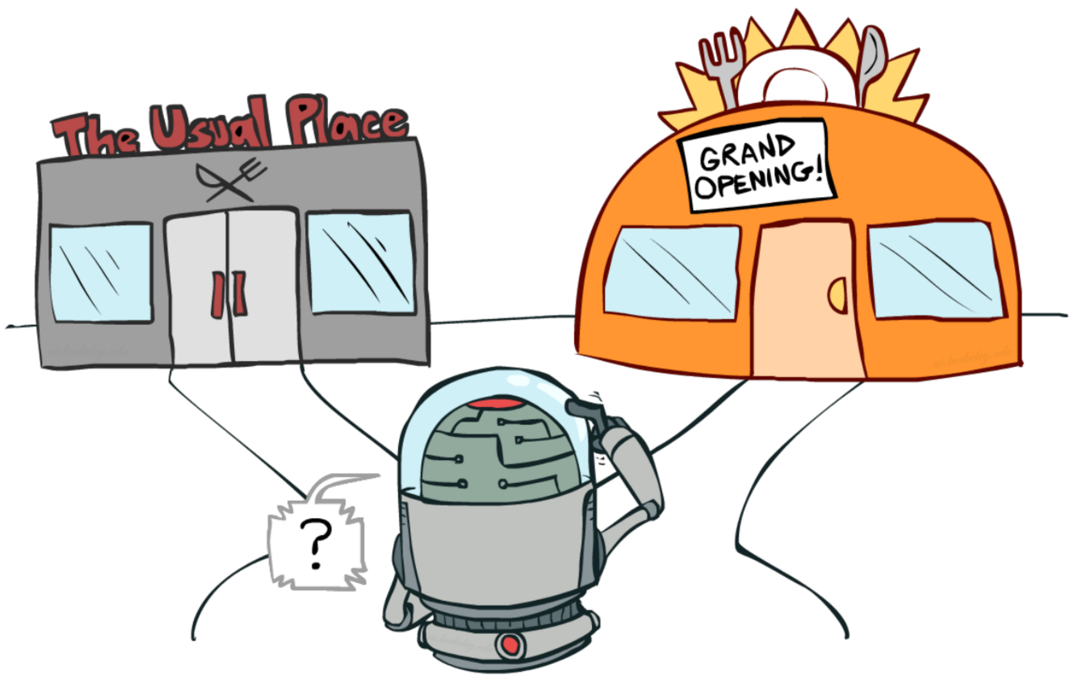


Learning to Navigate Over Stochastic Transport Networks Using Multi-Armed Bandits

A Contextual Approach for Efficient Online Learning in Road Network Graphs with Multi-Armed Bandits to Minimize Long-Term Travel Time

Master's thesis in Data Science and AI

Hannes Nilsson
Rikard Johansson

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Learning to Navigate over Stochastic Transport Networks using Multi-Armed Bandits

A Contextual Approach for Efficient Online Learning in Road Network Graphs with Multi-Armed Bandits to Minimize Long-Term Travel Time

Hannes Nilsson
Rikard Johansson

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Learning to Navigate over Stochastic Transport Networks using Multi-Armed Bandits
A Contextual Approach for Efficient Online Learning in Road Network Graphs with Multi-Armed Bandits to Minimize Long-Term Travel Time
Hannes Nilsson[†]
Rikard Johansson[†]

Supervisor: Morteza Haghir Chehreghani, Department of Computer Science and Engineering
Examiner: Devdatt Dubhashi, Department of Computer Science and Engineering

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Agent concerned with explore-exploit dilemma. Source: UC Berkeley AI course slides[1], lecture 11.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2024

[†]Both authors contributed equally to all parts of this project.
[1]http://ai.berkeley.edu/lecture_slides.html

Learning to Navigate over Stochastic Transport Networks using Multi-Armed Bandits
A Contextual Approach for Efficient Online Learning in Road Network Graphs with Multi-Armed Bandits to Minimize Long-Term Travel Time
Hannes Nilsson
Rikard Johansson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

As part of the ongoing process of phasing out fossil fuel vehicles, attempts have been made to extend the effective range and adoption rate of electric vehicles through navigation systems focused on energy consumption. One way to approach this problem is by viewing route selections as a multi-armed bandit problem. This allows the system to adapt and recommend better routes over time, to minimize energy consumption.

For navigation systems to be useful in practice, guiding vehicles from one point to another in minimal time is crucial. Therefore, this project examines the effectiveness of multi-armed bandit algorithms for time-efficient navigation in complex real-world environments, without initial information. For this purpose, we adapt a previously studied online learning framework developed for energy efficiency, and extract road segment travel time distributions from the traffic simulation software SUMO.

The framework is applied to the Luxembourg road network and our results demonstrate that contextual multi-armed bandits using tree ensembles are highly effective. More specifically, we show that TEUCB and TETS, which we implement using both XGBoost and random forest, outperform state-of-the-art contextual multi-armed bandits based on neural networks and linear models.

Further, by additional comparison of TEUCB and TETS to other bandit algorithms based on tree models, we identify at least two properties to explain their high level of performance. First, tree ensemble methods appear to offer relatively accurate travel time predictions from the contextual information available in this problem. Second, the ability to generalize over different arms and infer the travel time on one road segment from observations gathered on other ones, based on similarities, seems highly advantageous for this problem.

Keywords: decision-making under uncertainty, online learning, machine learning, reinforcement learning, multi-armed bandits, navigation, shortest path problem

Acknowledgements

We would like to express our gratitude to our supervisor Morteza Haghir Chehreghani and co-supervisor Niklas Åkerblom for their support throughout the thesis. Their insights and support have been invaluable. We also want to thank Jack Sandberg for his contributions to this work. Further, we thank Devdatt Dubhashi who has been our examiner for this thesis.
Additionally, we want to highlight that the computations and data handling of this project were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725. We are thankful for these resources, which have been critical to completing this thesis.
Last but not least, we would like to express our sincere gratidue to our respective families for their love and support.

Hannes Nilsson & Rikard Johansson, Gothenburg, June 2024

Contents

List of Figures

List of Tables

1
Introduction

This thesis focuses on intelligent navigation over road networks where the goal is to minimize the expected travel time throughout a predetermined time period T. The thesis will investigate different types of exploration strategies and machine-learning techniques. The feedback received from the environment when the navigating agent takes an action—that is selecting a path between two points in the road network—is based on real-world data. Essentially, the agent will learn to navigate without prior knowledge in a graph where the feedback from the edges is stochastic. In essence, the challenge consists of balancing between exploration and exploitation so as to minimize the time spent on the road in the long run.

1.1 Background

In [1], an adaption of the contextual multi-armed bandit framework for efficient navigation over stochastic road networks is developed. In their work, the objective is to minimize energy consumption over time by effectively balancing the trade-off between exploring new road segments and exploiting historically energy-efficient ones.

Their work makes the crucial assumption of independent energy consumption over different road segments, allowing for the use of traditional bandit methods based on Bayesian inference. However, independence may not hold in practice, suggesting that the problem may allow for more data-efficient bandit methods that find correlations in the energy required to travel on different road segments, based on some characteristics. This scenario is examined in an extended work [2], in which Gaussian Processes are used to find correlations between the energy consumption on different edges in the road graph for efficient estimation.

In addition to finding correlations between edges, taking characteristics into account allows one to handle non-stationary distributions. When aiming to accurately predict how long it takes to travel on a road segment, which is the scope of this work, the travel time is affected by the level of traffic and so highly depends on the time of the day. Hence, the current time of driving is considered as information available to the agents here, rendering the problem an instance of contextual combinatorial multi-armed bandits.

Just as has been done for the case of minimizing energy consumption over time, one could attempt to solve the time-minimization problem using Gaussian Processes. However, as the relationship between the time of the day and expected travel time on all different road segments is anticipated to be rather complex, Gaussian Process bandits may not offer satisfactory performance on this problem. Therefore, this work investigates the use of current state-of-the-art machine learning models, such as artificial neural networks and decision tree ensembles, in conjunction with sampling methods for bandits.

1.2 Problem formulation

The goal of this thesis is to investigate how multi-armed bandit algorithms can be used for efficient navigation in real-world road networks, and will compare contextual with non-contextual bandit algorithms. For the contextual bandit algorithms, there will be different types of base learners, such as decision trees, neural networks and linear models. Furthermore, the contextual bandits used can be separated into two different groups based on whether they consider a separate predictive model for each available arm, or uses one single model for making predictions on all arms.

Different algorithms will be applied and evaluated for navigating in real-world road networks represented by a graph $\mathcal{G}=(\mathcal{V}, \mathcal{E})$, explained in Chapter 2. The edge weights are determined by the particular environment, and the agents are tasked with navigating with increased complexity for the different problem settings.

This thesis will focus on solving the successive shortest path problem as effectively as possible, using multi-armed bandit algorithms. The path selections are evaluated according to the total expected travel time, and the distribution of the travel times for the edges are initially unknown.

Ultimately, we set out to identify which bandit method(s) best suits the problem at hand. As the focus is on minimizing total travel time, the thesis primarily emphasizes fast learning with limited data, in order to minimize the total time spent on the road. However, the thesis is not limited to data efficiency. As the algorithms are online learners, one could expect it to be beneficial for the code to run as fast as possible. Furthermore, applications may require that algorithms to have the potential of being trained and used on embedded hardware, not needing GPUs for training. Therefore, we will also investigate the computational resources required by the agents to solve the problem.

1.3 Thesis contributions

As discussed in Section 1.1, the problem of efficient navigation in road networks with respect to energy consumption has been well studied, and satisfactory results have been obtained using multi-armed bandits. However, when it comes to minimizing travel time, we are unaware of any successful prior research. For a navigation system to be useful in practice, both energy consumption and travel time are of utmost interest. We hope that the results we present in this work will provide valuable insights into how one might go about implementing multi-armed bandit methods for efficient navigation. Not only to minimize time consumption, but also in combination with previous works on energy efficiency for developing useful navigation tools.

1.4 Thesis outline

In Chapter 2, we will explain the theory of the shortest path problem, multi-armed bandits with various setups, machine learning methods that will be used, and the graph structure of the problem considered in the experiments.

Chapter 3 looks at the specific experimental structure and formalizes the problem. The dynamics of the different environments are described, as well as how the agents are evaluated.

The results are presented in Chapter 4 for the different agents evaluated, described Chapter 2, tasked with navigating in the different environments presented in Chapter 3.

Chapter 5 discusses the results obtained with the algorithms considered for the problem, difficulties with the environments, and what is left for future work.

In Chapter 6 we present what conclusions can be drawn from the study.

2
Theory

In this chapter, the shortest path problem, what makes up a graph, and multi-armed bandits will be explained.

2.1 Shortest path problem

Finding the shortest path between two vertices in a graph is a common, known, and solvable problem referred to as the shortest path problem. Shortest path-finding algorithms exist, such as Dijkstra's [3] or Bellman-Ford [4], that can find optimal paths in polynomial time. The graphs in this thesis represent a road network. In a road network the vertices represent the intersections and the edges represent road segments connecting those intersections. However, solving the shortest path problem, with Dijkstra's or Bellman-Ford's algorithm, requires known edge weights. Additionally, Dijkstra's algorithm requires the edge weights to be non-negative while Bellman-Ford requires no negative cycles.

To aggravate the problem further the feedback from traversing an edge is stochastic. Considering the road network, numerous factors may affect the time of traversing a road segment such as weather, accidents, congestion and style of driving.

More formally, given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{w})$. $\mathcal{V}$ denotes the set of vertices, and $\mathcal{E}$ denotes the set of edges. The set of edges $e = (u, v) \in \mathcal{E}$ denotes the edge between node u and node v. Each of the edges e $\in \mathcal{E}$ has an associated cost for traversing the edge weight $\mathbf{w}$. Each of the edges e $\in \mathcal{E}$, is associated with contextual features $\mathbf{x}_e$. For this thesis, the road network is modeled by the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{w})$, where each node $v \in \mathcal{V}$ represents an intersection. The edge $e = (u, v) \in \mathcal{E}$ represents a road segment connecting two vertices.

2.2 Multi-Armed Bandit problem

The Multi-Armed Bandit (MAB) problem is a classic problem that has gotten its name from an analogy to the type of slot machines at casinos often referred to as one-armed bandits. Here, instead of merely a single arm, an agent is repeatedly presented with a set of K arms to pull, each with its own distribution of rewards. At each time step t, the agent pulls one of them and receives a reward from its initially unknown probability distribution. The goal for the agent is to maximize the cumulative reward over a predefined time horizon T [5][6].

MABs provide an adaptive framework for problems where the trade-off between exploration and exploitation must be balanced deliberately. Sophisticated algorithms learn from the data collected over time and can learn to exclude choices with a low probability of being optimal. The framework allows for efficient exploration techniques in applications such as clinical trials, finance, recommender systems, and others [7][8].

2.2.1 Objective

The objective is to maximize the total expected rewards over the time horizon T. The agent needs to utilize a proper policy that balances the trade-off between exploiting—that is choosing the action that has shown to be best so far—and exploring potentially better actions. To achieve this, it must estimate the expected return for every action. Essentially, the agent is given K arms and T rounds. At each time step $t \in T$, the agent chooses an arm a_t, whereafter it receives a reward r_t drawn from the selected arm's reward distribution.

There are two essential assumptions. First, the agent only observes and receives the reward from the chosen actions, and gets no information regarding the rewards of the non-selected arms. Second, for all respective arms, their rewards are independent and identically distributed (i.i.d.) random variables. Every time an arm is pulled, or an action is taken, the rewards are sampled independently from this distribution.

Variable	**Description**
K	Number of arms
a_t	Arm or action selected at time t, the choice of a particular option
r_t	Reward or feedback obtained after selecting action a at time t
μ_a or $\mu(a)$	Expected reward of action a
$\hat{\mu}_a$ or $\hat{\mu}(a)$	Estimated value of the expected reward of action a
μ^* or $\mu(a^*)$	Expected reward of the optimal arm
N_a or $N(a)$	Number of times action a has been selected
t	Time step or trial (the round in which an action is selected)
T	Time horizon (the total number of time steps)

Table 2.1: Variables for Multi-Armed Bandits

In addition to the reward, another important metric for the bandit algorithms is the regret. The instantaneous regret R_t is calculated by subtracting the expected reward of the selected action $\mu(a_t)$ at time step t from the expected optimal reward μ^*. The metric that is generally used to evaluate bandit algorithms is the cumulative regret, which is calculated as the sum of instantaneous regrets over the time horizon T. More formally it is defined as

$$R(T) \triangleq \mu^* \cdot T - \mathbb{E}\left[\sum_{t=1}^{T} r_t\right] = \mu^* \cdot T - \sum_{t=1}^{T} \mu(a_t). \tag{2.1}$$

The Bayesian regret, for a problem instance $\mathcal{I}$, where $\mathbb{P}$ is the prior assumption, is defined as

$$BR(T) \triangleq \mathbb{E}_{\mathcal{I}\sim\mathbb{P}}\left[\mathbb{E}\left[R(T)|\mathcal{I}\right]\right] = \mathbb{E}_{\mathcal{I}\sim\mathbb{P}}\left[\mu^* \cdot T - \sum_{t}^{T}\left[T\right]\mu(a_t)\right]. \tag{2.2}$$

The major takeaway is that exploring actions that have a low probability of being optimal most likely increases regret, while under-exploration inhibits the process of identifying optimal actions which is detrimental for regret minimization in the long term. Therefore, information is gathered sequentially and is used to learn with sophisticated exploration-exploitation methods.

2.2.1.1 Theoretical regret bounds

As described in Section 2.2.1 the objective is to maximize the cumulative reward over the time horizon T, or as commonly referred to in the bandit terminology minimizing the cumulative regret [6].

Bandit algorithms commonly derive regret bounds [6] which is a theoretical description of the capabilities of the algorithms with a high degree of probability. The *lower* bound of the algorithm describes what the algorithm cannot achieve. Further, the *upper* bound limits how the algorithm performs in the worst-case scenario.

Formally, the gap between the optimal arm and the sub-optimal arm i is

$$\Delta_i = \mu^* - \mu_i. \tag{2.3}$$

The regret bounds can be *instance-dependent* or *instance-independent* [6]. The magnitude of the gap (Eq. (2.3)) can have a large impact on the regret bound for problem instances where multiple choices are close to optimal. For such problem instances, it may be beneficial with regret bounds not including the gap.

2.2.2 Bandit algorithms

2.2.2.1 Explore-first

The explore-first algorithm [6] consists of two phases, the exploration- and exploitation-phase. The first phase focuses on exploration where each arm is selected for a pre-arranged number of instances. Once the exploration phase is completed, the arm with the highest estimated reward is selected. Either the same arm can be chosen for the remaining time steps or until another arm has a higher estimated reward. This approach is simple but can be effective. For example, if the underlying probability distributions are highly concentrated around the expected rewards, one or a few samples could be sufficient. On the contrary, if the data changes over time or has a large variance, then explore-first may commit to sub-optimal actions. The

Algorithm 1 Explore-First Algorithm for Multi-Armed Bandits

```
Number of arms $K$, Number of initial pulls $N$,
for each time step $t = 1, 2, \ldots$ do
  if $t \leq KN$ then
    Exploration Phase: Select arm until all arms are selected N times
  else
    Exploitation Phase: Select the arm $a$ with the highest empirical reward estimate
  end if
  Pull arm $a$ and observe the reward $r_t$
end for
```

expected regret of explore first is

$$\mathbb{E}[R(T)] \leq T^{2/3} \times O(K\log T)^{1/3}. \tag{2.4}$$

A derivation of the regret bound is available in [6].

2.2.2.2 ϵ-greedy

The ϵ-greedy algorithm [6] utilizes uniform exploration throughout the time horizon. At each time-step t, the algorithm chooses an arm greedily with a probability of $1 - \epsilon$ and with a probability of ϵ it explores all arms uniformly. If the proportion of exploration is constant throughout the experiment, the regret is linear. However, a decreasing ϵ can provide a lower regret bound. For example, $\epsilon = 1/t$ allows a logarithmic regret bound [9]. Exploration probability $\epsilon_t = t^{-1/3}(K\log t)^{1/3}$ achieves the regret bound

$$\mathbb{E}[R(T)] \leq t^{2/3} \times O(K\log t)^{1/3}. \tag{2.5}$$

2.2.2.3 UCB1

The UCB1 algorithm [6], which is short for Upper Confidence Bound 1, employs the concept of *optimism under uncertainty.* Essentially, the algorithm stores two values

Algorithm 2 Epsilon-Greedy Algorithm

```
Initialize the estimated rewards for each arm: $\hat{\mu}_k \leftarrow 0$ for $k = 1, 2, \dots, K$
Initialize arm counts $N_k$ for each arm $a$
Initialize exploration parameter $\epsilon$
for $t = 1, 2, 3, \dots$ do
  if random number $r \leq \epsilon$ then
    Choose a random arm $a_t$ with equal probability among all arms
  else
    Choose the arm $a_t$ with the highest estimated value: $a_t = \arg\max_a \hat{\mu}(a)$
  end if
  Pull arm $a_t$ and observe the reward $r_t$
  Update arm counts: $N_{a_t} \leftarrow N_{a_t} + 1$
  Update arm-value estimate for $a_t$:
  $\hat{\mu}(a_t) \leftarrow \hat{\mu}(a_t) + \frac{1}{N_{a_t}} \cdot (r_t - \hat{\mu}(a_t))$
end for
```

for each arm a. The current expected reward, which is the mean of all previously observed rewards $\hat{\mu}_a$, and the number of times the arm has been played N_a.

