UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

CANDY ALEXANDRA HUANCA ANQUISE

Multi-objective reinforcement learning methods for action selection: dealing with multiple objectives and non-stationarity

Thesis presented in partial fulfillment of the requirements for the degree of Master of Computer Science

Advisor: Prof. Dr. Ana L. C. Bazzan

Porto Alegre
September 2021

CIP — CATALOGING-IN-PUBLICATION

Huanca Anquise, Candy Alexandra

Multi-objective reinforcement learning methods for action selection: dealing with multiple objectives and non-stationarity / Candy Alexandra Huanca Anquise. – Porto Alegre: PPGC da UFRGS, 2021.

58 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2021. Advisor: Ana L. C. Bazzan.

1. Multi-objective decision-making. 2. Multi-objective route choice. 3. Reinforcement learning. I. C. Bazzan, Ana L.. II. Título.

“You don’t need a reason to help other people.”

— Z. T.

ACKNOWLEDGMENTS

This work would not have been possible without the support of many people. I would like to thank my advisor, Prof. Dr. Ana L. C. Bazzan, whose feedback was invaluable in this entire research process. I appreciate even more her patience and encouragement. Also, thanks to Prof. Anderson Rocha Tavares for his guidance and support.

I would like to acknowledge my colleagues from Maslab, who are very kind people and always ready to help, even when the topic in discussion was not related to this research.

I would also like to thank the CAPES program for the postgraduate scholarship that made possible my stay in Brazil.

I am grateful to my friends for their support and company during these two years.

Finally, I owe my deepest gratitude to my family. Their unconditional love and support gained a different meaning during this work, and I am thankful for that.

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

DUCB	Discounted Upper-Confidence Bound
MAB	Multi-armed bandits
MAS	Multi-agent system
MDP	Markov Decision Process
MOMDP	Multi-objective Decision Process
PQL	Pareto Q-learning
PUCB1	Pareto Upper-Confidence Bound 1
QL	Q-learning
RL	Reinforcement learning
SWUCB	Sliding-window Upper-Confidence Bound
UCB	Upper-Confidence Bound

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Most real-world decision problems involve multiple and, usually, conflicting criteria. Multi-objective decision-making entails planning based on a model to find the best policy to solve such problems. If this model is unknown, learning through interaction provides the means to behave in the environment. Multi-objective decision-making in a multi-agent system poses many unsolved challenges. Among them, multiple objectives and non-stationarity, caused by simultaneous learners, have been addressed separately so far. In this work, algorithms that address these issues by taking strengths from different methods are proposed and applied to a route choice scenario formulated as a multi-armed bandit problem. Therefore, the focus is on action selection. In the route choice problem, drivers must select a route while aiming to minimize both their travel time and toll. The proposed algorithms take and combine important aspects from works that tackle only one issue: non-stationarity or multiple objectives, making possible to handle these problems together. The methods used from these works are a set of Upper-Confidence Bound (UCB) algorithms and the Pareto Q-learning (PQL) algorithm. The UCB-based algorithms are Pareto UCB1 (PUCB1), the discounted UCB (DUCB) and sliding window UCB (SWUCB). PUCB1 deals with multiple objectives, while DUCB and SWUCB address non-stationarity in different ways. PUCB1 was extended to include characteristics from DUCB and SWUCB. In the case of PQL, as it is a state-based method that focuses on more than one objective, a modification was made to tackle a problem focused on action selection. Results obtained from a comparison in a route choice scenario show that the proposed algorithms deal with non-stationarity and multiple objectives, while using a discount factor is the best approach. Advantages, limitations and differences of these algorithms are discussed.

Keywords: Multi-objective decision-making. Multi-objective route choice. Reinforcement learning.

1 INTRODUCTION

Decision-making and reinforcement learning (RL) are turning increasingly popular in multi-agent systems (MAS). Decision-making involves planning by selecting a course of action, assuming that a model of the environment is known. If that model is unknown, learning through interaction (RL) may be done to solve a problem.

