Figure 5.12: Average Travel time on Sioux Falls with Thompson Sampling

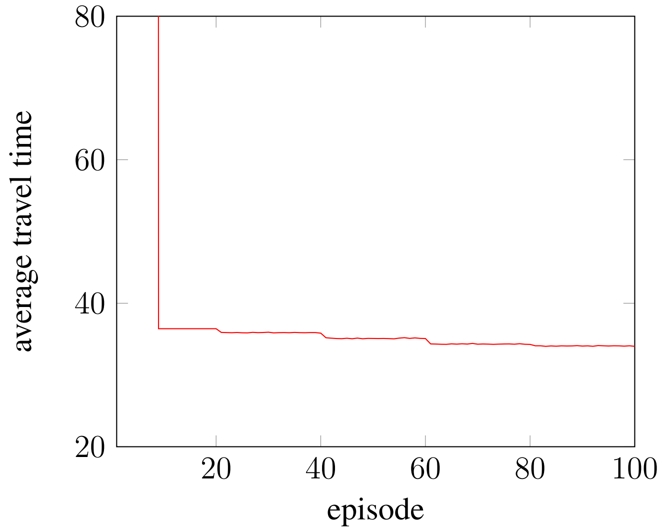


Figure 5.13: Average Travel time on Sioux Falls with Rexp3 and Rexp3MA. The Rexp3MA series has additional parameters: $pf = 0.001$ and $\gamma = 0.9$.

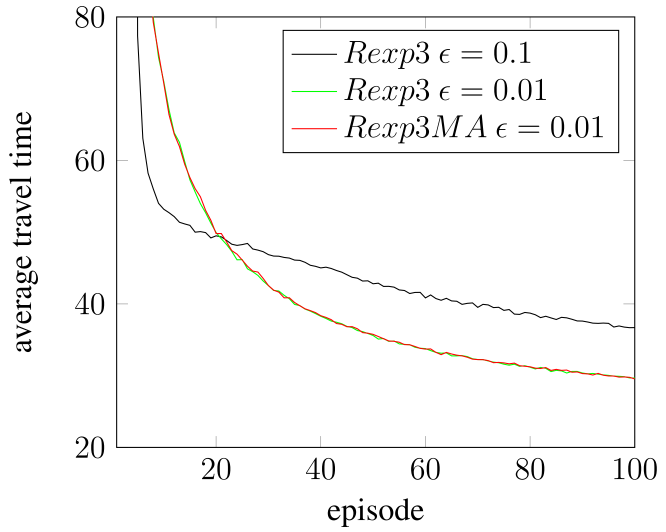


Figure 5.14: Average Travel time on Sioux Falls with Rexp3 and Q-learning over 300 episodes.