Each arm has a lower and an upper bound which envelops the expected reward with high probability. For every time arm a has been pulled and N_a incremented, the range of the estimated expected reward concentrates. This encourages selecting arms with high mean rewards and/or not sufficiently explored. Hence, the algorithm gradually increases the exploitation of the arms with the highest mean as t increases. Regret bounds for UCB1 from [6] are

$$\mathbb{E}[R(T)] = O(\sqrt{Kt \log T}) \text{ for all rounds } t \leq T, \tag{2.6}$$

and

$$\mathbb{E}[R(T)] \leq O(\log T) \left[\sum_{\text{arms a with } \mu(a) \leq \mu(a^*)} \frac{1}{\mu(a^*) - \mu(a)} \right]. \tag{2.7}$$

Comparing the regret bounds Eq. (2.6) and Eq. (2.7) show that if there exists arms where $\mu(a) \approx \mu(a^*)$ the regret in Eq. (2.7) can become large as it will be difficult to exclude arms.

2.2.2.4 Thompson sampling

Thompson sampling (TS) is a Bayesian approach to the MAB problem where the algorithm estimates each arm's probability of being the optimal one. In practice, this means that it keeps track of a distribution over each arm's expected reward. Initially, the probability distributions are initialized with a prior belief of the distribution with parameters θ_a. At each time step the algorithm samples from the posterior distribution of each arm and then chooses the arm which returns the highest sampled value. The probability distribution for arm a is then updated with the feedback and the procedure is repeated.

Algorithm 3 UCB1 (Upper Confidence Bound 1) Algorithm

```
**Input**: Number of arms $K$, Number of rounds $T$
Initialize the estimated rewards for each arm: $\hat{\mu}_k \leftarrow 0$ for each arm $a_k$
Initialize arm counts $N_k$ for each arm $a$
for $t = 1, 2, \ldots, T$ do
    for $k = 1, 2, \ldots, K$ do
        Compute the Upper Confidence Bound $U$ for arm $k$:
        $U_k \leftarrow \hat{\mu}_k + \sqrt{\frac{2 \ln t}{N_k}}$
    end for
    Choose arm $a_t$ with the maximum UCB: $a_t \leftarrow \arg\max_k U_k$
    Observe reward $r_t$ for arm $a_t$
    Update estimated reward for arm $a_t$:
    $\hat{\mu}_{a_t} \leftarrow \frac{\hat{\mu}(a_t) \cdot N_{a_t} + r_t}{N_{a_t} + 1}$
    Increment the number of pulls for arm $a_t$: $N_{a_t} \leftarrow N_{a_t} + 1$
end for
```

In Bayesian inference, when data is sparse, the probability distributions are broad or uniform. As more data is gathered, these distributions gradually become more concentrated around their respective means μ_a. One significant advantage of TS is its inherent incorporation of uncertainty and incorporating it into the decision-making process. Arms that haven't been explored thoroughly will eventually be selected due to their wide distribution. Subsequently, these arms can either be eliminated or exploited based on the feedback.

Algorithm 4 Thompson Sampling Algorithm

```
**Input**: Number of arms $K$, Number of rounds $T$
Initialize the parameters $\theta$ for the prior distribution for each arm.
for $t = 1, 2, \ldots, T$ do
    for $k = 1, 2, \ldots, K$ do
        Sample reward from the posterior distribution $\mathcal{D}_k$ for arm $k$:
        $\hat{\mu}_k \sim \mathcal{D}_k$
    end for
    Choose arm $a_t$ with the highest sampled value: $a_t \leftarrow \arg\max_k \hat{\mu}_k$
    Observe reward $r_t$ for arm $a_t$
    Update the parameters $\theta_k$ of the distribution for arm $a_k$ selected at time step $t$:
end for
```

The Bayesian Regret [6] for Thompson sampling is

$$BR(T) \leq O(\sqrt{KT \log T}). \tag{2.8}$$

Finally, TS enables the use of informative priors where the arms could be initialized based on prior beliefs or initial assumptions regarding the underlying reward distributions $\mathcal{D}_k$. The prior information could help with the initial exploration of different arms. Informative priors can be used in domains where expert knowledge or data gathered is already available. On the other hand, uninformative priors can be used where neither expert knowledge nor data is available.

2.2.2.5 Bayesian UCB

Another Bayesian approach to the MAB problem is the Bayesian Upper Confidence Bound algorithm [10], or BayesUCB for short. As the name suggests, this algorithm makes use of the *optimism in the face of uncertainty* principle for efficient exploration, similar to UCB1. However, adopting a Bayesian framework, BayesUCB relies on predefined prior probability distributions over all the available arms, and the successive updating of the posterior distributions using observed rewards.

The difference compared with Thompson Sampling is therefore that instead of sampling rewards from the arms, BayesUCB calculates confidence bounds based on quantiles of their respective posterior distributions. The quantile function $Q(\beta, \lambda)$ over a distribution λ is defined such that the probability that a random variable distributed according to λ is less than $Q(\beta, \lambda)$ equals β.

What BayesUCB does is to select the arm a which maximizes $Q(\beta, \lambda_a)$, where λ_a is the posterior distribution of the expected reward from arm a and β is shared among all arms. To obtain a suitable upper bound in practice, [10] suggests selecting $\beta = 1 - \frac{1}{t(\log T)^c}$, where t is the current time step, T the time horizon, and c a constant. For cases where the goal is to minimize the returns from the arms, as in [11], a corresponding optimistic estimate is instead obtained by setting $\beta = \frac{1}{t(\log T)^c}$.

Algorithm 5 Bayesian Upper Confidence Bound Algorithm

```
**Input**: Number of arms $K$, Number of rounds $T$, Quantile parameter $c$
Initialize the parameters for the prior distribution $\lambda_a$ for each arm $a$
**for** $t = 1, 2, \ldots, T$ **do**
    **for** $k = 1, 2, \ldots, K$ **do**
        Calculate the upper confidence bound as a quantile of the posterior distribution for arm $k$:
        $U_k = Q\left(1 - \frac{1}{t(\log T)^c}, \lambda_{a_k}\right)$
    **end for**
    Choose arm $a_t$ with the highest upper confidence bound: $a_t \leftarrow \arg\max_k U_k$
    Observe reward $r_t$ for arm $a_t$
    Update the parameters of the posterior distribution $\lambda_{a_t}$ for arm $a_t$
**end for**
```

2.2.3 Contextual Multi-Armed Bandit Problem

Many of the more simplistic non-contextual algorithms such as explore-first, ϵ-greedy, UCB1 and TS may have issues with complex real-world situations where the underlying probability distributions may be non-stationary. Uniform exploration followed by greedy action selection, for example, may work well for the initial data distribution. However, if the underlying process generating the data is non-stationary due to seasonality or other fluctuations, the simplifications of expecting stationary distributions may be troublesome. To control for such fluctuations, the MAB framework can be extended into *contextual multi-armed bandits*. Before choosing an arm and receiving the corresponding reward r_t, the agent obtains contextual features for every arm for the current time step t. These features are commonly organized into a feature vector, where each component represents a scalar value of some quantity which partly determines the expected reward.

The formal framework considered here is adopted from [12], with slight modifications to match the scope of this project, and is standard for contextual bandit problems. A contextual bandit algorithm A, also called an agent, proceeds in discrete trials $t = 1, 2, 3, \ldots$

In trial t:

1. The agent observes a set $\mathcal{A}_t$ of arms or actions together with their feature vectors $x_{t,a}$ for $a \in \mathcal{A}_t$. The vector $\mathbf{x}_{t,a}$ summarizes information of arm a and global information common to all arms. $x_{t,a}$ will be referred to as the context vector, or simply the context.
2. Based on observed payoffs in previous trials, A chooses an arm $a_t \in \mathcal{A}_t$, and receives payoff r_t whose expectation depends on both the global, and the arm-specific context.
3. The agent then improves its arm-selection strategy with the new observation, $(\mathbf{x}_{t,a_t}, a_t, r_t)$.

The way in which the context is processed differs depending on the particular contextual bandit algorithm. However, the goal of the agent is always to utilize the contextual information to guide the process of balancing exploration and exploitation, in order to minimize the regret over the time horizon T. Note that payoff r_t only contains information about the one chosen arm in each trial, and that no feedback is observed for the other arms $a \neq a_t$.

2.2.4 Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational models that draw inspiration from the biological networks of neurons in the brains of humans and other animals. Their fundamental building block, the neuron, computes a weighted sum of numerical inputs and applies an activation function to this sum to produce its output. These neurons can be arranged in parallel and series to form networks of various configurations. The most common layout is the fully connected feed-forward neural network, where neurons are placed in layers, and where the output of the neurons in one layer becomes the inputs to all of the neurons in the next one [13]. Unless stated otherwise, we will let the term neural networks refer to this particular architecture.

Fully connected artificial neural networks are commonly used as function approximators. In fact, all continuous functions can be approximated to any level of accuracy with a fully connected neural network with a sufficient number of neurons in one of its layers [14]. With this in mind, it makes sense that these computational models are popular for regression problems. However, they have shown tremendous performance on many classification tasks as well [15]. The versatility of ANNs is made possible by a large number of tunable parameters, called weights and biases, which control the computation of the weighted sums in each neuron in the network.

To find appropriate values of the weights and biases for the particular function at hand, also known as the objective function, the network is fed with known input-output pairs and trained to minimize the difference between the objective function output and network prediction given the input example. The outputs may be subject to noise, which complicates the procedure of finding the underlying objective function during training. The difference calculation can be defined in various ways, and the metric used is commonly referred to as the loss function, which generally also incorporates some form of regularization. Although different methods exist for training ANNs given some input-output data, the most popular one is called error back-propagation, which is outlined in algorithm 6.

Formally, a fully connected ANN is defined as

$$f(\mathbf{x};\boldsymbol{\theta}) = \mathbf{W}_L\sigma\Big(\mathbf{W}_{L-1}\sigma\Big(...\sigma(\mathbf{W}_1\mathbf{x})\Big)\Big), \tag{2.9}$$

where L is the number of layers, σ is the activation function, W_l is the weight matrix, which encodes the weights and biases of layer l, $\boldsymbol{\theta}$ is a vector of all tunable parameters in the network, and $\mathbf{x}$ is the input vector [16].

Algorithm 6 Train ANN with error back-propagation

```
Require: Set of input vectors $\{\mathbf{x}_i\}$, set of target outputs $\{y_i\}$, initial network parameters $\boldsymbol{\theta}^{(0)}$, learning rate $\eta$, regularization parameter $\lambda$, number of training epochs $N$, loss function $\mathcal{L}$.
1: for $i = 1$ to $N$ do
2:     $\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \eta\nabla\mathcal{L}\Big(\{\mathbf{x}_i\};\boldsymbol{\theta}^{(i)}\Big)$.
3: end for
4: return Final network parameters: $\boldsymbol{\theta}^{(N)}$.
```

2.2.5 Decision Trees

Decision trees were first proposed in 1963 as a way of accounting for interaction effects in multi-variate data analysis for prediction tasks [17]. Similar to ANNs, decision trees can be employed for both regression and classification tasks, but the two types of computational models differ fundamentally. While ANN's construct continuous functions, decision trees are inherently discrete, as seen from how they are designed.

A decision tree is made up of *nodes* that are organized in a hierarchical manner, where each node has one single *parent node*, and a set of *child nodes*. Generally, the number of child nodes is capped at two, which results from the nodes giving rise to *binary splits*. A binary split represents the partitioning of the space of allowed inputs into two disjoint sets, based on a single input variable. The prediction made by the decision tree on a data point is made by starting at the top node, which is also referred to as the *root node*, and traversing the tree down one level at a time, at each of which the data point is assigned a child node based on its feature values and the tree nodes' splitting rules. Once a data point reaches a node that does not have any children, also known as a *leaf node*, it is assigned a prediction value that is associated with that leaf node.

The interpretation of this value can differ depending on whether the decision tree is used for classification or regression, and on the particular procedure of building the tree, but it is chosen to represent the training data samples assigned to the same leaf. Hence, what a decision tree does is essentially to partition the input space into smaller and smaller subsets the deeper the tree is, in order to group the inputs into regions where the data appears to be similar.

In decision trees used for regression, which are often referred to simply as *regression trees*, the leaf node values are most often generated by calculating the arithmetic mean of the target value of the training samples that lie in the subset associated with that leaf node. The goal, then, is often to minimize the variance of target values from training data samples in a leaf, to obtain accurate predictions. However, one would also like to ensure that the model generalizes well to unseen data and is not sensitive to noise in the target values. On the one hand, a decision tree is sometimes required to be deep to account for complex interactions and relationships in the data. On the other hand, finer and finer partitions increase the risk of fitting it to noise. In practice, this often turns out to be challenging, as decision trees are infamous for notoriously overfitting to the training data [18].

2.2.6 Ensemble methods

Ensemble methods are machine learning techniques for solving supervised learning tasks utilizing multiple models which combined produce a higher predictive performance compared to single complex models. The concept of ensembles considers the models within the ensemble as *weak* or *base learners*—as they individually possess little predictive power—whereas the aggregated models can provide solid predictions. The collective prediction made by simple models together mitigates bias relatively well compared to having only one expressive model, which may be more prone to overfitting. Ensemble methods are commonly implemented using *boosting* or *bagging*, introduced in sections 2.2.6.1 and 2.2.6.2. Bagging has shown good performance built on high-variance low-bias weak learners and boosting commonly outperforms bagging [19]. Thus, decision trees are commonly used as the learners in an ensemble. However, the concept of ensembles is general and not limited to decision trees.

2.2.6.1 Bagging

Bootstrap aggregating [20], or bagging for short, is a commonly used ensemble technique, which utilizes the concept of randomly resampling the data points without replacement to provide weak learners with diversity. The subset can be a subsampling of the features or instances available in the training data. Sampling without replacement encourages the learners to focus on the important parts of their subset of the data. Each learner in an ensemble predicts an output $\hat{y}_i$ based on the observed data, and the ensemble predicts the final output according to

$$\hat{y}_{\text{ensemble}} = \sum_{i=1}^{N} w_i \hat{y}_i. \tag{2.10}$$

The weight of each learner w_i can be uniformly set to $1/N$ or based on the performance of the learners from validation such as out-of-bag errors, which are the errors made on the data points which were not selected in the random resampling when building a particular tree. The only requirement is that the sum of weights equals 1. One advantageous property of tree ensembles based on bagging is that as each model in the ensemble is trained individually on different subsets, it enables parallel processing.

2.2.6.2 Boosting

Boosting [20] is an iterative technique where the training objective is based on the errors of the previous learner. The weak learner's data points are weighted such that previous data points that have been predicted poorly are assigned higher weights. This forces the model to put more emphasis on these errors. As boosting algorithms are trained in a sequential manner it does not allow parallel computing in a natural way, which stands in contrast to bagging methods.

Gradient-boosting is a supervised machine-learning technique that—as the name suggests—draws on the general concepts of boosting. Similar to boosting and bagging, its predictions are based on multiple weak learners. In practice, decision trees are commonly used as the weak learner due to their simplicity, speed of training, and not requiring normalized, standardized, or even numerical inputs [19]. Decision trees can naturally interact with datasets with mixed data types, values are missing, and scale well with large datasets.

Gradient-boosted trees are trained sequentially, with the new tree constructed to handle the misclassification of its predecessors. Essentially, this means that the newly added tree does not explicitly try to predict the target variable. Instead, the prediction is based on the errors of the previous model or set of models.

Essentially, it is an ensemble method that combines several weak learners making it possible to detect complex relationships in the training data. Bagging creates several decision trees where adding new trees should improve performance by reducing the variance. On the contrary, boosting starts with one weak learner which has a high bias predicting a constant value, and the subsequent learners form the decision boundaries.

2.2.6.3 Random forest

Random forest (RF) is another ensemble method that utilizes decision trees as the weak learners [21]. The mechanics of constructing a random forest is highly similar to that of bagging, but introduces a few additional details. The trees are pruned to a limited depth to enforce the weakness of the learners and avoid overfitting. Furthermore, only a subset of the contextual features available are considered when deciding on the split criterion of a non-terminal node. This encourages the trees of the ensemble to learn different relationships present in the data, in order to build trees with little correlation between each other.

Algorithm 7 Random Forest Algorithm

```
Require: Training dataset $(X, y)$, Number of trees: $T$, number of features to consider at each split $m$
1: Initialize an empty set of decision trees $F$
2: for $t = 1$ to $T$ do
3:     Randomly select $m$ features from the dataset
4:     Randomly sample $n$ examples from the dataset with replacement
5:     Create a decision tree $T_t$ using the selected features and sampled examples
6:     Add $T_t$ to $F$
7: end for
8: return Ensemble of decision trees $F$
```

2.2.6.4 XGBoost

XGBoost, eXtreme Gradient Boosting [22], is an optimized gradient-boosted decision tree method used for regression, classification, and ranking problems [22]. The algorithm thrives on large complex data sets and has been proven efficient at various prediction tasks [23].

Initially, the first predictor predicts a constant value for all observations, for both classification and regression, and the gradient of the loss function is calculated for each observation. The way the gradients are calculated depends on which loss function the model is evaluated on. In practice, mean squared error (MSE) is commonly used for regression and binary cross-entropy loss or log-loss—both are identical and the names are used interchangeably—is used for classification tasks. Then, a new weak learner is trained to counteract the errors from the previous tree.

The new tree gets added with a learning rate ν, and the complete predictor is updated. The leaves in the tree are evaluated on a similarity score which is calculated as the sum of all residuals squared. The model can then compare the evaluated threshold where the data is split by S_{right} and S_{left} where S is the similarity score. The quality of the split is evaluated on the gain, where the gain is calculated based on the similarity scores in the leaves, and the output of the leaf is based on the sum of the residuals. For regression, when using MSE loss, the values are calculated as follows:

$$S = \frac{\left(\sum \text{residuals}\right)^2}{\text{number of residuals} + \lambda}, \tag{2.11}$$

$$\text{gain} = S_{\text{right}} + S_{\text{left}} - S_{\text{root}}, \tag{2.12}$$

$$\text{output} = \frac{\left(\sum \text{residuals}\right)}{\text{number of residuals} + \lambda}. \tag{2.13}$$

γ and λ are two regularization parameters that can be used to control the model and avoid overfitting. When a tree has been completed, a check is performed on the data set to make sure the gain of the split performed in each respective node is significant enough. If the gain of a split is smaller than γ, the corresponding node is discarded and the whole branch following it is pruned off the tree. The output value for a node is calculated as the sum of the residuals divided by the total number of residuals plus λ. The regularization parameter λ makes the prediction less sensitive to individual observations.

Compared to vanilla gradient-boosted decision trees, XGBoost allows for parallelization which can have a high impact on the efficiency with large data sets. Further, it can utilize many different types of optimization techniques, such as approximate greedy algorithm, weighted quantile splitting, and cache-aware access [22].

Algorithm 8 XGBoost Regression

```
Require: Training data: $(X, y)$, Number of trees: $T$, Learning rate: $\eta$
1: Initialize model: $F_0(x) = 0.5$
2: for $t = 1$ to $T$ do
3:     Compute negative gradient: $g_t = -\frac{\partial}{\partial F_{t-1}} \sum_{i=1}^{N} L(y_i, F_{t-1}(x_i))$
4:     Fit a weak learner to the negative gradient: $h_t = \text{argmin}_h \sum_{i=1}^{N} L(y_i, F_{t-1}(x_i) + \eta h(x_i))$
5:     Update the model: $F_t(x) = F_{t-1}(x) + \eta h_t(x)$
6: end for
7: return Final model: $F_T(x)$
```

2.2.7 Contextual Bandit Algorithms

As discussed in section 2.2.3, the way in which the additional information available to an agent in the contextual multi-armed bandit problem is utilized depends on the specific algorithm, or agent. Therefore, the choice of algorithm is largely dependent on the particular problem at hand. With knowledge of how the rewards depend on the contextual features, one may choose an agent that can explore the set of arms in an efficient manner and by that achieve a low level of regret. On the other hand, if the anticipated relationship between contexts and rewards turns out not to hold, imposing such models may do more harm than good. Therefore, in an instance where little is known about the reward function and its dependence on the available information, one is generally better off by selecting a versatile and adaptive model.