The interaction between multiple agents deployed in an environment can make learning challenging when there are many of them. Moreover, simultaneous learning leads to a severe non-stationarity issue. From the point of view of an agent, the environmental dynamics change constantly due to multiple agents concurrently trying to improve their policies considering only their own interests.

While most approaches for MAS focus on optimizing an agent's policy regarding a single objective, many tasks are more naturally described by multiple, possibly conflicting, objectives. In multi-objective MAS, the reward signal is a vector, where each element corresponds to an objective. Considering a multi-objective perspective on decision-making problems makes trade-offs between objectives possible.

This work uses a repeated game to model the interaction between agents, which implies basically a single state. While this kind of formulation somehow means a simplified modeling, such a composition is useful in real-world problems involving the selection of several kinds of actions, while also providing a challenging problem for independent learning agents. While many multi-agent RL algorithms assume the agent can observe the actions or rewards of the other agents in the game, this can be unrealistic, as this information may not be available (NOWÉ; VRANCX; HAUWERE, 2012). RL methods must deal with non-stationary rewards that are influenced by other agents. For this class of RL problems, some algorithms have been proposed, such as the UCB family (by using a multi-armed bandits formulation), learning automata, and even simplifications of Q-learning (QL).

A wide range of domains can benefit from a multi-objective multi-agent approach using a repeated game modelling. An example is route choice.

In route choice, a large number of drivers must select a path to travel from an origin to a destination. Their route choices typically depend on minimizing their travel times. However, these choices are influenced by more than just one cost function, e.g., travel time and toll costs. One relevant concern is that the resources are scarce, thus agents compete for them and this makes the learning task much more difficult.

Thus, route choice involves a competition for resources, while the simultaneous learning of the agents results in a non-stationary environment. Most existing works deal with these problems separately, focusing in a single-agent scenario with multiple objectives or tackling only the non-stationarity issue. Proposals like Pareto UCB1 (DRUGAN; NOWÉ, 2013), henceforth PUCB1, and Pareto Q-learning (MOFFAERT; NOWé, 2014), henceforth PQL, deal with a multi-objective scenario but focus on a single agent, though their approaches differ. PUCB1 is a multi-armed bandits algorithm, while PQL is a state-based algorithm that works in a deterministic environment.

Non-stationarity was addressed by other members of the UCB family, such as the discounted UCB (DUCB) (KOCSIS; SZEPESVÁRI, 2006) and sliding-window UCB (SWUCB) (GARIVIER; MOULINES, 2011). However, these do not deal with more than one objective. Meanwhile, works in multi-objective reinforcement learning (MORL), considering multiple agents, are recent (RADULESCU et al., 2020). Most existing proposals for extending those popular algorithms for the multi-objective case concentrate on single-agent scenarios. Therefore, the issues related to competition for scarce resources, as well as non-stationarity, need further investigation when we deal with multi-objective RL.

The present work addresses these challenges by proposing, implementing, and testing extensions of algorithms that were proposed for multi-objective decision-making in a MAS. Strengths from other works focused solely on either non-stationarity or multiple objectives are taken and combined to produce algorithms applied to a route choice scenario, where thousands of commuters have to pick a route from their origins to their destinations. This problem is addressed from a decentralized perspective, which is more realistic than using a centralized traffic assignment approach. After all, a commuter does not have full knowledge of all possible routes and, especially, is not aware of all costs at all times. Two objectives are considered for the sake of evaluation: travel time and a monetary cost that can be a toll.

Then, this work focuses on if we can deal successfully with multiple objectives and non-stationarity jointly by taking strengths from other works that address these problems separately. To this end, this dissertation presents the following contributions:

- Extensions to PUCB1 are introduced to cope with the severe non-stationarity of a multi-agent setting. PUCB1 works on multi-objective scenarios with a single agent, while non-stationarity was tackled by other members of the UCB family, such as DUCB and SWUCB, which deal with a single objective. It is unclear whether these

variants of the UCB can indeed handle the kind of non-stationarity caused by other agents learning simultaneously, where the pace of change is continuous or at least very fast. It seems that the extensions of UCB are able to deal successfully with changes that happen in large intervals of time, which is not the same type of non-stationarity that arises due to multiple learners.