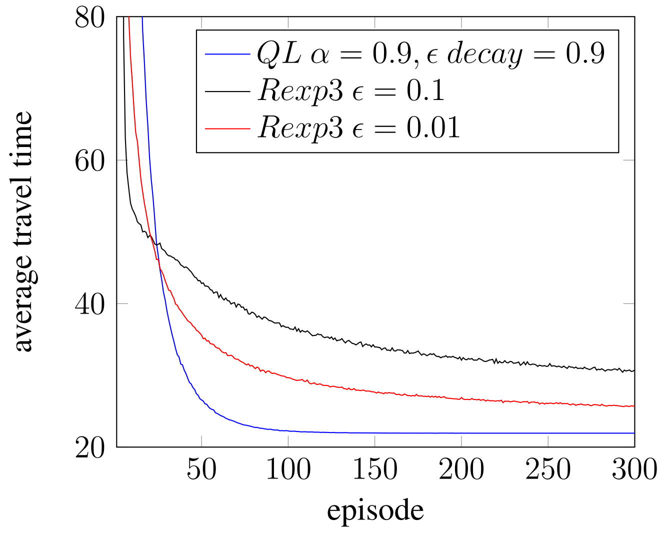
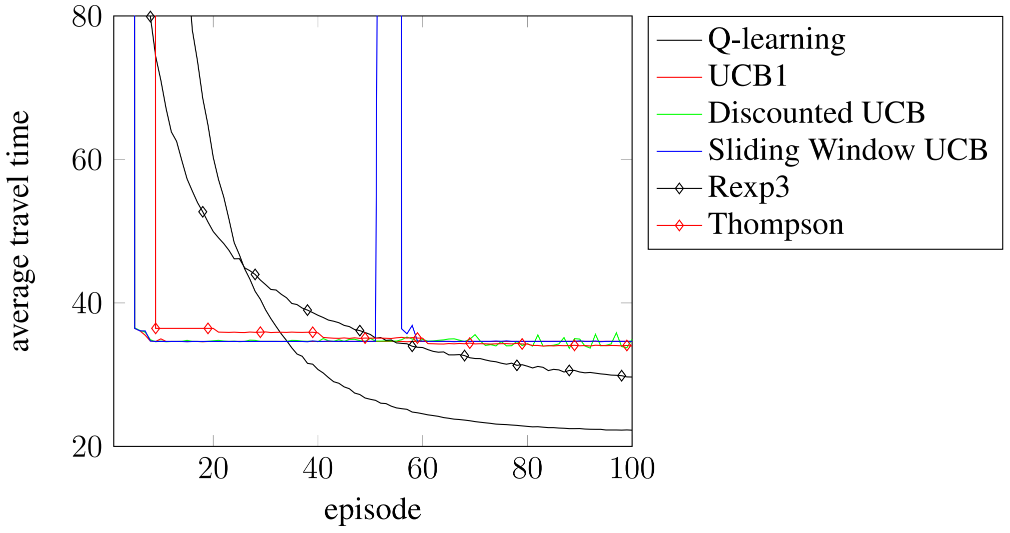


Figure 5.15: Average Travel time on Sioux Falls with all algorithms



Comparison of Results

We show in Table 5.1 the average travel times for both networks with the different algorithms. In Table 5.2, we include results for these networks with the method of successive averages (MSA), the user equilibrium and system optimum. Table 5.3 shows approximate run times measured using an i5-2540M processor with 6 GiB of RAM and Debian 8.

Table 5.1: 100th episode average travel time for both networks and all algorithms

algorithm	OW		Sioux Falls	
	avg	std. dev.	avg	std. dev.
Q-learning	67.19	0.04	22.26	0.16
UCB1 Random	68.53	0.24	52.88	1.02
UCB1 Sequential	83.82	-	34.64	-
Discounted UCB Random	67.83	0.59	35.97	1.42
Discounted UCB Sequential	78.24	-	29.19	-
Sliding Window UCB Random	68.21	1.31	43.57	3.81
Sliding Window UCB Sequential	72.76	-	34.59	-
Thompson Sampling	70.95	0.09	33.99	0.29
Rexp3	76.37	0.54	25.71	0.37
Rexp3MA	76.26	0.28	29.55	0.52

Table 5.2: User equilibrium, system optimum and MSA results for both networks

	OW	SF
MSA	67.46	20.78
UE	67	20.76
SO	66.92	19.95

Table 5.3: Approximate run times for 100 episodes

algorithm	run time (s)	
	OW	SF
Q-learning	1.93	5.74
UCB1	2.6	7
Discounted UCB	10.16	23.21
Sliding Window UCB	5.72	13.6
Thompson Sampling	7.4	13.12
Rexp3	21.47	45.93
Rexp3MA	20.92	44.26

The Qualities and Shortfalls of Bandits Algorithms

We noticed a significant shortfall of the UCB algorithm and its padding function. In the domain of the route choice problem, reward values can fluctuate significantly. With networks such as Sioux Falls, it varies by orders of magnitudes. The problem is that the padding function does not scale in the same manner. The ξ parameter serves to scale this function value but it is constant and of little use when rewards values change in the way they do. The consequence of this is that the padding function value is basically ignored. Table 5.4 shows an example of the mean of each route and the padding function value for one agent.