When deriving theoretical bounds on the performance of contextual multi-armed bandit methods, it is often necessary to make assumptions on how the reward function depends on the arms' contextual features. Some algorithms are highly specialized for a narrow class of reward functions they were specifically designed for, and work well with, while others are more general. In the subsections that follow, we describe a selection of contextual multi-armed bandit algorithms.

2.2.7.1 Linear Upper Confidence Bound

The LinUCB [12] algorithm, which stands for Linear Upper Confidence Bound, makes the key assumption that the expected reward of an arm a is a linear function of the feature vector in each trial:

$$\mathbb{E}[r_{t,a}|\mathbf{x}_{t,a}] = \mathbf{x}_{t,a}^T \boldsymbol{\theta}_a^*. \tag{2.14}$$

This allows the agent to utilize the contextual information available in each trial of a contextual MAB. Successively, it then estimates the unknown coefficient vector as $\hat{\boldsymbol{\theta}}_a$ through ridge regression [24] applied to the data it observes of an arm:

$$\hat{\boldsymbol{\theta}}_a = (\mathbf{D}_a^T \mathbf{D}_a + \mathbf{I}_d)^{-1} \mathbf{D}_a^T \mathbf{c}_a. \tag{2.15}$$

Here, $\mathbf{I}_d$ is the $d \times d$ identity matrix, $\mathbf{c}_a$ the m-dimensional vector of rewards from choosing arm a m times, and $\mathbf{D}_a$ is an $m \times d$ design matrix containing the feature vectors of arm a from those trials. A major advantage of the model assumption of linear rewards is that the observed data can also be used to estimate confidence bounds in the predictions effectively. It turns out that with a probability of at least $|1-\delta|$,

$$\left| \mathbf{x}_{t,a}^T \hat{\boldsymbol{\theta}}_a - \mathbb{E}[r_{t,a}|\mathbf{x}_{t,a}] \right| \leq \alpha \sqrt{\mathbf{x}_{t,a}^T (\mathbf{D}_a^T \mathbf{D}_a + \mathbf{I}_d)^{-1} \mathbf{x}_{t,a}}, \tag{2.16}$$

for any $\delta > 0$ and $\mathbf{x}_{t,a} \in \mathbb{R}^d$, with constant $\alpha = 1 + \sqrt{\ln(2/\delta)/2}$ ([25]). This gives the algorithm a natural way of calculating upper confidence bounds for all arms, and the selection in trial t is made such that

$$a_t \overset{\text{def}}{=} \underset{a}{\operatorname{argmax}} \left(\mathbf{x}_{t,a}^T \hat{\boldsymbol{\theta}}_a + \alpha \sqrt{\mathbf{x}_{t,a}^T \boldsymbol{\Gamma}_a^{-1} \mathbf{x}_{t,a}} \right), \tag{2.17}$$

where $\boldsymbol{\Gamma}_a \overset{\text{def}}{=} \mathbf{D}_a^T \mathbf{D}_a + \mathbf{I}_d$. Hence, LinUCB provides a nice and relatively simple extension of the popular UCB methods for contextual MAB problems. There is a slightly different version of LinUCB [26], which rather than employing a separate linear model for each arm keeps track of merely a single one for all of them. One major benefit of this alternative approach is that less memory is required to store the models. Another is that it allows for using data gathered on one arm can be utilized for inferring information about another arm. A problem with this second approach, however, is that it assumes that the coefficients θ are shared for the reward functions of the different arms, which may not be the case.

Algorithm 9 LinUCB Algorithm

```
Input: $\alpha \in \mathbb{R}_+$ (exploration factor)
for $t = 1, 2, 3, \dots T$ do
  for $k = 1, 2, 3, \dots, K$ do
    if $t = 1$ then
      $\mathbf{\Gamma}_k \leftarrow \mathbf{I}_d$ ($d$-dimensional identity matrix)
      $\mathbf{b}_k \leftarrow \mathbf{0}_{d \times 1}$ ($d$-dimensional zero vector)
    end if
    Observe the feature vector $\mathbf{x}_{t,k} \in \mathbb{R}^d$
    Compute the estimated model parameter vector $\hat{\boldsymbol{\theta}}_k \leftarrow \mathbf{\Gamma}_k^{-1} \mathbf{b}_k$
    Compute the estimated reward mean $\hat{\mu}_{t,k} \leftarrow \mathbf{x}_{t,k}^T \hat{\boldsymbol{\theta}}_k$
    Compute the confidence bound $\sigma_{t,k} \leftarrow \alpha \sqrt{\mathbf{x}_{t,k}^T \mathbf{\Gamma}_k^{-1} \mathbf{x}_{t,k}}$
  end for
  Select the arm with the highest upper confidence bound:
    $a_t \leftarrow \arg\max_k (\hat{\mu}_{t,k} + \sigma_{t,k})$, with ties broken arbitrarily
  Observe the reward $r_t \in \mathbb{R}$ for the chosen arm $a_t$
  Update the model variables based on the context and reward:
    $\mathbf{\Gamma}_{a_t} \leftarrow \mathbf{\Gamma}_{a_t} + \mathbf{x}_{t,a_t} \mathbf{x}_{t,a_t}^T$
    $\mathbf{b}_{a_t} \leftarrow \mathbf{b}_{a_t} + r_t \mathbf{x}_{t,a_t}$
end for
```

2.2.7.2 Linear Thompson Sampling

Similar to LinUCB, Linear Thomson Sampling, also known as LinTS [27], makes the assumption that the expected rewards obtained from any action taken is from a function that is linear in the contextual features $\mathbf{x}_{t,a}$, with coefficients θ_a^* that may be different for all arms. The main difference between the two algorithms is—as the name implies—that LinTS grounds its exploration strategy in the concept of Thompson Sampling. Hence, an important part of the algorithm is to estimate expected rewards through sampling. For LinTS, this is done through sampling of the model parameters $\hat{\theta}_a$ from a normal distribution that is adapted to the observed contexts and rewards.

The version of LinTS in [27] employs a single linear model to use for all arms, and therefore only samples one vector of model parameters per time step. Just as for LinUCB 2.2.7.1, however, one may also consider a version of LinTS which employs a separate linear model for each arm.

2.2.7.3 TreeBootstrap

Similar to the versions of LinUCB and LinTS described above, TreeBootstrap [28] employs a different predictive model for every arm in the set of available arms, in order to estimate their expected rewards. Here, however, instead of assuming a linear relationship between contexts and rewards, TreeBootstrap takes a more general approach, and assigns a decision tree to every arm. In each time step, the decision trees are trained on context-reward pairs previously observed from the corresponding arms. Then, the fitted trees are used to predict the rewards that

Algorithm 10 LinTS Algorithm

```
Input: $v \in \mathbb{R}_+$ (exploration factor)
for $t = 1, 2, 3, \ldots T$ do
  for $k = 1, 2, 3, \ldots, K$ do
    if $t = 1$ then
      $\mathbf{\Gamma}_k \leftarrow \mathbf{I}_d$ ($d$-dimensional identity matrix)
      $\mathbf{b}_k \leftarrow \mathbf{0}_{d\times 1}$ ($d$-dimensional zero vector)
    end if
    Observe the feature vector $\mathbf{x}_{t,k} \in \mathbb{R}^d$
    Sample the estimated model parameter vector $\hat{\boldsymbol{\theta}}_k \sim \mathcal{N}(\mathbf{\Gamma}_k^{-1}\mathbf{b}_k,\, v^2\mathbf{\Gamma}^{-1})$
    Compute the estimated reward mean $\hat{\mu}_{t,k} \leftarrow \mathbf{x}_{t,k}^T\hat{\boldsymbol{\theta}}_k$
  end for
  Select the arm with the highest sampled mean reward:
    $a_t \leftarrow \arg\max_k (\hat{\mu}_{t,k})$, with ties broken arbitrarily
  Observe the reward $r_t \in \mathbb{R}$ for the chosen arm $a_t$
  Update the model variables based on the context and reward:
    $\mathbf{\Gamma}_{a_t} \leftarrow \mathbf{\Gamma}_{a_t} + \mathbf{x}_{t,a_t}\mathbf{x}_{t,a_t}^T$
    $\mathbf{b}_{a_t} \leftarrow \mathbf{b}_{a_t} + r_t\mathbf{x}_{t,a_t}$
end for
```

will be observed in that time step, should a specific arm a be selected, based on its current contextual information encoded in $x_{t,a}$.

As for the selection strategy, TreeBootstrap plays the arm associated with the highest tree prediction. It may seem like this is a purely greedy strategy, which should raise concerns about ensuring sufficient exploration. However, this is accounted for through the process of bootstrapping—hence the name TreeBootstrap. In practice, this means that the trees are not trained on the whole set of previously observed data points $D_{t,a}$, but instead on a data set $\tilde{D_{t,a}}$ obtained through sampling with replacement. In [28], the creators of the algorithm draw a parallel between this way of handling the exploration-exploitation dilemma and Thompson Sampling. They also note that the suggested bootstrapping framework does not necessarily require decision trees to function, and can be used with other base learners.

Algorithm 11 TreeBootstrap Algorithm

```
for $t = 1, 2, 3, \ldots T$ do
  for $k = 1, 2, 3, \ldots, K$ do
    Observe the feature vector $\mathbf{x}_{t,k} \in \mathbb{R}^d$
    Sample bootstrapped data set $\tilde{D}_{t,k}$ from $D_{t,k}$
    Fit decision tree $\tilde{F}_{t,k}(\cdot\,; \{\cdot,\cdot\})$ to $\tilde{D}_{t,k}$
  end for
  $a_t \leftarrow \arg\max_k \tilde{F}_{t,k}(x_t,\,; \tilde{D}_{t,k})$, with ties broken arbitrarily
  Play $a_t$ and observe reward $r_{t,a_t}$
  Update $D_{t,a_t}$ with $(x_{t,a_t}, r_{t,a_t})$
end for
```

2.2.7.4 Neural Upper Confidence Bound

For problems where the reward of different arms is correlated, given their contextual features, bandit methods which assigns a separate model per arm may fail to make effective use of the additional information provided in the problem. In such cases, more general methods are needed. One such method is NeuralUCB [16], which assumes that

$$\mathbb{E}[r_{t,a}|\mathbf{x}_{t,a}] = h(\mathbf{x}_{t,a}), \tag{2.18}$$

where $h()$ is some bounded function. Note that $h()$ only depends on the context vector, and would assign the same expected reward for two arms with identical features.

To estimate the function $h()$, NeuralUCB employs a fully connected artificial neural network with n neurons in its hidden layers, equipped with the ReLU [29] activation function σ:

$$f(\mathbf{x}_{x,t};\boldsymbol{\theta}) = \sqrt{n}\mathbf{W}_L\sigma\Big(\mathbf{W}_{L-1}\sigma\Big(\dots(\mathbf{W}_1\mathbf{x})\Big)\Big). \tag{2.19}$$

Here, $\mathbf{W}_1 \in \mathbb{R}^{n\times d}$, $\mathbf{W}_i \in \mathbb{R}^{n\times n}$, $2 \leq i \leq L-1$, and $\mathbf{W}_L \in \mathbb{R}^n$ are the parameter matrices corresponding to each layer of the ANN. $\boldsymbol{\theta} \in \mathbb{R}^{nd+n^2(L-2)+n}$ represents a vector of the concatenated flattened parameter matrices. As the name suggests, the exploration strategy of NeuralUCB is based on the upper confidence bound method. To construct the confidence bound given a feature vector $\mathbf{x}_{t,a}$, the algorithm makes use of the neural network gradient $g()$ with respect to $\boldsymbol{\theta}$:

$$g(\mathbf{x}_{t,a};\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} f(\mathbf{x}_{t,a};\boldsymbol{\theta}) \in \mathbb{R}^p. \tag{2.20}$$

Using this gradient, the NeuralUCB agent makes its selection as per

$$a_t \overset{\text{def}}{=} \underset{a}{\operatorname{argmax}} \left(f(\mathbf{x}_{x,t};\boldsymbol{\theta}) + \gamma_{t-1}\sqrt{\frac{g(\mathbf{x}_{t,a};\boldsymbol{\theta})^T\mathbf{Z}_{t-1}^{-1}g(\mathbf{x}_{t,a};\boldsymbol{\theta})}{n}} \right), \tag{2.21}$$

where

$$\mathbf{Z}_t = \frac{1}{n}\sum_{i=1}^{t} g(\mathbf{x}_{t,a_t};\boldsymbol{\theta})g(\mathbf{x}_{t,a_t};\boldsymbol{\theta})^T, \tag{2.22}$$

and γ_t is a parameter to be computed (see algorithm 12). The artificial neural network is trained through error back-propagation on the data it collects in the form of context-reward pairs $(\mathbf{x}_{t,a_t}, r_t)$. One fundamental difference of the NeuralUCB algorithm as compared to LinUCB, other than the particular model used to estimate the reward function, is the fact that NeuralUCB only maintains one single reward estimator for all arms. While LinUCB keeps track of a unique estimator for each arm present in the set of available ones, NeuralUCB aims to generalize its neural network enough to be able to accurately predict the reward of each arm in every time step.

Algorithm 12 NeuralUCB Algorithm

```
**Input:** $T$ (number of rounds), $\lambda$ (regularization parameter), $\nu$ (exploration parameter), $\delta$ (confidence parameter), $S$ (norm parameter), $n$ (network width)
Initialize: $\boldsymbol{\theta}_0$ randomly
Initialize: $\mathbf{Z}_0 \leftarrow \lambda \mathbf{I}_n$
Initialize: $\gamma_0 \leftarrow 0$
for $t = 1, 2, 3, \ldots T$ do
    for $k = 1, 2, 3, \ldots, K$ do
        Observe the feature vector $\mathbf{x}_{t,k} \in \mathbb{R}^d$
        Compute the estimated reward mean $\hat{\mu}_{t,k} \leftarrow f(\mathbf{x}_{x,t}; \boldsymbol{\theta}_{t-1})$
        Compute the confidence bound $\sigma_{t,k} \leftarrow \gamma_{t-1}\sqrt{g(\mathbf{x}_{t,a}; \boldsymbol{\theta}_{t-1})^T \mathbf{Z}_{t-1}^{-1} g(\mathbf{x}_{t,a}; \boldsymbol{\theta}_{t-1})/n}$
    end for
    Select the arm with the highest upper confidence bound:
        $a_t \leftarrow \arg\max_k (\hat{\mu}_{t,k} + \sigma_{t,k})$, with ties broken arbitrarily
    Observe the reward $r_t \in \mathbb{R}$ for the chosen arm $a_t$
    Compute $\mathbf{Z}_t \leftarrow \mathbf{Z}_{t-1} + g(\mathbf{x}_{t,a_t}; \boldsymbol{\theta}_{t-1}) g(\mathbf{x}_{t,a_t}; \boldsymbol{\theta}_{t-1})^T/n$
    Set $\boldsymbol{\theta}_t$ by training the neural network on all observed context-reward pairs
    Compute
    $\gamma_t \leftarrow \sqrt{1 + C_1 n^{-1/6}\sqrt{\log n} L^4 t^{7/6} \lambda^{-7/6}}$
        $\cdot \left(\nu\sqrt{\log\frac{\det \mathbf{Z}_t}{\det \lambda \mathbf{I}} + C_2 n^{-1/6}\sqrt{\log n} L^4 t^{5/3} \lambda^{-1/6} - 2\log\delta} + \sqrt{\lambda} S\right)$
        $+ (\lambda + C_3 t L)\left[(1-\eta n \lambda)^{J/2}\sqrt{t/\lambda} + m^{-1/6}\sqrt{\log n} L^{7/2} t^{5/3} \lambda^{-5/3}(1 + \sqrt{t/\lambda})\right.$
end for
```

2.2.7.5 Neural Thompson Sampling

Neural Thompson Sampling, also referred to as NeuralTS [30], is a contextual MAB algorithm that closely resembles NeuralUCB. There is a fundamental difference, however. Instead of acting according to the optimism under uncertainty principle, NeuralTS employs Thompson Samping to handle the exploration of arms efficiently. Practically, the fundamental difference compared to NeuralUCB is found in how the neural network gradient is interpreted, which affects the arm selection process. In each trial, and for every arm, NeuralTS first calculates a variance from the network gradient:

$$\sigma_{t,a}^2 = \frac{\lambda g(\mathbf{x}_{t,a}; \theta)^T \mathbf{Z}_{t-1}^{-1} g(\mathbf{x}_{t,a}; \theta)}{n}, \tag{2.23}$$

where

$$\mathbf{Z}_t = \frac{\sum_{i=1}^{t} g(\mathbf{x}_{t,a_t}; \theta)^T g(\mathbf{x}_{t,a_t}; \theta)}{n}, \tag{2.24}$$

just like in NeuralUCB. Next, estimated reward values are sampled for each arm like

$$\tilde{r}_{t,k} \sim \mathcal{N}\Big(f(\mathbf{x}_{t,a}; \boldsymbol{\theta}), \nu^2 \sigma_{t,k}^2\Big), \tag{2.25}$$

where $\mathcal{N}$ denotes a normal distribution and ν is an exploration parameter.

Having sampled estimated reward values for each arm this way, the algorithm selects one arm according to:

$$a_t \overset{\text{def}}{=} \underset{a}{\operatorname{argmax}}\, \tilde{r}_{t,a}. \tag{2.26}$$

Algorithm 13 NeuralTS Algorithm

```
Input: $T$ (number of rounds), $\lambda$ (regularization parameter), $\nu$ (exploration parameter), $n$ (network width)
Initialize $\theta_0$ randomly
Initialize $\mathbf{Z}_0 \leftarrow \lambda \mathbf{I}_n$
for $t = 1, 2, 3, \ldots T$ do
    for $k = 1, 2, 3, \ldots, K$ do
        Observe the feature vector $\mathbf{x}_{t,k} \in \mathbb{R}^d$
        Compute the estimated reward mean $\hat{\mu}_{t,k} \leftarrow f(\mathbf{x}_{x,t}; \theta)$
        Compute the variance $\sigma^2_{t,k} \leftarrow \lambda g(\mathbf{x}_{t,a}; \theta)^T \mathbf{Z}^{-1}_{t-1} g(\mathbf{x}_{t,a}; \theta)/n$
        Sample estimated reward $\tilde{r}_{t,k} \sim \mathcal{N}\Big(f(\mathbf{x}_{t,a}; \boldsymbol{\theta}), \nu^2 \sigma^2_{t,k}\Big)$
    end for
    Select the arm with the highest upper confidence bound:
        $a_t \leftarrow \arg\max_k \tilde{r}_{t,k}$, with ties broken arbitrarily
    Observe the reward $r_t \in \mathbb{R}$ for the chosen arm $a_t$
    Compute $\mathbf{Z}_t \leftarrow \mathbf{Z}_{t-1} + g(\mathbf{x}_{t,a_t}; \boldsymbol{\theta}_{t-1}) g(\mathbf{x}_{t,a_t}; \boldsymbol{\theta}_{t-1})^T/n$
    Set $\boldsymbol{\theta}_t$ by training the neural network on all observed context-reward pairs
end for
```

2.2.7.6 Tree Ensemble Upper Confidence Bound

Tree Ensemble Upper Confidence Bound (TEUCB) [31] is a contextual MAB algorithm which utilizes tree ensembles for prediction, and a UCB method to select actions with the concept of optimism under uncertainty in mind.