- A modification of the PQL algorithm. While this algorithm deals with multiple objectives, it is intended to work in a deterministic environment. Changes are introduced in order to enable its use in a rapidly changing environment whilst employing a single state and focusing on action selection.

Experiments were carried out in a 4-node network with thousands of drivers, whose objectives were to minimize travel time and toll. Results show that the proposed algorithms address the issues of both non-stationarity and multiple objectives by combining strengths from methods that tackle only one of those issues. The UCB-based algorithms present more flexibility when compared to the algorithm based on PQL.

This work is organized as follows. Chapter 2 covers concepts, notations and related work of reinforcement learning and traffic assignment. Chapter 3 discusses the proposed algorithms based on methods provided in Chapter 2. Subsequently, Chapter 4 focuses on the experiments, the scenario where they were carried out, a discussion of the results and some metrics that can be used for comparison. Finally, Chapter 5 contains a summary of the contributions of this work and the conclusions derived from it.

2 BACKGROUND AND RELATED WORK

This chapter presents concepts, definitions and works related to RL and multi-objective decision-making (Sections 2.1 - 2.3). Particularly, the algorithms which this work is based on are addressed in Section 2.3. The traffic assignment problem is introduced in Section 2.4, including the single-objective and multi-objective case. The chapter finalizes by giving an overview, in Section 2.5, of the RL algorithms that deal with the issues of non-stationarity and multiple objectives, being separately or not.

2.1 Markov Decision Process (MDP)

Sequential decision problems are represented by Markov Decision Processes (MDPs), where actions influence immediate rewards and also subsequent states, extending this influence to future rewards.

In general, MDPs assume that the Markov hypothesis is valid. This means that the system satisfies the Markov property: the current state of the system is all the information that is needed to decide which action to take. There is no dependence on previous states, actions or rewards.

2.1.1 Single-objective Markov Decision Process

A single-objective MDP is a tuple $\langle S, A, T, R \rangle$ (BELLMAN, 1957), where:

- S is a set of states
- A is a set of actions
- T is a state transition function $T : S \times A \times S \rightarrow [0, 1]$ that gives the probability of a next state given an action and a current state
- R is a reward function $R : S \times A \rightarrow \mathbb{R}$

In this setting, an MDP models an agent acting in a stochastic environment where the objective is to maximize a long-term measure of total reward, which is called return.

A policy is a mapping from states to actions, defined over the entire state A policy is a mapping from states to actions, defined over the entire state space. It determines what to do given knowledge of the current state.

Solving an MDP means to find an action sequence that, starting from any state, yields the maximum possible expected return that can be achieved starting from that state. This solution is denoted by the optimal policy π^*.

2.1.2 Multi-objective Markov Decision Process (MOMDP)

A multi-objective MDP (MOMDP) is an extension of MDP used to represent a multi-objective problem. In this model (ROIJERS et al., 2013), the reward is no longer represented by a scalar. A vector of n rewards, one for each objective, is used instead.

A MOMDP is a tuple $\langle S, A, T, R \rangle$, where:

S, A, T	are the same as in an MDP
R	is a reward function $R : S \times A \rightarrow \mathbb{R}^n$

In general, the goal is to find a coverage set of non-dominated policies.

The presence of a reward vector implies the need of information to prioritize objectives and that a partial ordering must be defined. For example, a policy can outperform another with respect to one objective, but its performance might be worse when considering another objective. Consequently, it is not possible to determine which policy is better without additional information about how to prioritize the objectives.

2.2 Multi-objective decision making

Sequential decision making, modeled as MDPs, is present in many real-world tasks. The utility or desirability of actions concerning such tasks is mostly represented by a scalar value. However, most real-world problems are naturally multi-objective and can be modeled by an MOMDP.

Scalarization, by using a set of weights, makes possible to apply single-objective methods to the multi-objective problem. Each weight quantifies the relative importance of the corresponding objective. However, it is not always possible or feasible to do this scalarization *a priori*. In (ROIJERS et al., 2013), three scenarios of multi-objective decision making were identified.