Table 5.4: Means and padding function values of agent in the Sioux Falls network

route	mean	padding function value
1	-6.00	0.38
2	-52.26	2.86
3	-352.46	2.86
4	-273.59	2.86

In this case, the first route is always taken. None of the padding function values of the other routes can make the algorithm choose them and experiment with them. If the agents do not experiment, they do not discover better route allocations. Notice how on Figures 5.2 and 5.9 the average travel time remains constant after the algorithm's initialization. Agents would concentrate on a route and would experiment the others when their observations were discarded or discounted but then return to their previous setup. Three of the four routes on Table 5.4 have the same padding function values. For this to happen, they must have been played an equal number of times. The non-stationary variants (Discounted UCB and Sliding Window UCB) had similar problems. Furthermore, the results of these algorithms when randomly initialized tended to vary more between runs than the other algorithms tested as is shown in Table 5.2. There would be some actions that were barely explored and when their respective observations expired, many agents would attempt it again at the same time causing the jump in average travel times seen in Figures 5.3, 5.4, 5.10 and 5.11. Applying these algorithms in a multi agent scenario is not straightforward. Agents coordinate their actions by always exploring at the same time and very often choosing the same actions as each other. Interestingly, the low exploration of these algorithms seem graver than the stationarity of the UCB1 algorithm in itself because they were never capable of exploring enough to have learned significant knowledge at any given time for that knowledge to be no longer valid.

Thompson Sampling, although having a random component, has similar problems with exploration as UCB. After the initial exploration, it remains mostly static. Exploration in later episodes is significantly restricted. With the OW network, the performance was better than with Sioux Falls (probably because the first is simpler). In fact, even with the first network, it explored poorly. Of all agents, all of them had routes that were only tried after the initialization phase. In most cases, the agent effectively restricts their plays to two or fewer routes. This is not completely unexpected. This algorithm was not designed with non-stationarity in mind. If it forms an *opinion* on a route too soon, as is the case, it will never be able to approach optimality since the reward value of a route depends on what the other agents are attempting and can vary sharply.

Rexp3 is at the same time a good and bad algorithm. It explores significantly more than the others and, more importantly, thanks to its stochastic nature, the agents are independent when exploring. There is little use for exploration if all agents decide to explore at the same time the same routes. However, the Rexp3's simultaneous *forgetting* event cannot be used for the route choice problem in the way it is intended and the algorithm can be slow to converge. Rexp3MA did not produce significant improvements to the average travel time in relation to Rexp3. The performance of the two algorithms with the Sioux Falls network was surprising after the poor performance with the OW. Though both had the closest results to the one of Q-learning, they were still underperformers. The greater ability for exploration in relation to the other algorithms was probably the reason for its better performance with the larger network. For the OW, although it explored, it was not able to produce good results.

6 CONCLUSION

The traffic assignment problem is that of allocating trips in a traffic network. We call the route choice problem its variant where drivers choose the routes they will follow instead of a centralized system.

The route choice problem has been modeled as a multi-agent reinforcement learning problem. This work studies the applicability of Bandit algorithms to it. More specifically, we compare the results presented by these algorithms to Q-learning which has been used previously with good results.

The UCB family of bandit algorithms demonstrated severe difficulty for effective exploration in this problem. It is mainly attributed to its deterministic behavior that when used in a multi-agent scenario, restricts the ability of the algorithm to provide good solutions.

Rexp3 without *forgetting*, while not having good results in the OW network, was the second best in the Sioux Falls only over-performed by Q-learning. Its approach to the reinforcement learning is closer to that of Q-learning than to UCB1 using a probabilistic random exploration instead of deterministic behavior that when synchronized among all agents have negative consequences.

While some of the bandit algorithms studied showed some interesting potential, they did not provide better solutions than Q-learning. However, work is needed to better ascertain the applicability of these algorithms to the problem.

Further work on this topic is needed in the study of additional bandit algorithms and, possibly, the proposal of newer ones better capable of handling the route choice problem or modifications to those existing. Another line of work could be the way the reward is calculated for each agent which could affect the results of the algorithms.