There are adaptations of UCB such as UCB1 [32], described in Section 2.2.2.3, which construct confidence bounds by encapsulating the expected reward for each action depending on the number of times the action has been chosen. One downside of the method is that the variance of the rewards is neglected. To combat this, variations of the UCB1 algorithm which include variance estimates are also suggested in [32]. These extended algorithms are called UCB1-Tuned and UCB1-Normal, where the former expects Bernoulli distributed rewards, while the latter expects Gaussian rewards. At a time step t UCB1-Normal selects the arm a_t with the maximal upper confidence bound $U_{t,a}$, according to

$$U_{t,a} = \tilde{\mu}_{t,a} + \sqrt{16\tilde{\sigma}^2_{t,a} \frac{\ln(t-1)}{n_{t,a}}}. \tag{2.27}$$

Here, $n_{t,a}$ is the number of times action a has been chosen, and $\tilde{\mu}_{t,a}$ and $\tilde{\sigma}^2_{t,a}$ quantifies the mean and variance of the reward samples obtained from choosing that action.

TEUCB draws inspiration from the UCB1-Normal algorithm when calculating upper confidence bounds based on predictions done by tree ensemble methods. In order to make use of the UCB1-Normal framework, a couple of assumptions are made on the tree ensembles used for TEUCB. First, it is assumed that the output value o_n of the n'th tree in the ensemble, given a context $\mathbf{x}$, is an arithmetic average of c_n independent and identically distributed random variables with finite mean μ_n and σ_n^2. This implies that o_n is itself a random variable with mean μ_n and variance $\frac{\sigma_n^2}{c_n}$, which can be approximated by the sample mean and variance of previously observed data points that are assigned to the same leaf as $\mathbf{x}$.

By the central limit theorem (CLT) [33], the distribution of o_n then tends to a Gaussian distribution as $c_n \to \infty$. Hence, as the sum of Gaussian distributions is also Gaussian, the total tree ensemble prediction is a random variable with a Gaussian distribution, for which UCB1-Normal provides an effective exploration strategy.

With this in mind, rather than just predicting a point estimate—which is often the case when using tree ensembles for supervised learning tasks—the tree ensembles are used to also estimate the variance of its predictions, based on the samples it was trained on. Furthermore, the tree ensemble model also tracks the number of training data points previously observed in the various leaves in all its trees. This way, the method can reduce the uncertainty in its predictions as more and more data is collected, hence narrowing the confidence bounds and taking better actions more frequently.

2.2.7.7 Tree Ensemble Thompson Sampling

Tree Ensemble Thompson Sampling [31] is a contextual MAB algorithm utilizing tree ensembles for prediction and exploration inspired by Thompson Sampling [34]. Traditionally, Thompson Sampling has a distribution for each possible action. The agent samples one reward from each posterior distribution and greedily selects action as in Algorithm 4.

TETS is constructed such that the sampled reward is based on the variance, mean, and number of previously observed samples that have been assigned to the same leaves as an input context $\mathbf{x}$, similar to TEUCB. In practice, actions may be given a larger exploration bonus when sampling rewards due to one of two reasons: Either a large variation in their observed rewards, or due to being under-explored and having only a few data points. Both of these things contribute to higher uncertainty in the estimation of the expected reward associated with an action.

2.2.8 Combinatorial Multi-Armed Bandits

Combinatorial Multi-Armed Bandits (CMAB) [35] is an extension of the MAB framework, where the agent simultaneously pulls several arms and receives feedback from them in each time step. Pulling a set of arms $\mathcal{S}_t$ in time step t is referred to as selecting a super-arm, where one super-arm is equivalent to pulling several base-arms. The set of available super-arms may be restricted due to combinatorial constraints. As in the standard MAB problem, the agent only receives feedback

Algorithm 14 Tree Ensemble Upper Confidence Bound

```
1: Input: Number of rounds $T$, number of initial random selection rounds $T_I$, number of trees in ensemble $N$, exploration factor $\nu$, tree ensemble regressor $F(\cdot\,;\{\cdot,\cdot\})$.
2: for $t = 1$ to $T_I$ do
3:     Randomly select and play an arm $a_t$
4:     Observe context $\mathbf{x}_{t,a_t}$ and reward $r_{t,a_t}$
5: end for
6: for $t = T_I + 1$ to $T$ do
7:     Observe contexts $\{\mathbf{x}_{t,a}\}_{a=1}^{K}$
8:     for $a = 1$ to $K$ do
9:         Initialize arm parameters:
           $\tilde{\mu}_{t,a} \leftarrow 0, \quad \tilde{\sigma}^2_{t,a} \leftarrow 0, \quad c_{t,a} \leftarrow 0$
10:        for n = 1 to $N$ do
11:            Assign leaf values:
12:            $(o_{t,a,n}, s_{t,a,n}, c_{t,a,n}) \leftarrow F_n\left(\mathbf{x}_{t,a}; \{(\mathbf{x}_{i,a_i}, r_{i,a_i})\}_{i=1}^{t-1}\right)$
13:            Estimate expected partial reward distribution:
14:            $\tilde{\mu}_{t,a,n} \leftarrow o_{t,a,n}, \quad \tilde{\sigma}^2_{t,a,n} \leftarrow \frac{s^2_{t,a,n}}{c_{t,a,n}}$
15:            Increment arm parameters:
16:            $\tilde{\mu}_{t,a} \leftarrow \tilde{\mu}_{t,a} + \tilde{\mu}_{t,a,n}$
17:            $\tilde{\sigma}^2_{t,a} \leftarrow \tilde{\sigma}^2_{t,a} + \tilde{\sigma}^2_{t,a,n}$
18:            $c_{t,a} \leftarrow c_{t,a} + c_{t,a,n}$
19:        end for
20:        Calculate UCB:
21:        $U_{t,a} \leftarrow \tilde{\mu}_{t,a} + \sqrt{\nu^2 \tilde{\sigma}^2_{t,a} \frac{\ln(t-1)}{c_{t,a}}}$
22:    end for
23:    $a_t \leftarrow \text{argmax}_a U_{t,a}$
24:    Play $a_t$ and observe reward $r_{t,a_t}$
25: end for
```

from the pulled arms. The goal for the agent is to maximize the cumulative sum of all the rewards, or as usual when it comes to bandits, to minimize the regret.

2.2.8.1 Semi-bandit feedback

Semi-bandit feedback allows the agent to decompose the reward received from a super-armed into its constituents—the base-arms. This lets the agent learn about the individual base-arms, even though action selection is done through super-arms. This allows for data gathering of the specific arms which otherwise would be exhaustive or infeasible. For the combinatorial semi-bandit setting, the cumulative reward that is to be minimized is defined as

$$R(T) \triangleq \sum_{t=1}^{T} \left(\sum_{i \in \mathcal{S}_t^*} \mu_i - \sum_{j \in \mathcal{S}_t} \mu_j \right), \tag{2.28}$$

where $\mathcal{S}_t^*$ denotes the optimal super-arm at time t.

Algorithm 15 Tree Ensemble Thompson Sampling

```
1: Input: Number of rounds $T$, number of initial random selection rounds $T_I$, number of trees in ensemble $N$, exploration factor $\nu$, tree ensemble regressor $F(\cdot\,;\{\cdot,\cdot\})$.
2: for $t = 1$ to $T_I$ do
3:     Randomly select and play an arm $a_t$
4:     Observe context $\mathbf{x}_{t,a_t}$ and reward $r_{t,a_t}$
5: end for
6: for $t = T_I + 1$ to $T$ do
7:     Observe contexts $\{\mathbf{x}_{t,a}\}_{a=1}^{K}$
8:     for $a = 1$ to $K$ do
9:         Initialize arm parameters:
           $\tilde{\mu}_{t,a} \leftarrow 0, \quad \tilde{\sigma}^2_{t,a} \leftarrow 0$
10:        for n = 1 to $N$ do
11:            Assign leaf values:
12:            $(o_{t,a,n}, s_{t,a,n}, c_{t,a,n}) \leftarrow F_n\left(\mathbf{x}_{t,a}; \{(\mathbf{x}_{i,a_i}, r_{i,a_i})\}_{i=1}^{t-1}\right)$
13:            Estimate expected partial reward distribution:
14:            $\tilde{\mu}_{t,a,n} \leftarrow o_{t,a,n}, \quad \tilde{\sigma}^2_{t,a,n} \leftarrow \frac{s^2_{t,a,n}}{c_{t,a,n}}$
15:            Increment arm parameters:
16:            $\tilde{\mu}_{t,a} \leftarrow \tilde{\mu}_{t,a} + \tilde{\mu}_{t,a,n}$
17:            $\tilde{\sigma}^2_{t,a} \leftarrow \tilde{\sigma}^2_{t,a} + \tilde{\sigma}^2_{t,a,n}$
18:        end for
19:        Sample reward $\tilde{r}_{t,a} \sim \mathcal{N}(\tilde{\mu}_{t,a}, \nu^2 \tilde{\sigma}^2_{t,a})$
20:    end for
21:    $a_t \leftarrow \text{argmax}_a \tilde{r}_{t,a}$
22:    Play $a_t$ and observe reward $r_{t,a_t}$
23: end for
```

2.3 Related work

Multi-armed bandit algorithms such as ϵ-greedy, Thompson sampling and Bayesian Upper Confidence Bound have been used to solve the shortest path problem where the objective is to minimize energy consumption [1]. There, TS and Bayes-UCB both showed to accumulate significantly less regret than ϵ-greedy. This result agrees with that of [36], where Thompson sampling beat the performance of ϵ-greedy methods on the shortest path problem in various graphs. This second work did not consider any version of UCB for the shortest path problem, and in the first work mentioned, there was no clear winner between TS and UCB.

In a work closely related to [1], contextual bandits which utilize Gaussian processes to find correlations between side information and arm rewards, so-called Gaussian process bandits, are used in order to minimize energy consumption during navigation [2]. There, both UCB-based and TS methods are examined, and TS is shown to outperform UCB on the task.

We acknowledge the similarity of the problems considered in the above-mentioned works to our setup. However, we are unaware of any previous works that tackle the challenge of minimizing travel time during repeated navigation in road graphs using multi-armed bandits. Furthermore, due to the expected difficulty of the problem, we consider state-of-the-art machine learning models for predicting rewards based on the edges' contextual information, which has not been done in the works considering energy-efficiency.

3 Methods

This chapter describes the experimental setups, different reward functions considered, and evaluation metrics. Initially, we describe how the online learning problem is structured for the experimental setup. Further, the dynamics of the reward function challenge the agents throughout the experiments. This allows for fair comparison where the agents' performances can be evaluated by examining how well they handle the contextual data for the different reward functions varying in complexity.

3.1 Experimental setup

This work sets out to sequentially solve the shortest path problem, with edge weights that are initially unknown, through repeated traveling. In order to do so, the problem is modeled as a combinatorial multi-armed bandit. The agents are presented with a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a start node, an end node, and a time horizon. The graph data contains the longitude, latitude, and altitude for each node representing an intersection. The edges in the graph represent the road segments connecting the intersections. There is also additional data for each edge containing speed limitations, the length of the edge, and if it contains any stops.

The agent's objective is to minimize the cumulative regret. The instantaneous regret at time step t is calculated as the difference between the expected sum of the optimal and the chosen path. Depending on which type of reward function is used, the optimal path may change from time step to time step, as the time of the day changes. Initially, the agent has no prior knowledge of the feedback function.

Traversing a path in the graph is equivalent to the combinatorial bandit problem described in Section 2.2.8. Simultaneously pulling all base-arms that make up a super-arm is equivalent to traversing the chosen edges. The agent then gets the feedback for each edge traversed as described in Section 2.2.8.1. The agent will try to predict the weights w_e $\forall e \in \mathcal{E}$ in order to make good path selection.

3.2 Reward functions

We consider three different reward functions of varying difficulty when evaluating multi-armed bandits for this navigation task. To make the problem as realistic as possible, the graph is a model of the actual road network of Luxemburg. Consequently, the graph $\mathcal{G}$ consists of 2247 nodes and 5651 edges. The ultimate test of the agents is a problem setting that closely resembles the actual traffic situation in the city. Additionally, to provide further insights into the workings of the different algorithms and compare them on problem instances that are more straightforward to analyze, we test the agents on two synthetically constructed problem settings as well; one in which the the reward function is linear in the contextual features and the other where it is piece-wise constant.

3.2.1 Linear reward function

The simplest reward function is based on the dot product of the edge data and a constant vector. This reward function allows for evaluating whether or not the agent can comprehend the importance of the data presented and find the following simple linear relationship:

$$\mu_e = \theta^T \mathbf{x}_{e,t}. \tag{3.1}$$

Here, $\mathbf{x}_{e,t}$ is the context vector for edge e at time step t and θ is a vector of constant coefficients, with the same number of components as $\mathbf{x}_{e,t}$. To add some stochasticity to the problem, the weight of an edge w_e, which will become the observed reward for that edge should it be selected by the agent, is sampled from a rectified normal distribution with mean μ_e at each time step. Adding this noise makes the problem instance more realistic.

3.2.2 Piece-wise constant reward function

The second reward function bases the feedback on the sum of the latitudes of the ends of the edges, and the weight of the edges will change depending on the time of the day. This allows for showing whether or not the agent learns the connection between weights and the time of day.

$$\mu_e = 10 \cdot \begin{cases} \frac{u_{\text{lat}}(e)+v_{\text{lat}}(e)}{2}, & \text{if time of day earlier than 12 noon} \\ 1 - \frac{u_{\text{lat}}(e)+v_{\text{lat}}(e)}{2}, & \text{otherwise.} \end{cases} \tag{3.2}$$

Here, $u_{\text{lat}}(e)$ and $v_{\text{lat}}(e)$ refer to the latitudinal coordinates of the start and end points of an edge e. As these coordinates are fixed and do not change over the time horizon T, it is worth noting that the expected rewards are obtained from a piece-wise constant function over the feature that encodes the time of the day. Similarly to the linear rewards, rectified Gaussian noise is added to the rewards r_t the agents observe.

3.2.3 Real-world reward function

The most challenging reward function, and the one of primary focus in this work, is based on data collected from the Simulation of Urban MObility (SUMO) software. The data consists of travel times recorded for 24 hours on the road segments in the Luxembourg SUMO Traffic (LuST) [37] simulator. Essentially, this exposes the agents to realistic situations where the traversal time across edges will vary depending on the time of the day. The edges are categorized as residential roads, arterial roads, or highways. An indication of the complexity of traffic flows is presented in Fig. 3.1. Although this graph does not show the travel time on the road segment which is the quantity of interest in this project, the expected travel time is highly influenced by the traffic demand presented.

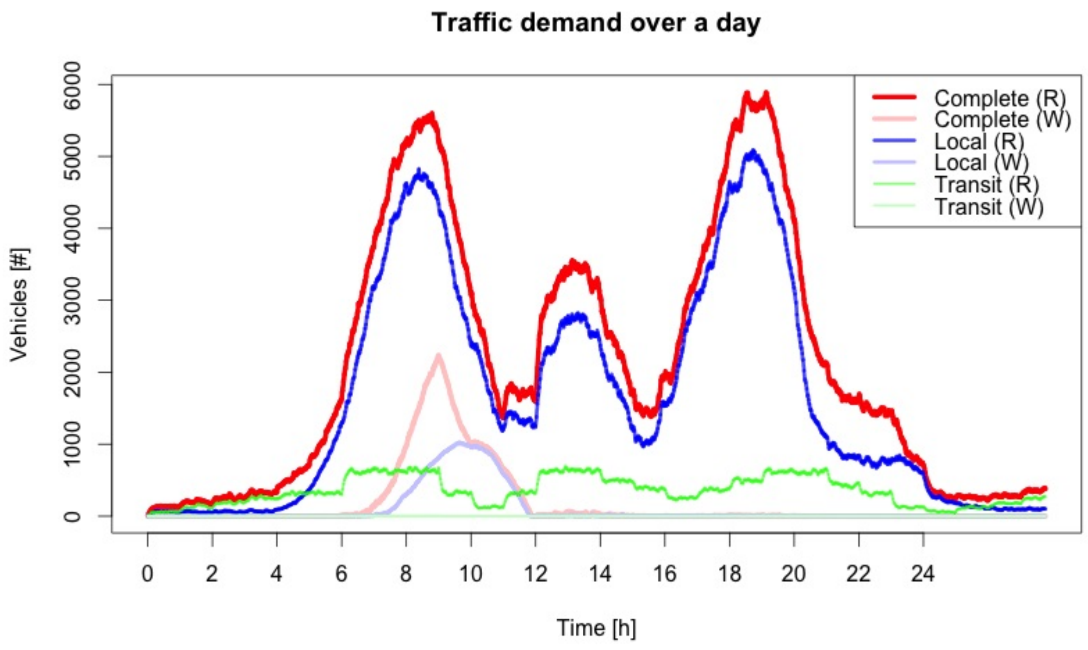


Figure 3.1: Traffic over the evaluated time horizon. R represents running vehicles and W represents waiting vehicles. Image from LuST: a 24-hour Scenario of Luxembourg City for SUMO Traffic simulations [38]

Since the congestion of traffic varies depending on the time of the day, as presented in Fig. 3.1, we need to use a probability distribution that mimics real-world behaviour. To solve this, we create distributions for each individual edge based on the sampled data using *kernel density estimation* (KDE) [39]. Kernel density estimation allows for drawing samples from a probability density function (PDF) obtained purely from observed data points $\mathbf{x}_i$. Formally, this kernel density estimated PDF is defined as:

$$\hat{f}_{\mathbf{H}}(\mathbf{x}) \triangleq \frac{1}{n}\sum_{i=1}^{n} K_{\mathbf{H}}\left(\mathbf{x}-\mathbf{x}_i\right), \tag{3.3}$$

where K is the *kernel* (a non-negative function), $\mathbf{H}$ is the *bandwidth* (a symmetric and positive definite matrix), and $K_{\mathbf{H}}$ is defined as:

$$K_{\mathbf{H}} \triangleq |\mathbf{H}|^{-1/2} K(\mathbf{H}^{-1/2}\mathbf{x}). \tag{3.4}$$

As of K, we select the multivariate Gaussian kernel, which means the kernel function will be the multivariate normal distribution. In such a case, the entries of the covariance matrix will be determined through the particular bandwidth matrix selection, together with the data. The constructed estimator depends critically on selecting an appropriate kernel and bandwidth for accurate approximation. Fortunately, there is a general guideline for computing the entries of the bandwidth matrix which tends to work well, known as *Silverman's rule of thumb* [40]:

$$\begin{cases} \sqrt{\mathbf{H}_{ij}} = \left(\frac{4}{d+2}\right)^{\frac{1}{d+4}} n^{\frac{-1}{d+4}} \sigma_i \qquad i = j, \\ \mathbf{H}_{ij} = 0 \qquad i \neq j, \end{cases} \tag{3.5}$$

where d is the number of dimensions and σ_i is the standard deviation of the i^{th} variable.

In practice, the agent receives the arm feedback, or edge weight w_e, interpreted as the time it took to drive along the edge, based on the KDE conditioned on the time of the day t_{day} sampled as

$$w_e \sim \hat{f}_{\mathbf{H}}(w_e|t_{\text{day}}).$$

In practice, we make use of the Nadaraya-Watson estimator to obtain such conditional samples [41] [42] [43]. The same technique is employed for calculating the conditional expectations needed to evaluate the regret associated with a selected path, which will be described in the following section.

The simulator that generated the travel times did not direct any cars to some of the road segments during the 24-hour simulation. Therefore, there are edges in the graph $\mathcal{G}$ containing no recorded traversals, and this has to be addressed. To solve this, the edge weight, or travel time, on these edges is set to the edge length divided by the speed limit.