The first scenario deals with unknown weights. A set of policies must be calculated first. When the information about the weights becomes available, the best

policy is selected from the set by using those weights.

In the decision support scenario, the weights are never explicitly known and scalarizing is not possible during the decision-making process. A set of policies is computed, and the user has to make the final decision according to their arbitrary preferences.

In the known weights scenario, scalarizing is possible. Despite that, if the function used for scalarizing is non-linear, the MDP may be difficult to solve. Even if the function is linear, the problem can become intractable. It is preferable to use a multi-objective method in that case.

2.3 Reinforcement learning (RL)

The standard RL method consists of the repeated interaction between an active decision-making agent and its environment (SUTTON; BARTO, 2018). In each time step, the agent perceives the current state of the environment and selects an action. The environment's response is a reward signal and a new perception which will form a new state. Thus, a RL problem is modeled as an MDP.

RL implies determining a mapping of situations (states) for behaviors (actions) to maximize the total future rewards. Two of its most important features are the trial-and-error method and the delayed reward. The agent does not know *a priori* which actions it should take to maximize its reward, hence the need to try them. However, the agent's actions may not only determine the immediate reward but they may also affect subsequent rewards. As such, the agent must learn which action is the most desirable considering the immediate reward and a delayed reward that can take place far in the future.

One popular RL method is Q-learning (QL). Proposed by (WATKINS, 1989), QL is a model-free algorithm used to learn the value of an action in a state. Being model-free means QL does not explicitly learn transition probabilities or reward functions. Rather, QL estimates the optimal policy directly from experience (i.e., the interaction between the agent and the environment).

QL approximates state-action values, called Q-values, which are numeric estimates of the quality for a state-action pair. A $Q(s, a)$ value represents the maximum discounted sum of the future rewards that an agent can receive if it begins in state s, chooses action a and continues with the optimal policy π^* (SUTTON; BARTO, 2018). Q-values are updated as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

Where:

- s is the current state
- a is the action taken
- s' is the next state
- a' are all actions that were probed in state s'
- α is the learning rate
- r is the reward received when moving from state s to state s' by taking action a
- γ is the discount factor

The algorithm learns an action-value function Q which approximates the optimal one while the agent interacts with its environment. When the Q-values are near the optimal values, the most appropriate policy is the greedy one, e.g., to choose the action with the highest $Q(s,a)$.

Q-Learning is also an off-policy algorithm. This means that it approximates the optimal Q function, independently of the policy used for action selection.

2.3.1 Multi-armed bandits (MAB)

The MAB problem is one of the simplest RL problems, as it is stateless (or considers a single state). It was first studied in (THOMPSON, 1933) and later described by (ROBBINS, 1952). A basic MAB problem consists of a set of K arms (or actions) with different reward distributions, and T rounds (or plays). The player selects an arm in each round and collects its reward. The goal is to maximize the total reward obtained.

The MAB problem is an instance of sequential decision making and offers a theoretical formulation for analyzing the exploration-exploitation dilemma in sequential experiments. The player needs to balance reward maximization based on the knowledge gained (exploitation) and attempt new actions to acquire new information (exploration).

Regarding the MAB problem, the Upper-Confidence Bound (UCB) family of algorithms was proposed by (AUER; CESA-BIANCHI; FISCHER, 2002). In this model, the reward for each arm is drawn independently from a fixed distribution that is different

for each action.

In UCB1, the simplest algorithm, the arm a that maximizes the upper bound of a confidence interval for the arm's expected reward is selected. The bound is the sum of two terms:

$$\bar{R}(a) + \sqrt{\frac{2 \ln n_e}{N_t(a)}}$$

The first term is the current average reward $\bar{R}(a)$. The second term is a padding function related to a one-sided confidence interval of the average reward (AUER; CESA-BIANCHI; FISCHER, 2002). The second term grows with the total number of actions the player has taken (n_e) but decreases with the number of times this particular action has been tried ($N_t(a)$). If an action was played several times, its average reward is considered accurate and the value of the padding function is small. If the action was played a few times, the padding function has a higher value, representing the lower accuracy of its average reward. Thus, an action that has not been explored as often as other actions will have a bigger padding function. This makes the algorithm explore quickly unknown actions, by initially playing each arm once, before engaging in selecting the more promising action.