REFERENCES

BAR-GERA, H. Traffic assignment by paired alternative segments. **Transportation Research Part B: Methodological**, v. 44, n. 8-9, p. 1022–1046, sep 2010. ISSN 01912615.

BAZZAN, A. L. C.; CAGARA, D.; SCHEUERMANN, B. An evolutionary approach to traffic assignment. In: **2014 IEEE Symposium on Computational Intelligence in Vehicles and Transportation Systems (CIVTS)**. IEEE, 2014. (SSCI), p. 43–50. ISBN 978-1-4799-4498-9. Available from Internet: <http://dx.doi.org/10.1109/CIVTS.2014.7009476>.

BAZZAN, A. L. C.; CHIRA, C. Hybrid evolutionary and reinforcement learning approach to accelerate traffic assignment (extended abstract). In: BORDINI, R. et al. (Ed.). **Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015)**. IFAAMAS, 2015. p. 1723–1724. Available from Internet: <http://www.aamas2015.com/en/AAMAS_2015_USB/aamas/p1723.pdf>.

BRAESS, D. Über ein Paradoxon aus der Verkehrsplanung. **Unternehmensforschung**, v. 12, p. 258, 1968.

CHAPELLE, O.; LI, L. An Empirical Evaluation of Thompson Sampling. **Advances in Neural Information Processing Systems**, p. 2249—-2257, 2011. Available from Internet: <http://explo.cs.ucl.ac.uk/wp-content/uploads/2011/05/An-Empirical-Evaluation-of-Thompson-Sampling-Chapelle-Li-2011.pdf>.

GARIVIER, A.; MOULINES, E. On upper-confidence bound policies for switching bandit problems. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 6925 LNAI, n. 1985, p. 174–188, 2011. ISSN 03029743.

GUR, Y.; ZEEVI, A.; BESBES, O. Stochastic Multi-Armed-Bandit Problem with Non-stationary Rewards. **Advances in Neural Information Processing Systems 27**, p. 199–207, 2014. ISSN 10495258. Available from Internet: <http://papers.nips.cc/paper/5378-stochastic-multi-armed-bandit-problem-with-non-stationary-rewards.pdf>.

KULESHOV, V.; PRECUP, D. Algorithms for the multi-armed bandit problem. **Journal of Machine Learning**, v. 1, p. 1–32, 2010. Available from Internet: <http://www.cs.mcgill.ca/{~}vkules/bandits.p>.

ORTÚZAR, J. d. D.; WILLUMSEN, L. G. **Modelling transport**. 4. ed. Chichester, UK: John Wiley & Sons, 2011.

RAMOS, G. de. O.; SILVA, B. C. da; BAZZAN, A. L. C. Learning to minimise regret in route choice. In: DAS, S. et al. (Ed.). **Proc. of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017)**. São Paulo: IFAAMAS, 2017. p. 846–855. Available from Internet: <http://ifaamas.org/Proceedings/aamas2017/pdfs/p846.pdf>.

SHEFFI, Y. **Urban transportation networks**. Englewood Cliffs, NJ: Prentice-Hall, 1985.

STEFANELLO, F.; SILVA, B. C. da; BAZZAN, A. L. C. Using topological statistics to bias and accelerate route choice: preliminary findings in synthetic and real-world road networks. In: **Proceedings of Ninth International Workshop on Agents in Traffic and Transportation**. New York, USA: [s.n.], 2016. p. 1–8. Available from Internet: <http://ceur-ws.org/Vol-1678/paper11.pdf>.

WARDROP, J. G. Some theoretical aspects of road traffic research. **Proceedings of the Institution of Civil Engineers, Part II**, v. 1, n. 36, p. 325–362, 1952.

YEN, J. Y. Finding the k shortest loopless paths in a network. **Management Science**, v. 17, n. 11, p. 712–716, 1971. Available from Internet: <http://pubsonline.informs.org/doi/abs/10.1287/mnsc.17.11.712>.