3.3 Evaluation metrics

The agents' choices are compared to an oracle agent, who always chooses the optimal path. The oracle consistently receives the true expected value μ_e of all edges e and therefore finds the shortest path in every time step t. Utilizing Dijkstra's algorithm, the oracle thus accumulates zero regret. On the contrary, the agents are unaware of the underlying distributions, and only receive stochastic feedback as described in Section 3.2. To evaluate their performances, though, the true expected travel times μ_e of the edges traversed in the path p are considered. The total expected travel time is then compared to that of the oracles path p^* to calculate the regret in time step t as

$$R(t) = \underbrace{\sum_{\forall e \in p} \mathbb{E}(\mu_e)}_{\text{Agent's path}} - \underbrace{\sum_{\forall e \in p*} \mathbb{E}(\mu_e)}_{\text{Oracle's path}}. \tag{3.6}$$

At each time step t and for every edge e, the agent is presented with a context vector $\mathbf{x}_{e,t} \in \mathbb{R}^d$. In practice, the agent is presented with a randomly sampled time of day at each time step t and is evaluated compared to the optimal path conditioned on the sampled time.

4 Results

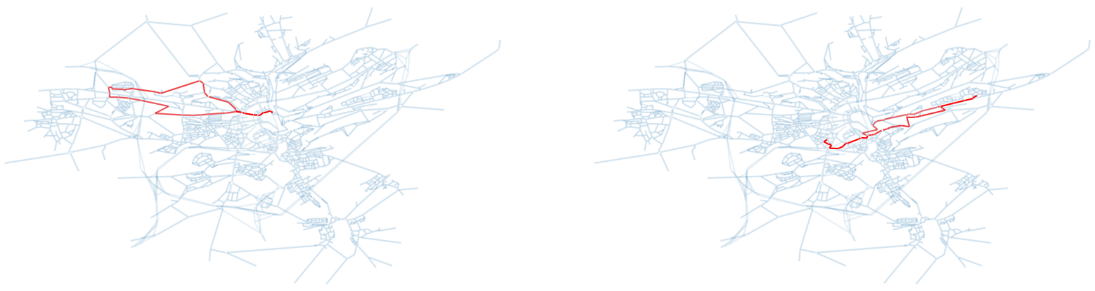
This chapter presents the results from experiments in the different environments explained in ??. The complexity of the synthetic reward functions varies from a linear function to a simple but non-linear function that is piece-wise constant with respect to the features. Finally, the most complex reward function is based on real-world data.

For each of the three environments, the agents are evaluated on two pairs of start- and end-nodes in the road network. In the sections presenting the results and the path, these paths will be referred to as problem instance 1 and problem instance 2. Depending on the complexity of the reward function the number of optimal paths will vary. The oracle's path selection will present the optimal paths in figures for the corresponding environments.

Finally, all agents estimate the edge weights $\mathbf{w_e}$ for all edges $e \in \mathcal{E}$. The shortest path is then selected using Dijkstra's algorithm. The regret is calculated by Eq. (3.6) based on the path's edge weights as described in ??.

4.1 Linear reward function

In the linear environment, the edge weights are based on the linear relationship defined in Section 3.2.1. This environment yields the optimal paths presented in Fig. 4.1. Furthermore, it is worth noting that there are two optimal paths in both problem settings.



(a) The paths chosen by the oracle with the linear reward function in path 1.

(b) The paths chosen by the oracle with the linear reward function in path 2.

Figure 4.1: Shows the optimal paths by the oracle in the linear environment.

The cumulative regrets for the evaluated agents are presented in Fig. 4.2 and Fig. 4.3, as well as the corresponding tables 4.1 and 4.2. There is generally a large gap between the contextual and non-contextual agents on this simple contextual reward function, where the contextual ones perform better, but with a couple of exceptions. One observation is that TreeBootstrap performs worst, with significantly slower learning and more regret accumulated. It appears that having a single decision tree or tree ensemble per arm is in this case only a disadvantage compared with collecting statistics on each arm without using the contexts.

Another observation is that LinTS accumulates much regret early, and tends to perform worst out of all agents initially. However, this agent seems to learn to choose good routes and displays a relatively flat regret curve eventually, beating both ϵ-greeedy and TS on both problem instances.

The third non-contextual agent, BayesUCB, on the other hand, accumulates a much lower regret on problem instance 1, shown in Fig. 4.2, this seems to be attributed to converging on a fairly good path(s) much faster than ϵ-greedy and TS. This can be seen by studying the shape of the curves, where the one corresponding to BayesUCB presents a similar slope later in the experiment, but reaches this slope faster. On the second problem instance, for which the regret plots are presented in Fig. 4.3, BayesUCB not only seems to learn quicker, but also to find better paths than the other two non-contextual agents, achieving a regret plot that is more similar to the contextual ones.

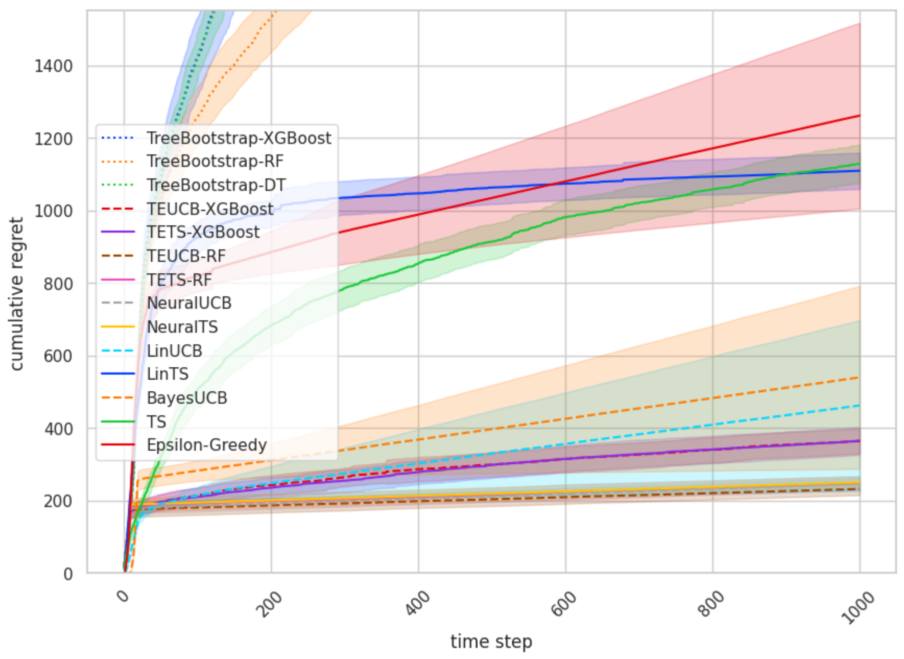


Figure 4.2: Cumulative regret for 14 different agents on the synthetic linear reward function, on problem instance 1.

Table 4.1: Mean and standard deviation of cumulative regret for each agent on problem instance 1 in the linear road graph.

Bandit	**Mean**	**Std**
BayesUCB	540	399
Epsilon-Greedy	1261	405
LinTS	1109	79
LinUCB	462	373
NeuralTS	250	28
NeuralUCB	247	25
TETS-RF	248	30
TETS-XGBoost	365	61
TEUCB-RF	**232**	26
TEUCB-XGBoost	365	54
TreeBootstrap-DT	4698	162
TreeBootstrap-RF	2976	214
TreeBootstrap-XGBoost	4848	250
TS	1129	84

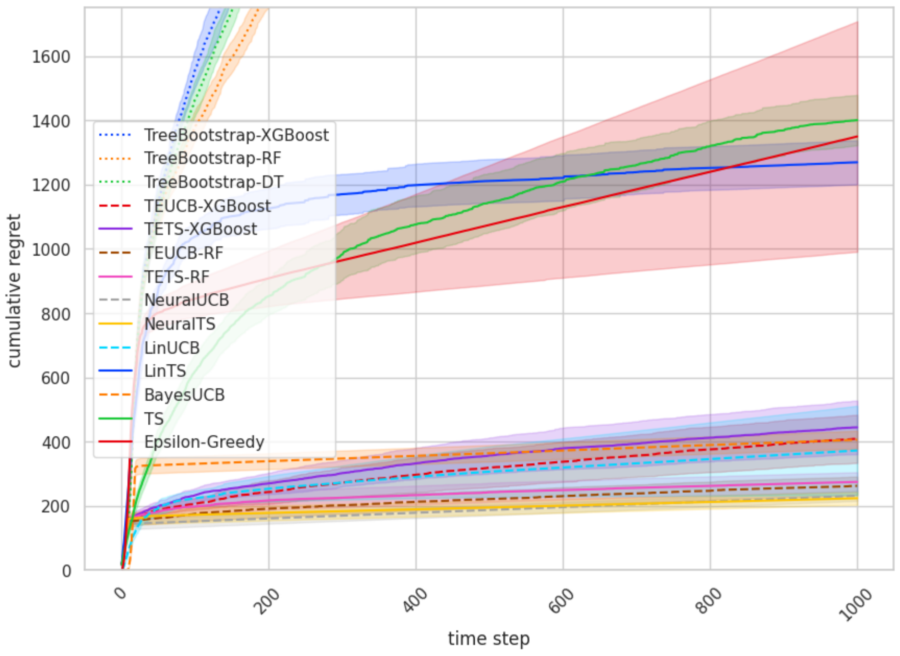


Figure 4.3: Cumulative regret for 14 different agents on the synthetic linear reward function, on problem instance 2.

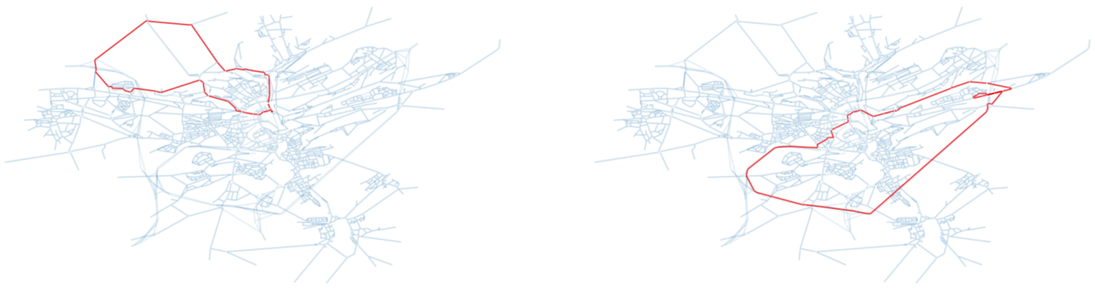
Table 4.2: Mean and standard deviation of cumulative regret for each agent on problem instance 2 in the linear road graph.

Bandit	Mean	Std
BayesUCB	406	44
Epsilon-Greedy	1350	567
LinTS	1269	108
LinUCB	373	220
NeuralTS	**224**	32
NeuralUCB	232	25
TETS-RF	275	48
TETS-XGBoost	445	132
TEUCB-RF	263	46
TEUCB-XGBoost	409	119
TreeBootstrap-DT	5676	285
TreeBootstrap-RF	3915	375
TreeBootstrap-XGBoost	5782	304
TS	1400	124

4.2 Piece-wise constant reward function

The piece-wise constant reward function bases the feedback on the longitudinal sum of the two endpoints of the edges. This gives rise to two different optimal paths in each problem instance as presented in Fig. 4.4. Similar to the linear environment, the results of the experiments in this environment indicate that non-contextual agents encounter difficulties finding the relationship between contextual features and the reward function, which in this case contains a discrete split based on the time of day. Regarding the contextual MAB algorithms, the linear and neural network-based agents, which assume linear and smooth reward functions respectively, struggle to find good paths in this environment.

Tree-based models seem to perform the best in this environment, and these agents eventually present a significantly flatter regret curve than the other methods in general. Further, comparing TEUCB and TETS with TreeBootstrap, the two former methods appear to learn much quicker than the latter.



(a) Problem instance 1. **(b)** Problem instance 2.

Figure 4.4: Shows the optimal paths by the oracle in the piece-wise constant environment for both problem instances.

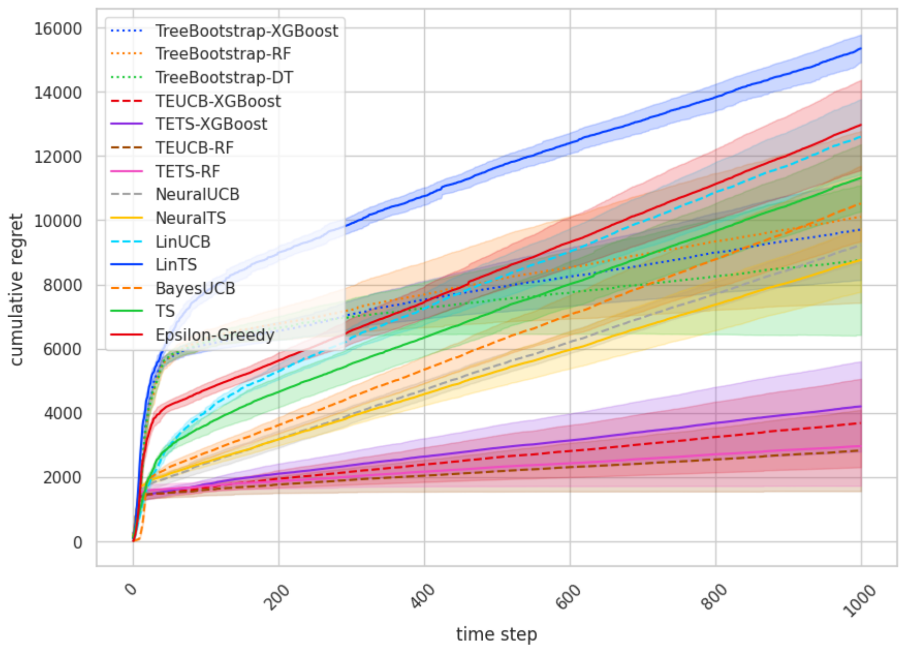


Figure 4.5: Cumulative regret for 14 different agents on the synthetic piece-wise constant reward function, on problem instance 1.

Table 4.3: Mean and standard deviation of cumulative regret for each agent on problem instance 1 in the piece-wise constant road graph.

Bandit	**Mean**	**Std**
BayesUCB	10523	1861
Epsilon-Greedy	12967	2222
LinTS	15345	683
LinUCB	12597	1850
NeuralTS	8762	1567
NeuralUCB	9226	845
TETS-RF	2963	1951
TETS-XGBoost	4200	2218
TEUCB-RF	**2828**	2004
TEUCB-XGBoost	3683	2178
TreeBootstrap-DT	8755	3683
TreeBootstrap-RF	10104	4231
TreeBootstrap-XGBoost	9704	2478
TS	11318	1651

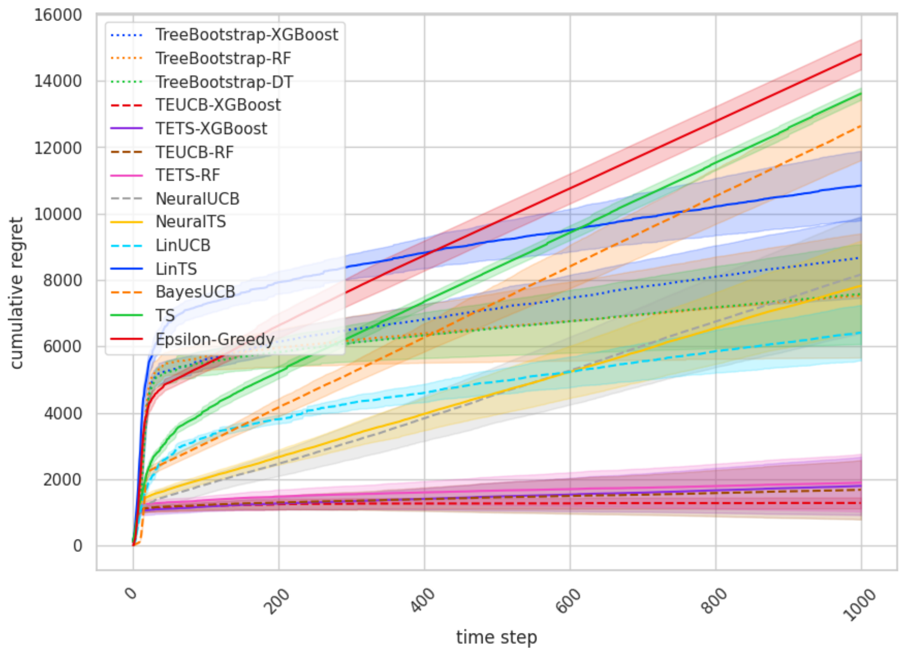


Figure 4.6: Cumulative regret for 14 different agents on the synthetic piece-wise constant reward function, on problem instance 2.

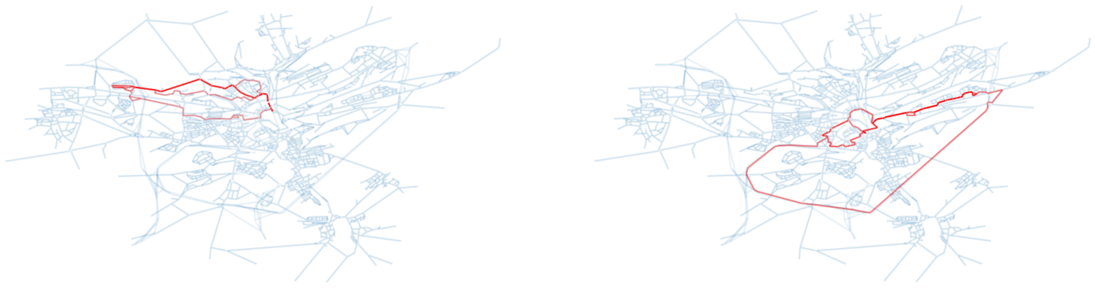
Table 4.4: Mean and standard deviation of cumulative regret for each agent on problem instance 2 in the piece-wise constant road graph.

Bandit	Mean	Std
BayesUCB	12637	1627
Epsilon-Greedy	14794	720
LinTS	10841	1656
LinUCB	6412	1319
NeuralTS	7822	2151
NeuralUCB	8162	2794
TETS-RF	1895	1371
TETS-XGBoost	1798	1386
TEUCB-RF	1683	1414
TEUCB-XGBoost	**1284**	265
TreeBootstrap-DT	7574	2375
TreeBootstrap-RF	7533	2956
TreeBootstrap-XGBoost	8673	1920
TS	13609	289

4.3 Real-world experiments

The environment considered here emulates real-world traffic flow where the traversal time is highly conditioned on the time of day. Certain start and end points in this environment provide a challenging environment where four or more paths can be optimal depending on the time of the day, which can be seen in Fig. 4.7. As Fig. 3.1 indicates, this reward function is rather complex. This is also reflected by the non-contextual and linear agents' inabilities to learn to make good path selections, which is clear from looking at Fig. 4.8, Fig. 4.9, Table 4.5 and Table 4.6.

We also note that, as for the piece-wise constant reward function, tree-based methods tend to learn better paths than the neural agents on this task, although TreeBootsstrap learns slow and gathers more regret initially. Finally, the TEUCB and TETS implementations using XGBoost outperform those using random forests.



(a) The paths chosen by the oracle in the real world experiment for problem instance 1.

(b) The paths chosen by the oracle in the real world experiment for problem instance 2.

Figure 4.7: Shows the optimal paths by the oracle in the real-world experiment environment.

4.3.1 Runtime analysis

Although the main focus of this work—and multi-armed bandits in general—is on regret minimization, it is also relevant to ask how resource-intensive the various algorithms are. It may be infeasible to implement complex models on smaller computing platforms. Therefore, we will compare the run time for the different algorithms for the real-world navigation problem. All agents were run on a single desktop CPU, apart from NeuralUCB and NeuralTS, which were run on an NVIDIA A40 GPU. The reason is the inefficiency of training neural networks on a small number of cores and the large benefit one obtains in terms of runtime through parallelization. It should be noted that this significantly increases the cost of achieving the level of performance we present with the neural agents compared with all others.

All reported runtimes are summarized in Table 4.7. The experiments took about an hour on the GPU for a neural agent. In contrast, TETS and TEUCB run at comparable times on a single CPU, as the experiments with TEUCB and TETS took about 1.5 hours with XGBoost, and 5 hours using random forests. For the linear agents,

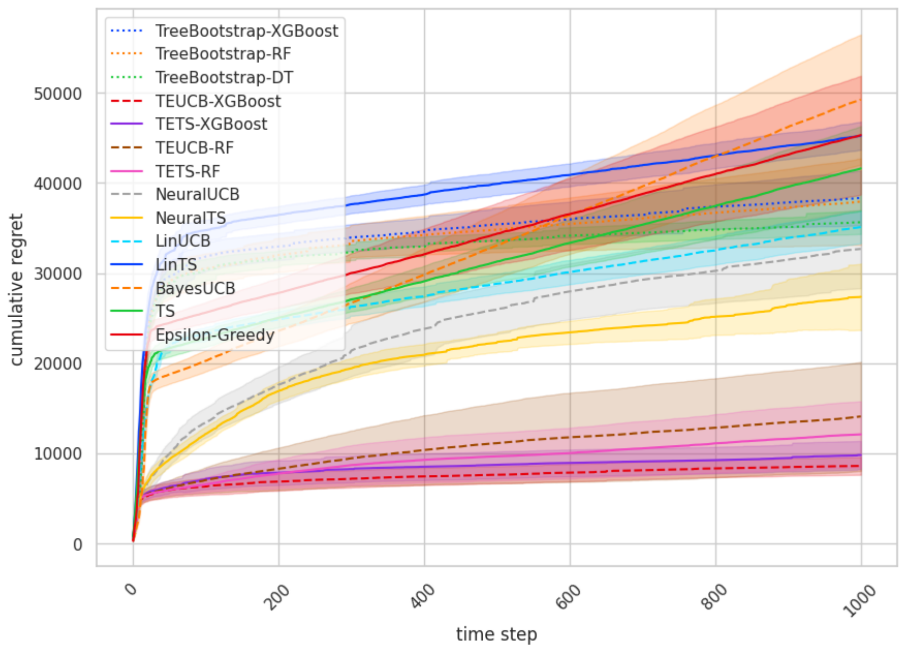


Figure 4.8: Cumulative regret for 14 different agents on the real-world reward function, on problem instance 1.

Table 4.5: Mean and standard deviation of cumulative regret for each agent on problem instance 1 in real-world road graph.

Bandit	**Mean**	**Std**
BayesUCB	49264	11394
Epsilon-Greedy	45322	10409
LinTS	45250	2469
LinUCB	35132	2848
NeuralTS	27373	5860
NeuralUCB	32685	6902
TETS-RF	12102	5802
TETS-XGBoost	9810	2468
TEUCB-RF	14097	9575
TEUCB-XGBoost	**8601**	1556
TreeBootstrap-DT	35641	2061
TreeBootstrap-RF	37932	7586
TreeBootstrap-XGBoost	38344	4800
TS	41625	7413

LinUCB and LinTS, as well as TreeBootstrap with a single decision tree per arm, the experiments took approximately 30 minutes. TreeBootstrap with random forest took roughly 20 hours, while TreeBootstrap with XGBoost ran the experiments in 4 hours. The non-contextual methods, BayesUCB, standard Thompson Sampling, and Epsilon-Greedy all completed the experiment in about 10 minutes.

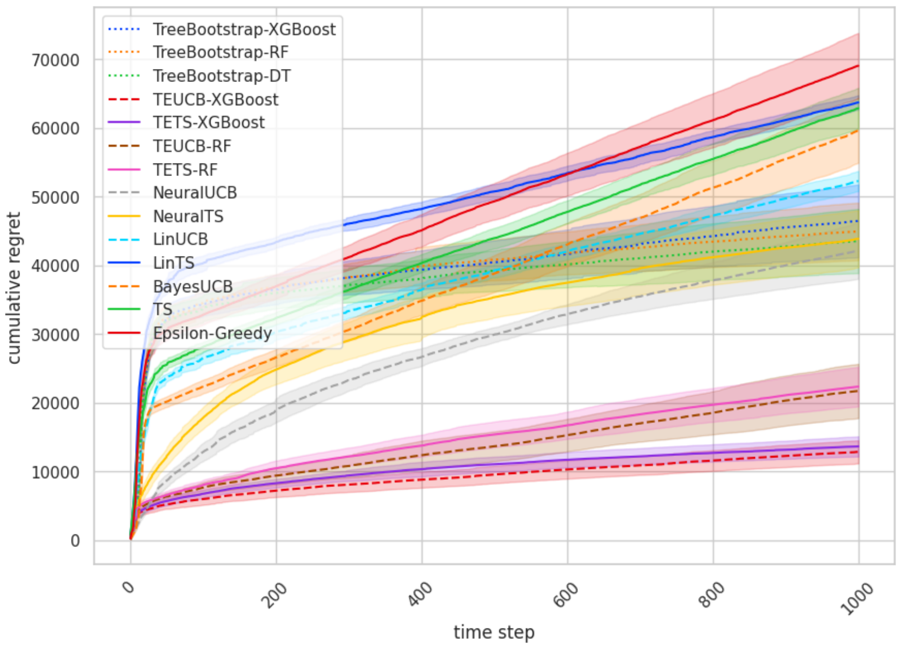


Figure 4.9: Cumulative regret for 14 different agents on the real-world reward function, on problem instance 2.

Table 4.6: Mean and standard deviation of cumulative regret for each agent on problem instance 2 in real-world road graph.

Bandit	**Mean**	**Std**
BayesUCB	59597	7448
Epsilon-Greedy	68995	7531
LinTS	63660	1659
LinUCB	52238	2321
NeuralTS	43846	6554
NeuralUCB	42152	6526
TETS-RF	22351	4549
TETS-XGBoost	13662	2324
TEUCB-RF	21722	6249
TEUCB-XGBoost	**12863**	2655
TreeBootstrap-DT	43519	7388
TreeBootstrap-RF	44916	6620
TreeBootstrap-XGBoost	46439	8377
TS	62820	4719

Table 4.7: Approximate runtime for each of the 14 different agents on the real-world reward problem setting.

Bandit	**Runtime (hours)**
BayesUCB	0.2
Epsilon-Greedy	0.2
LinTS	0.5
LinUCB	0.5
NeuralTS (GPU)	0.9
NeuralUCB (GPU)	0.9
TETS-RF	4.8
TETS-XGBoost	1.5
TEUCB-RF	4.8
TEUCB-XGBoost	1.5
TreeBootstrap-DT	0.5
TreeBootstrap-RF	19.7
TreeBootstrap-XGBoost	4.2
TS	0.2

5 Discussion

In this chapter, we will discuss the observations found by examining the results presented in Chapter 4. Further, we will share our interpretation of the results and experiences conducting these experiments.

5.1 Contextual vs. non-contextual bandits

The increased complexity of the environments uncovers the strengths and weaknesses of the different agents. First and foremost, the distinction between contextual and non-contextual agents shows that considering the contextual features available vastly increases the learning rate and is crucial for all environments. Non-contextual agents may eventually find near-optimal paths in basic environments but encounter difficulties as the environments get more complex. Moreover, the results show that making a linear assumption is beneficial as long as this underlying assumption holds for the reward function, but may cause huge problems where it does not. For the non-linear reward functions examined in this work, the linear bandits have trouble learning the environment and finally concentrate on paths less optimal than the ones that bandits with more complex models does.

5.2 Tree ensembles vs. neural networks

Initially, when we started this project we encountered difficulties with the neural agents as training relatively large neural networks in combination with searching for good hyperparameters turned out to be a challenge. Therefore, we gathered data from the path traversal and made the problem into a supervised learning problem. When we worked with the offline supervised learning problem we saw that the neural network had a significantly larger MSE compared to XGBoost when estimating the edge weights. Therefore we decided to implement TETS and TEUCB. Later, we got access to GPU capabilities which made it feasible to increase the sizes of the neural networks and perform exhaustive grid search over wide ranges of hyperparameters. This allowed us to find neural network architectures and training specifications which significantly increased the performance of NeuralUCB and NeuralTS. Still, the neural agents could not match the low level of regret achieved by TEUCB and TETS.

We believe that one possible explanation for this is the way in which neural networks utilize weights and biases to fit a smooth function to the data. On the contrary, decision trees discretely partition the data based on some splitting criteria. For this problem, two adjacent or closely located road segments may have similar contextual features, although one road segment is very rapid while the other may be very slow. As such edges would give rise to vastly different travel times, the decision tree-based models may isolate these segments as the MSE in the nodes would be too large. In comparison, the neural network-based models may adjust the weights and biases to set the predicted function value somewhere between the two sets of observed travel times, as long as the edge features are similar enough. Hence, distinguishing between edges that are similar in context would be challenging, and could be devastating for regret minimization when travel times on them vary significantly.

Further, decision trees are not as dependent on pre-processing procedures such as re-scaling or imputation compared to neural networks [44]. From the experiments we have performed, our experience is that tree ensembles are less sensitive to hyper-parameter tuning.

5.3 Upper confidence bounds vs. Thompson sampling

In contrast to the observation from [2] for the similar problem of minimizing energy consumption while navigating in road networks, we do not see an advantage of TS over UCB methods. If anything, UCB tends to perform slightly better than the corresponding agents using TS as their exploration method in our experiments. However, this difference is not significant, and we cannot claim definite superiority of one method over the other.

5.4 One vs. multiple models

There are different strategies for whether or not to assign one model for each arm. The neural models, TEUCB and TETS utilize one model that predicts the expected weights for all edges in the graph. This stands in contrast to the linear bandits and TreeBootstrap, which have one model for each specific edge. Having one model per arm can be beneficial for problems where the number of actions is limited and static, and where their reward distributions are uncorrelated. On the contrary, for problems where the rewards of one arm help predicting the rewards of another through shared contextual features, distinct models may lead to excessive exploration. The tendency of the agents that employ a separate model for each arm to explore many more road segments can be seen in the figures displaying the agents' path selections, presented in Appendix A.

5.5 Run time

From the run times of the algorithms in the real-world navigation task, reported in Section 4.3.1, it is evident that the choice of the agent has a huge impact on the computational requirements. When it comes to the neural agents, a GPU is needed to obtain decent results in a reasonable amount of time, which makes these methods costly and may limit the settings in which they are feasible.

Looking at the models that employ a separate model for every arm, we note that the choice of base model makes a significant difference. Although this proved not to be the case for this problem, having unique models for each arm may be beneficial for other bandit problems with less correlated arms. In such cases, it can be important to find a good balance between speed and accuracy, and select algorithms that achieve low regret while also operating within a reasonable amount of time. Using a linear model or a single decision tree the algorithms solve this problem relatively fast, but when employing an ensemble of 100 trees for every arm, they tend to run much slower.

The choice of the particular type of model used for the tree-based methods were shown to affect the inherent strengths and weaknesses of the corresponding bandits. As presented in Chapter 4, the performance of TreeBootstrap, TEUCB and TETS varied depending on whether they were implemented using decision trees, random forests, or XGBoost ensembles. The selection of tree model showed to have a particularly large effect on the runtimes. In practise, one may therefore have to decide whether a slight decrease in regret at the cost of a significant increase in time consumption is worth it or not.

Further, random forest was shown to be much slower than XGBoost in our experiments. It should be noted, however, that due to the way in which random forest works, with trees that are built independently from each other, it may be possible to speed up the process through parallelization on a GPU. As mentioned above, though, GPUs tend to be expensive which can be a constraint.

5.6 Future work

As this work takes much inspiration from the works of [1] and [2], which look at efficient navigation from the standpoint of minimizing energy consumption, it would be interesting to try and merge the three. For a navigation system to be useful in real vehicles, both energy efficiency and travel time minimization are likely to be of immense importance. Therefore, it may be worth looking into how one might combine them. Also, since different multi-armed bandits have been shown to solve the two problems effectively, it would be interesting to see which bandit algorithms are best suited for such combined task.

Another thing that one might want to examine is what happens if the travel times on road segments are subject to data drift. In this work, the stochastic aspects considered were the variability of the expected travel times depending on the time of the day, as well as the deviations from these expected travel times in the realized travel times. In reality, however, it is reasonable to believe that the expected travel times may also vary depending on the season or other external factors. For instance, certain edges may experience shifts in travel times due to construction work at or near those road segments. Therefore, it would be interesting to investigate how well the methods implemented in this work handle these types of data drift, and potentially what additional mechanisms are needed for successful regret minimization. One such mechanism could be considering replay buffers [45] as rolling windows instead of the full set of previous observations.

As discussed in Section 5.2, the choice of base model appears to be substantial when it comes to how successful the agent is at the task. Tree ensemble methods turned out to be most successful at navigating in the real-world environment, which was the main focus of this thesis. Most likely this correlates with higher accuracy, or more specifically lower error when estimating the edge weights, as was seen when evaluating XGBoost and neural networks on offline data. In light of this, if the goal is to evaluate the effectiveness of the exploration methods alone, one could consider different underlying estimator models with similar accuracy, and then compare the cumulative regret of those agents. This could lead to a more fair comparison between the neural agents' gradient-based exploration with the exploration methods of TEUCB and TETS based on leaf assignments.

6 Conclusion

In this thesis, we have represented a road network as a graph, in order to develop effective navigation methods. Additionally, a combinatorial and contextual multi-armed bandit framework was incorporated, providing sophisticated exploration techniques and efficient learning of the edge weight distributions corresponding to travel time.

Further, we demonstrated that the tree ensemble methods TEUCB and TETS, which we implemented using both XGBoost and random forest as the tree ensemble methods, generalize well, learn effectively, and are resource-efficient. These things combined make TEUCB and TETS the best algorithms for time-efficient navigation over the stochastic road networks considered in this thesis. In particular, XGBoost implementations demonstrated a particularly high level of performance.

Finally, we hope that the results of this thesis inspire further exploration of tree ensembles for contextual multi-armed bandits due to their simplicity and solid performance.

Bibliography

[1] N. Åkerblom, Y. Chen, and M. H. Chehreghani, "Online learning of energy consumption for navigation of electric vehicles," *Artificial Intelligence*, vol. 317, p. 103 879, Apr. 2023. DOI: `10.1016/j.artint.2023.103879`. [Online]. Available: `https://doi.org/10.1016%2Fj.artint.2023.103879` (return to pages 1, 26, 47).

[2] J. Sandberg, N. Åkerblom, and M. H. Chehreghani, *Combinatorial gaussian process bandits in bayesian settings: Theory and application for energy-efficient navigation*, 2023. arXiv: `2312.12676 [cs.LG]` (return to pages 1, 26, 46, 47).

[3] E. Dijkstra, "A note on two problems in connexion with graphs.," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959 (return to page 5).

[4] *On a routing problem*, 1958. DOI: `10.1090/QAM/102435` (return to page 5).

[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second. The MIT Press, 2018. [Online]. Available: `http://incompleteideas.net/book/the-book-2nd.html` (return to page 6).

[6] A. Slivkins, *Introduction to multi-armed bandits*, 2024. arXiv: `1904.07272 [cs.LG]` (return to pages 6–10).

[7] D. Bouneffouf and I. Rish, *A survey on practical applications of multi-armed and contextual bandits*, 2019. arXiv: `1904.10040 [cs.LG]` (return to page 6).

[8] A. Slivkins, *Introduction to multi-armed bandits*, 2022. arXiv: `1904.07272 [cs.LG]` (return to page 6).

[9] N. C.-B. Peter Auer and P. Fischer, "Finite-time analysis of the multiarmed bandit problem*," 2002. DOI: `10.1023/A:1013689704352`. [Online]. Available: `https://doi.org/10.1023/A:1013689704352` (return to page 8).

[10] E. Kaufmann, O. Cappe, and A. Garivier, "On bayesian upper confidence bounds for bandit problems," in *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, N. D. Lawrence and M. Girolami, Eds., ser. Proceedings of Machine Learning Research, vol. 22, La Palma, Canary Islands: PMLR, 21–23 Apr 2012, pp. 592–600. [Online]. Available: `https://proceedings.mlr.press/v22/kaufmann12.html` (return to page 11).

[11] N. Åkerblom, Y. Chen, and M. Haghir Chehreghani, "Online learning of energy consumption for navigation of electric vehicles," *Artificial Intelligence*, vol. 317, p. 103 879, 2023. DOI: `10.1016/j.artint.2023.103879` (return to page 11).

[12] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proceedings of the*

19th international conference on World wide web, ACM, Apr. 2010. DOI: 10.1145/1772690.1772758. [Online]. Available: https://doi.org/10.1145%2F1772690.1772758 (return to pages 12, 18).

[13] B. Mehlig, *Machine Learning with Neural Networks: An Introduction for Scientists and Engineers*. Cambridge University Press, Oct. 2021, ISBN: 9781108494939. DOI: 10.1017/9781108860604. [Online]. Available: http://dx.doi.org/10.1017/9781108860604 (return to page 12).

[14] S. Haykin, *Neural Networks: A Comprehensive Foundation* (International edition). Prentice Hall, 1999, ISBN: 9780132733502. [Online]. Available: https://books.google.no/books?id=bX4pAQAAMAAJ (return to page 13).

[15] S. K and S. S, "Review on classification based on artificial neural networks," *The International Journal of Ambient Systems and Applications*, vol. 2, pp. 11–18, Dec. 2014. DOI: 10.5121/ijasa.2014.2402 (return to page 13).

[16] D. Zhou, L. Li, and Q. Gu, "Neural contextual bandits with ucb-based exploration," in *Proceedings of the 37th International Conference on Machine Learning*, ser. ICML'20, JMLR.org, 2020 (return to pages 13, 21).

[17] J. N. Morgan and J. A. Sonquist, "Problems in the analysis of survey data, and a proposal," *Journal of the American Statistical Association*, vol. 58, pp. 415–434, 1963. [Online]. Available: https://api.semanticscholar.org/CorpusID:1825515 (return to page 13).

[18] M. Bramer, "Avoiding overfitting of decision trees," in Nov. 2016, ISBN: 978-1-4471-7306-9. DOI: 10.1007/978-1-4471-7307-6_9 (return to page 14).

[19] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer, 2009 (return to pages 14, 15).

[20] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996 (return to page 15).

[21] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001. DOI: 10.1023/A:1010950718922 (return to page 16).

[22] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16, ACM, 2016. DOI: 10.1145/2939672.2939785 (return to pages 16, 17).

[23] C. Bentéjac, A. Csörgő, and G. Martínez-Muñoz, "A comparative analysis of gradient boosting algorithms," *Artificial Intelligence Review*, vol. 54, no. 3, pp. 1937–1967, Aug. 2020, ISSN: 1573-7462. DOI: 10.1007/s10462-020-09896-5. [Online]. Available: http://dx.doi.org/10.1007/s10462-020-09896-5 (return to page 16).

[24] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970, ISSN: 00401706. [Online]. Available: http://www.jstor.org/stable/1267351 (visited on 11/01/2023) (return to page 18).

[25] T. J. Walsh, I. Szita, C. Diuk, and M. L. Littman, "Exploring compact reinforcement-learning representations with linear regression," in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, ser. UAI '09, Montreal, Quebec, Canada: AUAI Press, 2009, pp. 591–598, ISBN: 9780974903958 (return to page 18).

[26] W. Chu, L. Li, L. Reyzin, and R. Schapire, “Contextual bandits with linear payoff functions,” in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, G. Gordon, D. Dunson, and M. Dudík, Eds., ser. Proceedings of Machine Learning Research, vol. 15, Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 208–214. [Online]. Available: `https://proceedings.mlr.press/v15/chu11a.html` (return to page 18).

[27] S. Agrawal and N. Goyal, “Thompson sampling for contextual bandits with linear payoffs,” in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ser. ICML’13, Atlanta, GA, USA: JMLR.org, 2013, III–1220–III–1228 (return to page 19).

[28] A. N. Elmachtoub, R. McNellis, S. Oh, and M. Petrik, “A practical method for solving contextual bandit problems using decision trees,” *arXiv preprint arXiv:1706.04687*, 2017 (return to pages 19, 20).

[29] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML’10, Haifa, Israel: Omnipress, 2010, pp. 807–814, ISBN: 9781605589077 (return to page 21).

[30] W. Zhang, D. Zhou, L. Li, and Q. Gu, *Neural thompson sampling*, 2021. arXiv: `2010.00827 [cs.LG]` (return to page 22).

[31] H. Nilsson, R. Johansson, N. Åkerblom, and M. H. Chehreghani, *Tree ensembles for contextual bandits*, 2024. arXiv: `2402.06963 [cs.LG]` (return to pages 23, 24).

[32] P. Auer, N. Cesa-Bianchi, and P. Fischer, “Finite-time analysis of the multiarmed bandit problem,” *Machine Learning*, vol. 47, no. 2, pp. 235–256, 2002, ISSN: 1573-0565. DOI: `10.1023/A:1013689704352` (return to page 23).

[33] Y. Dodge, “The concise encyclopedia of statistics,” in New York, NY: Springer New York, 2008, pp. 66–68, ISBN: 978-0-387-32833-1. DOI: `10.1007/978-0-387-32833-1_50` (return to page 24).

[34] W. R. Thompson, “On the likelihood that one unknown probability exceeds another in view of the evidence of two samples,” *Biometrika*, vol. 25, no. 3/4, pp. 285–294, 1933, ISSN: 00063444 (return to page 24).

[35] N. Cesa-Bianchi and G. Lugosi, “Combinatorial bandits,” *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1404–1422, 2012, JCSS Special Issue: Cloud Computing 2011, ISSN: 0022-0000. DOI: `10.1016/j.jcss.2012.01.001` (return to page 24).

[36] D. J. Russo, B. Van Roy, A. Kazerouni, I. Osband, Z. Wen, *et al.*, “A tutorial on thompson sampling,” *Foundations and Trends® in Machine Learning*, vol. 11, no. 1, pp. 1–96, 2018 (return to page 26).

[37] L. Codeca, R. Frank, S. Faye, and T. Engel, “Luxembourg sumo traffic (lust) scenario: Traffic demand evaluation,” *IEEE Intelligent Transportation Systems Magazine*, vol. 9, no. 2, pp. 52–63, 2017. DOI: `10.1109/MITS.2017.2666585` (return to page 30).

[38] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, *LuST: a 24-hour Scenario of Luxembourg City for SUMO Traffic simulations*, ResearchGate, Available from: `https://www.researchgate.net/figure/Traffic-Demand-`

over-a-day-R-represents-the-running-vehicles-and-W-the-waiting-ones_fig4_276062925 [accessed 2 May, 2024], 2018 (return to page 31).

[39] S. Weglarczyk, "Kernel density estimation and its application," *ITM Web of Conferences*, vol. 23, p. 00 037, 2018. DOI: 10.1051/itmconf/20182300037 (return to page 31).

[40] B. W. Silverman, *Density Estimation for Statistics and Data Analysis.* London: Chapman & Hall, 1986 (return to page 32).

[41] E. A. Nadaraya, "On estimating regression," *Theory of Probability & Its Applications*, vol. 9, no. 1, pp. 141–142, 1964. DOI: 10.1137/1109020. eprint: https://doi.org/10.1137/1109020. [Online]. Available: https://doi.org/10.1137/1109020 (return to page 32).

[42] G. S. Watson, "Smooth regression analysis," *Sankhyā: The Indian Journal of Statistics, Series A (1961-2002)*, vol. 26, no. 4, pp. 359–372, 1964, ISSN: 0581572X. [Online]. Available: http://www.jstor.org/stable/25049340 (visited on 04/23/2024) (return to page 32).

[43] H. J. Bierens, "The nadaraya–watson kernel regression function estimator," in *Topics in Advanced Econometrics: Estimation, Testing, and Specification of Cross-Section and Time Series Models.* Cambridge University Press, 1994, pp. 212–247 (return to page 32).

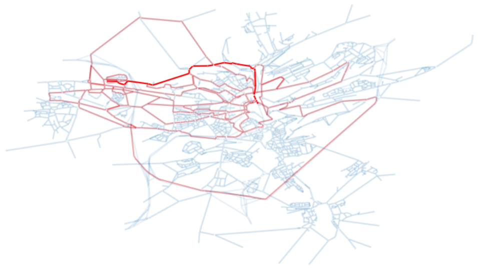
[44] K. Cabello-Solorzano, I. Ortigosa de Araujo, M. Peña, L. Correia, and A. J. Tallón-Ballesteros, "The impact of data normalization on the accuracy of machine learning algorithms: A comparative analysis," in *18th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2023)*, P. García Bringas *et al.*, Eds., Cham: Springer Nature Switzerland, 2023, pp. 344–353, ISBN: 978-3-031-42536-3 (return to page 46).

[45] S. Di-Castro, S. Mannor, and D. D. Castro, "Analysis of stochastic processes through replay buffers," in *Proceedings of the 39th International Conference on Machine Learning*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., ser. Proceedings of Machine Learning Research, vol. 162, PMLR, 17–23 Jul 2022, pp. 5039–5060. [Online]. Available: https://proceedings.mlr.press/v162/di-castro22a.html (return to page 48).

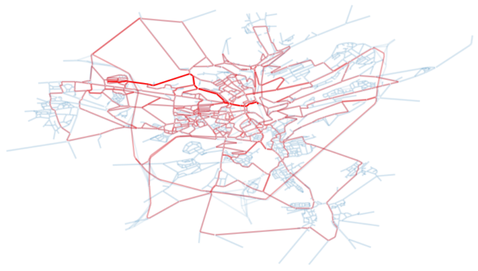
A

Appendix

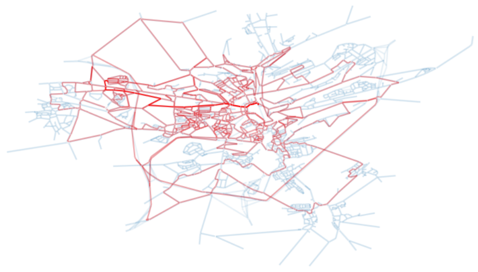
In this chapter, we will display the path selection of the agents in the evaluated environments.



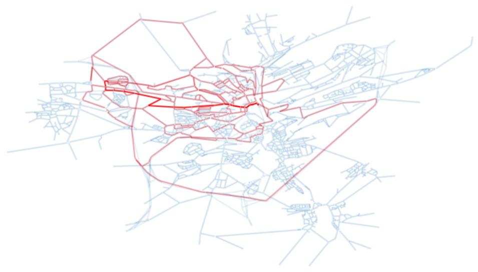
(a) Bayesian UCB



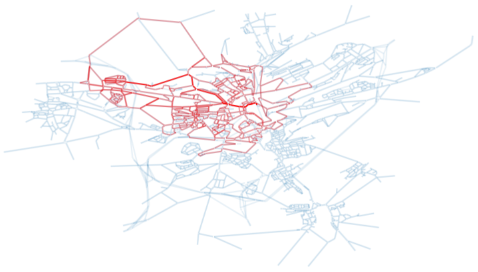
(b) Epsilon-Greedy



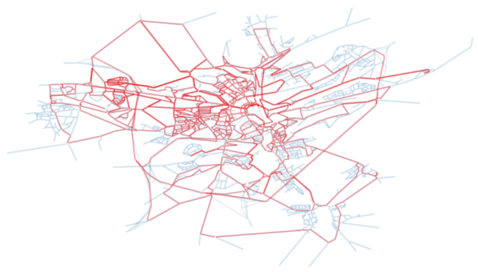
(c) LinTS



(d) LinUCB



(e) Thompson Sampling

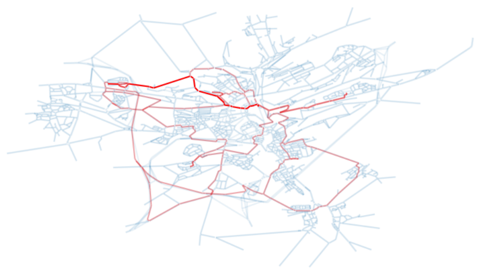


(f) TreeBoot DT

Figure A.1: Path selections for linear reward function problem instance 1

(a) TETS RF

(b) TETS XGB


(c) TEUCB RF

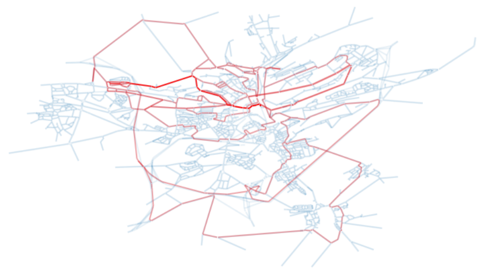
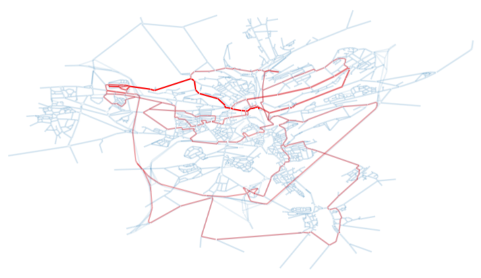
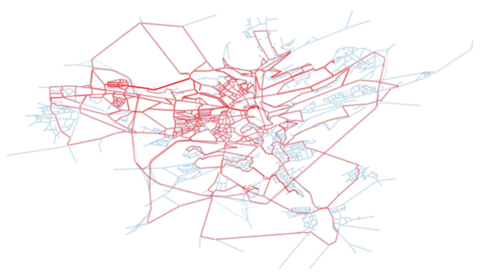

(d) TEUCB XGB

Figure A.2: Path selections for linear reward function problem instance 1


(a) Neural TS

(b) Neural UCB


(c) TreeBoot RF

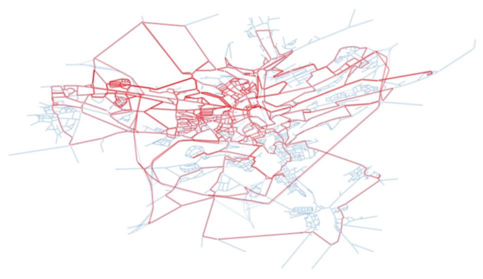
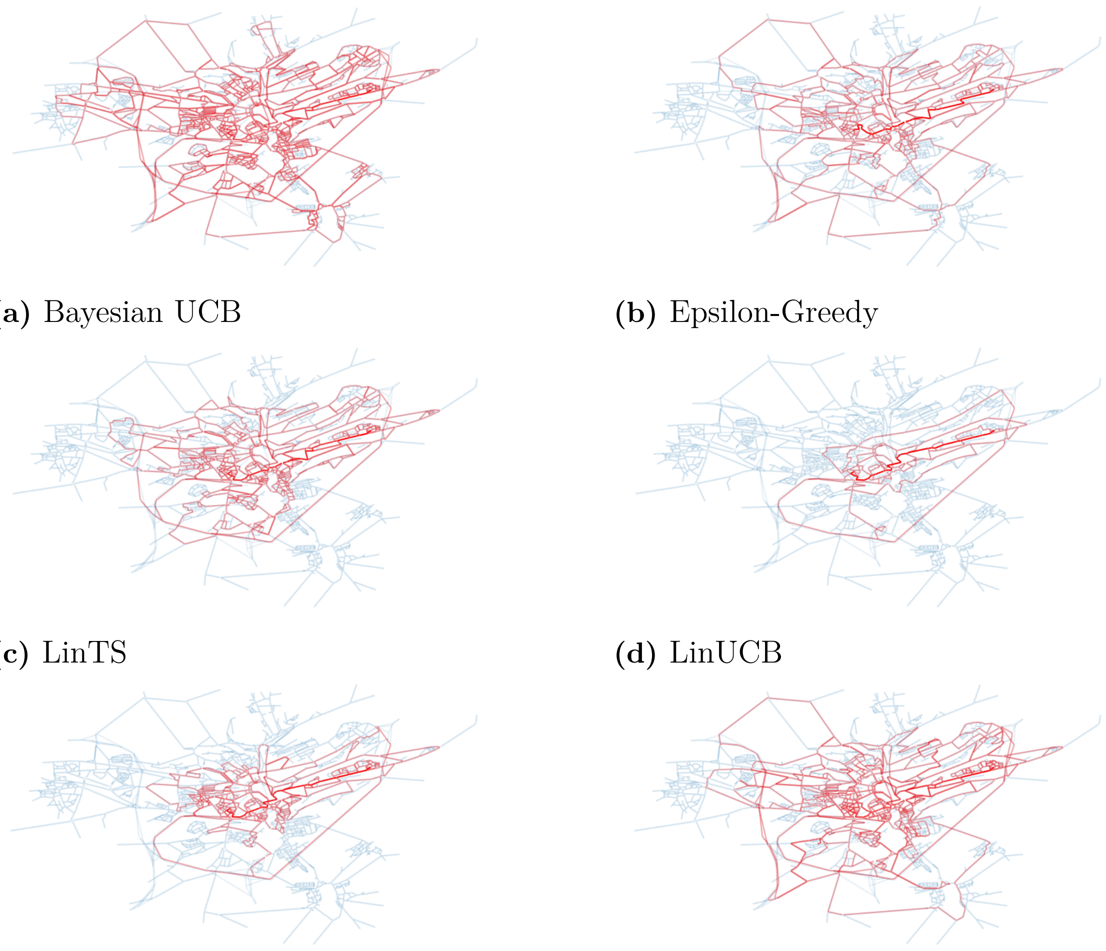

(d) TreeBoot XGB

Figure A.3: Path selections for linear reward function problem instance 1


(a) Bayesian UCB

(b) Epsilon-Greedy

(c) LinTS

(d) LinUCB

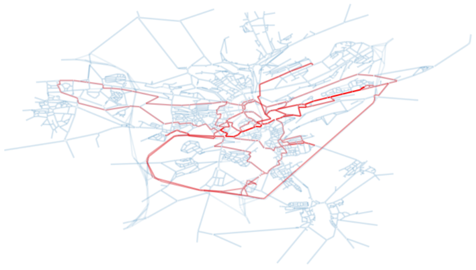
(e) Thompson Sampling

(f) TreeBoot DT

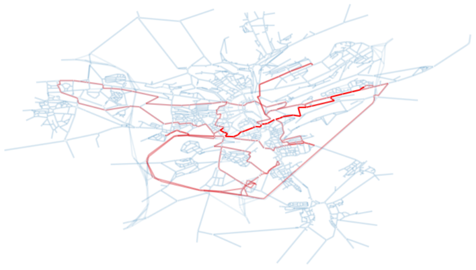
Figure A.4: Path selections for linear reward function problem instance 2

(a) TETS RF

(b) TETS XGB

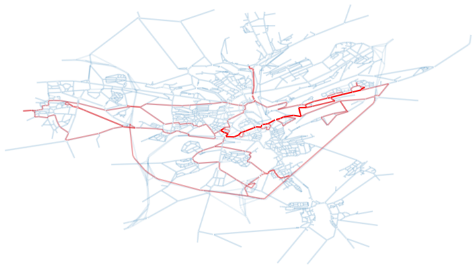


(c) TEUCB RF



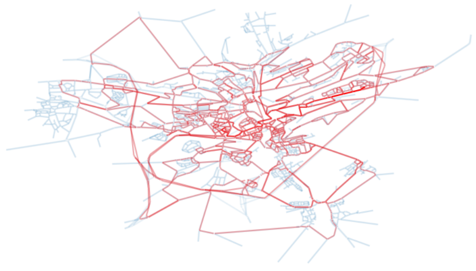
(d) TEUCB XGB

Figure A.5: Path selections for linear reward function problem instance 2

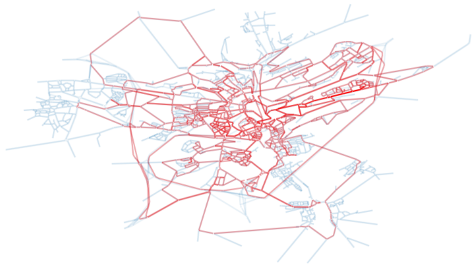


(a) Neural TS

(b) Neural UCB

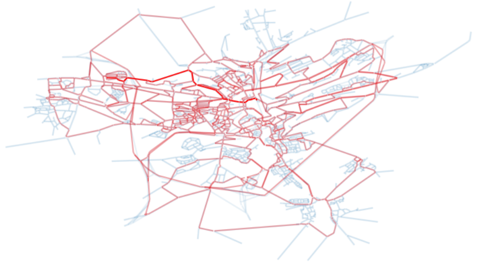


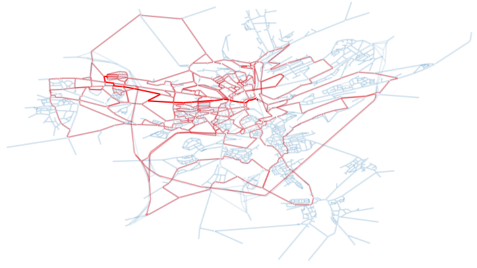
(c) TreeBoot RF

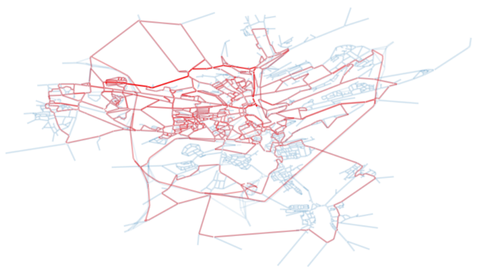


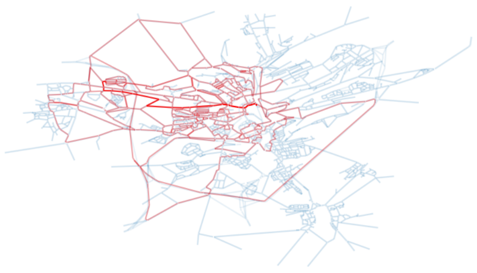
(d) TreeBoot XGB

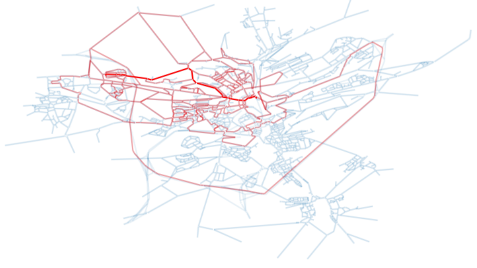
Figure A.6: Path selections for linear reward function problem instance 2


(a) Bayesian UCB


(b) Epsilon-Greedy


(c) LinTS


(d) LinUCB


(e) Thompson Sampling

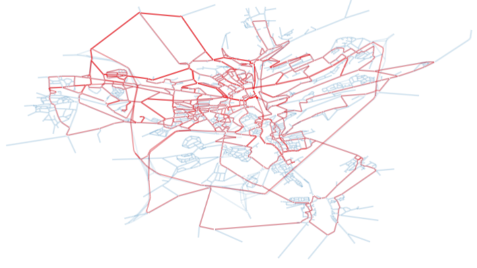
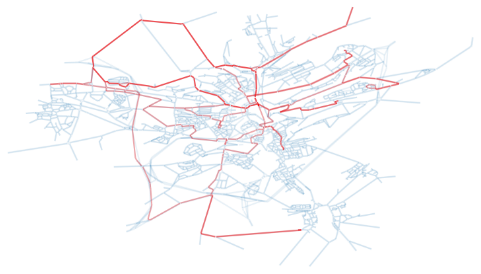

(f) TreeBoot DT

Figure A.7: Path selections for piece-wise constant reward function problem instance 1

(a) TETS RF

(b) TETS XGB


(c) TEUCB RF

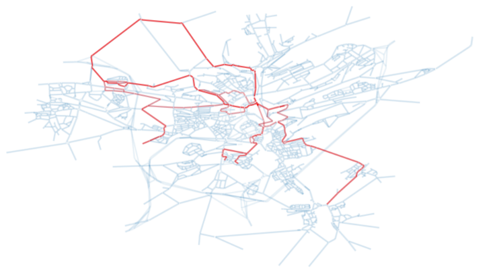
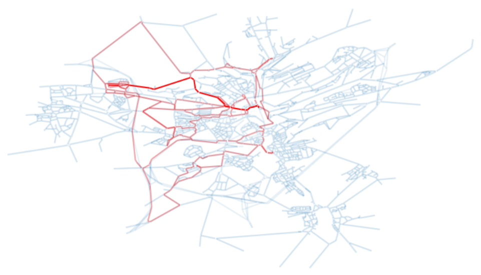
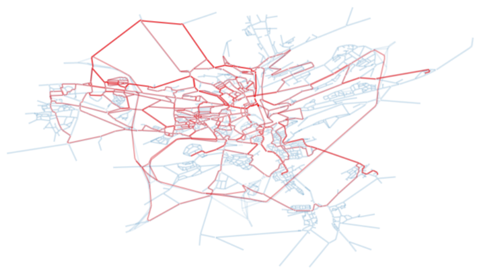

(d) TEUCB XGB

Figure A.8: Path selections for piece-wise constant reward function problem instance 1


(a) Neural TS

(b) Neural UCB


(c) TreeBoot RF

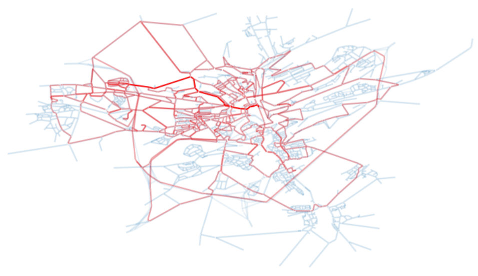
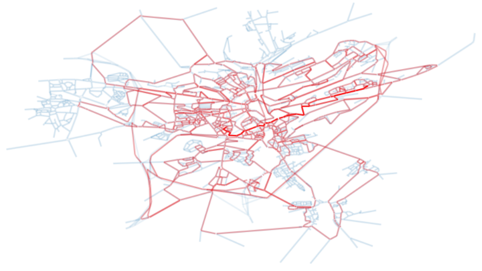
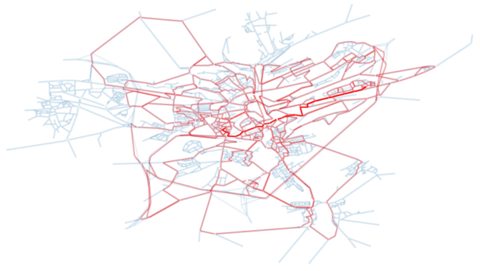

(d) TreeBoot XGB

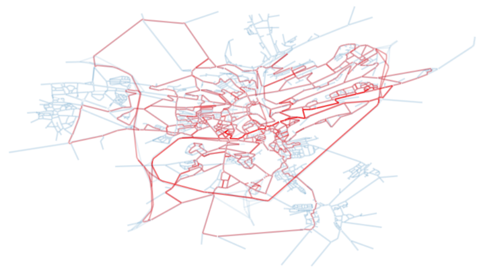
Figure A.9: Path selections for piece-wise constant reward function problem instance 1



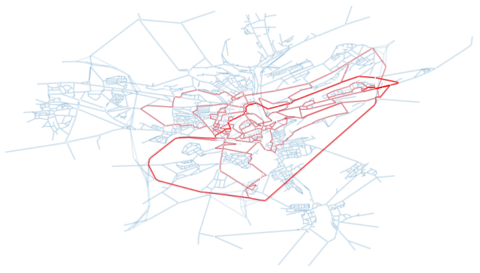
(a) Bayesian UCB



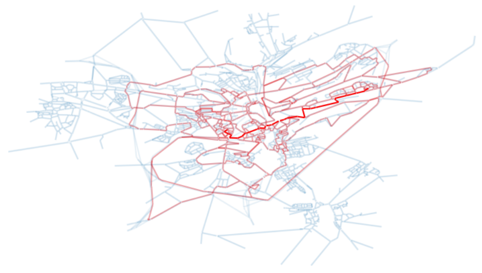
(b) Epsilon-Greedy



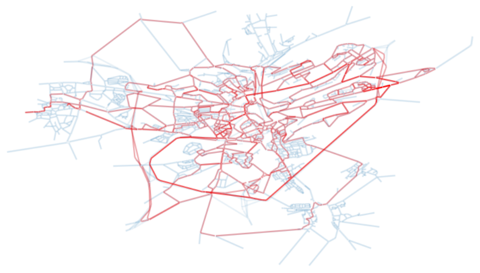
(c) LinTS



(d) LinUCB



(e) Thompson Sampling

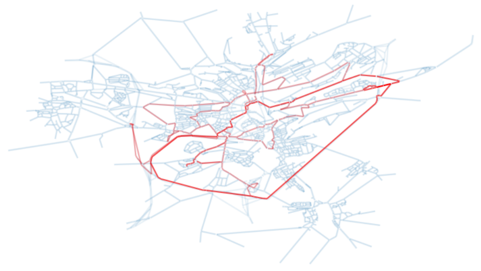


(f) TreeBoot DT

Figure A.10: Path selections for piece-wise constant reward function problem instance 2

(a) TETS RF

(b) TETS XGB


(c) TEUCB RF

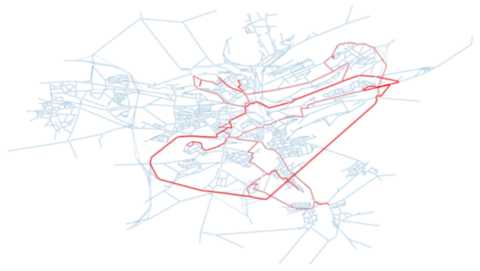
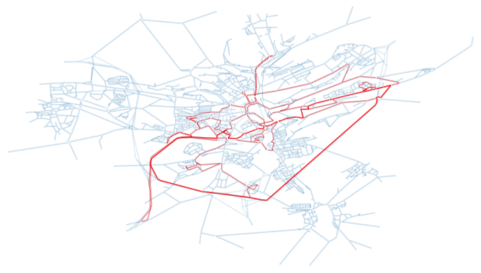
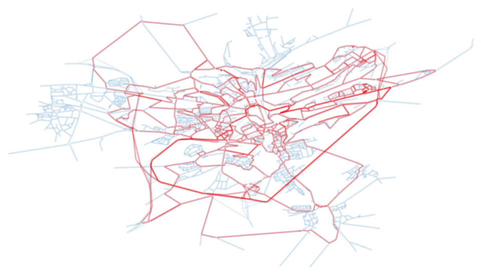

(d) TEUCB XGB

Figure A.11: Path selections for piece-wise constant reward function problem instance 2


(a) Neural TS

(b) Neural UCB


(c) TreeBoot RF

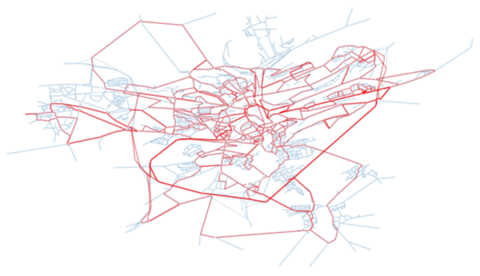
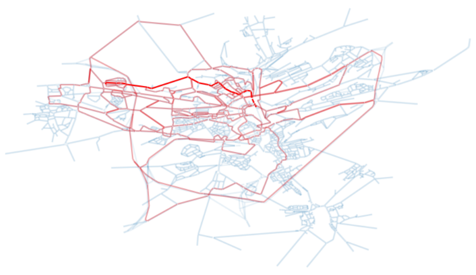
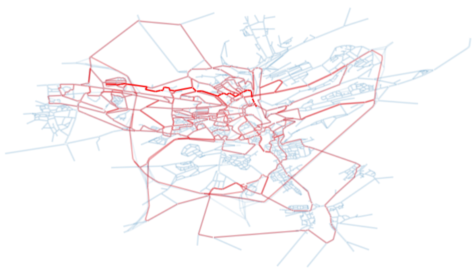
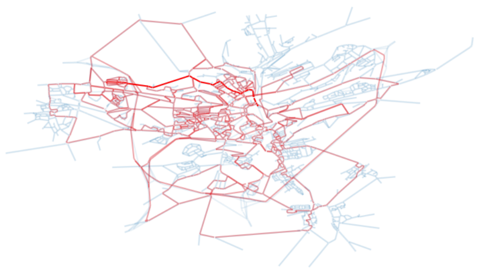
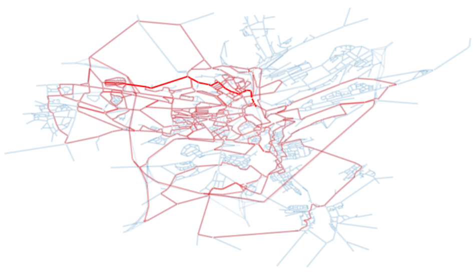

(d) TreeBoot XGB

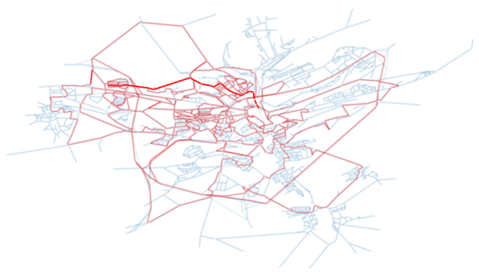
Figure A.12: Path selections for piece-wise constant reward function problem instance 2


(a) Bayesian UCB


(b) Epsilon-Greedy


(c) LinTS


(d) LinUCB


(e) Thompson Sampling

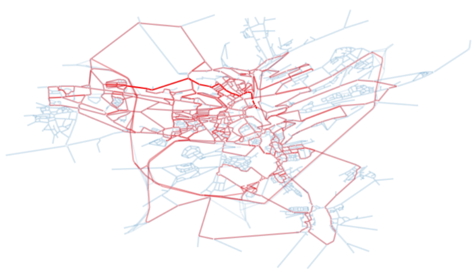
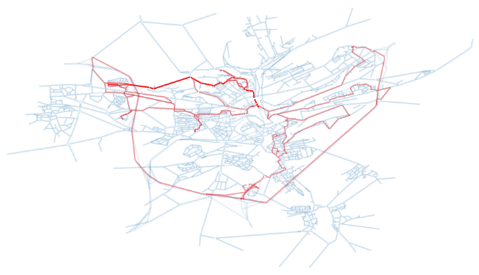

(f) TreeBoot DT

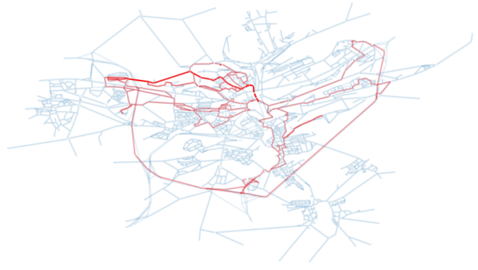
Figure A.13: Path selections for real-world reward function problem instance 1

(a) TETS RF

(b) TETS XGB

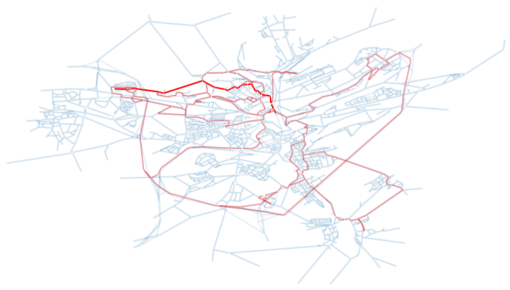


(c) TEUCB RF



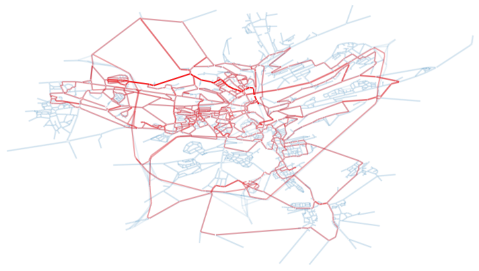
(d) TEUCB XGB

Figure A.14: Path selections for real-world reward function problem instance 1

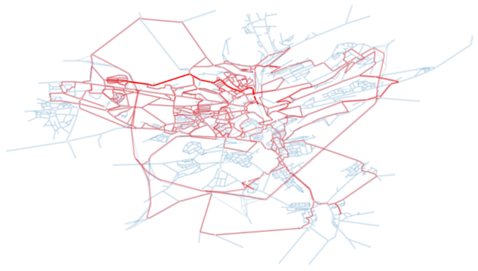


(a) Neural TS

(b) Neural UCB

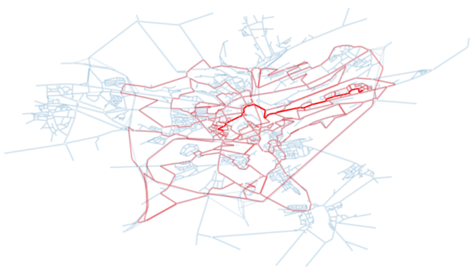


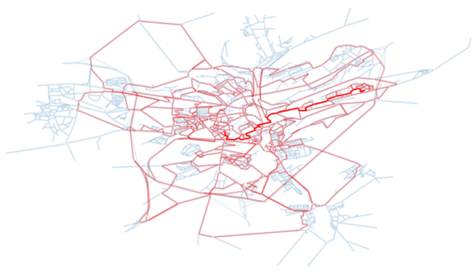
(c) TreeBoot RF

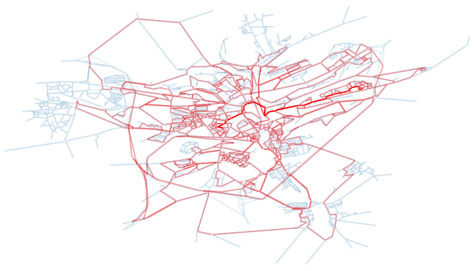


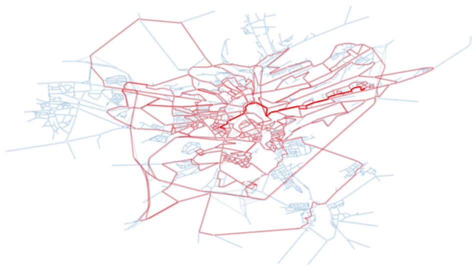
(d) TreeBoot XGB

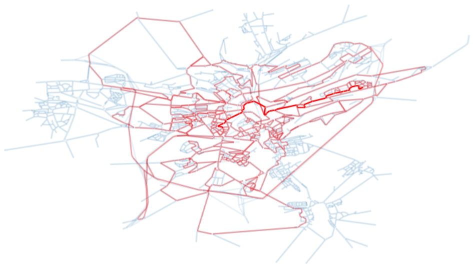
Figure A.15: Path selections for real-world reward function problem instance 1


(a) Bayesian UCB


(b) Epsilon-Greedy


(c) LinTS


(d) LinUCB


(e) Thompson Sampling

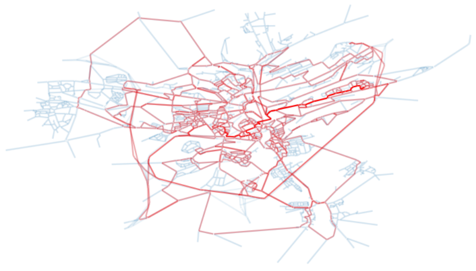
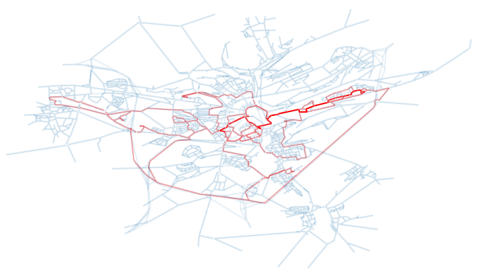
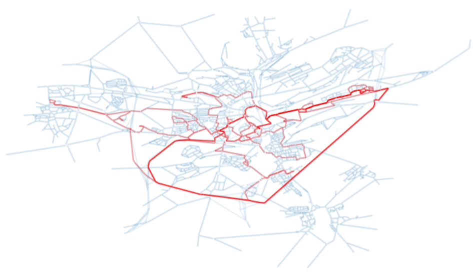
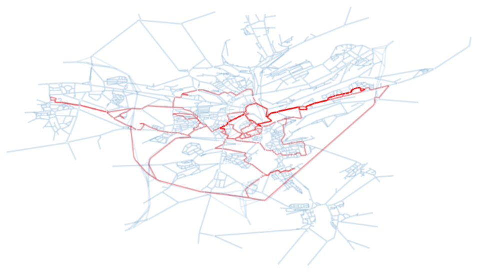

(f) TreeBoot DT

Figure A.16: Path selections for real-world reward function problem instance 2


(a) TETS RF


(b) TETS XGB


(c) TEUCB RF

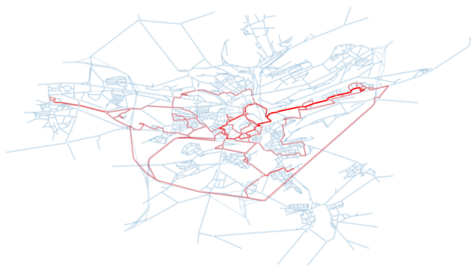
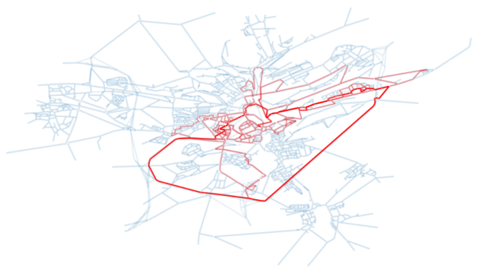
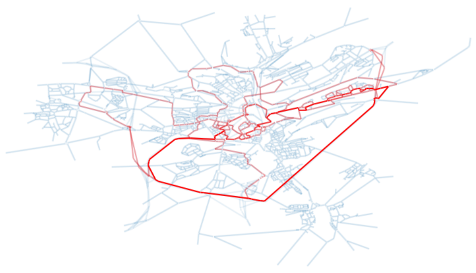
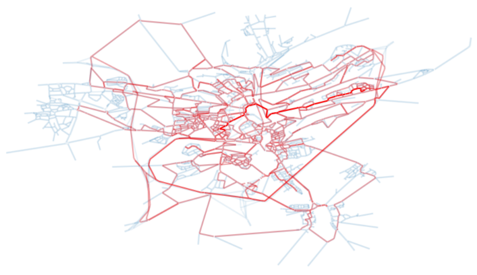

(d) TEUCB XGB

Figure A.17: Path selections for real-world reward function problem instance 2


(a) Neural TS


(b) Neural UCB


(c) TreeBoot RF

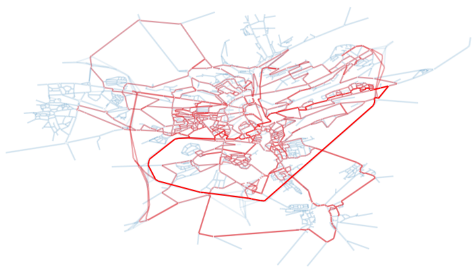

(d) TreeBoot XGB

Figure A.18: Path selections for real-world reward function problem instance 2