There are variants proposed to deal with the non-stationary MAB problem. In this problem, the distributions of the arms are modeled by a sequence of independent random variables from different distributions which may vary through time. The discounted UCB algorithm (DUCB) (KOCSIS; SZEPESVÁRI, 2006) was proposed to deal with non-stationarity. It is an adaptation of the UCB algorithm and works by averaging all past rewards while using a discount factor γ to give more weight to more recent observations. Another variant of UCB is sliding-window UCB (SWUCB) (GARIVIER; MOULINES, 2011). It also addresses the non-stationary case, but averages the past rewards using only the last w plays.

2.3.2 Multi-objective reinforcement learning (MORL)

The algorithms used to solve multi-objective problems can be developed by planning, given a model of the MDP, or by learning through interaction with an unknown MDP. The latter case refers to MORL.

Regarding this work, there are two MORL algorithms that will be explained: PUCB1 and PQL. As their names indicate, they are based on QL and the UCB1

algorithm, respectively.

2.3.2.1 Pareto UCB1 (PUCB1)

The PUCB1 algorithm was proposed in (DRUGAN; NOWÉ, 2013) as an extension of the standard UCB1 algorithm for the multi-objective MAB problem using the Pareto dominance relationship. In the multi-objective setting, the reward vector $\vec{R}$ contains one dimension for each objective.

Similar to the UCB1 algorithm, during the t-th episode, for each action (i.e., arm) a, the sum of two terms is calculated:

– The average reward vector $\bar{R}(a)$. Eq. 2.1 shows how $\bar{R}(a)$ is calculated, where $\vec{R}(a)$ is the reward vector received and $N(a)$ is a scalar and the number of times action a was selected (i.e., the number of times arm a was played).

$$\bar{R}(a) = \bar{R}(a) + \frac{\vec{R}(a) - \bar{R}(a)}{N_t(a)} \tag{2.1}$$

– A padding function $c_t(a)$. It corresponds to the upper bound of a confidence interval of the average reward, as shown in Eq. 2.2, where $|O|$ denotes the number of objectives, $\mathcal{A}^*$ is the set of Pareto optimal actions and n_e is the number of episodes or plays.

$$c_t(a) = \sqrt{\frac{2\ln(n_e \sqrt[4]{|O||\mathcal{A}^*|})}{N_t(a)}} \tag{2.2}$$

If $\mathcal{A}^*$ is not known *a priori*, the term $|\mathcal{A}^*|$ can be replaced by the number of actions $|K|$.

In each episode, aiming to maximize all objectives, the Pareto set $\mathcal{A}'$ is found, such that $\forall \ell \notin \mathcal{A}'$, there exists an action $a \in \mathcal{A}'$ that dominates action ℓ, as stated by Eq. 2.3, where the padding function (a scalar) is added to each of the elements of the average reward vector. An action is selected uniformly at random from $\mathcal{A}'$.

$$\bar{R}(\ell) + c_t(\ell) \not\succ \bar{R}(a) + c_t(a) \tag{2.3}$$

Exploitation is satisfied by the average reward vector, while exploration depends on the upper bound represented by the padding function. The agent explores more during the initial episodes, allowing the less pulled arms to be eventually chosen against the arms with a high average reward.

2.3.2.2 Pareto Q-learning (PQL)

PQL (MOFFAERT; NOWé, 2014) integrates the Pareto dominance relation into a MORL approach. Based on QL, PQL is a state-based method that separates the immediate reward vector from the set of expected future discounted reward vectors to allow calculating Q-sets, denoted by $\tilde{Q}_{set}$. Such Q-sets are composed of vectors. As PQL was defined for a state-based case, the set of expected future discounted reward vectors relies on a function, called ND, that finds those vectors that correspond to the possible future states, and which are not Pareto-dominated.

Eq. 2.4 shows how Q-sets are calculated. $\bar{R}(s,a)$ denotes the immediate reward vector and $ND_t(s,a)$, the set of non-dominated vectors in the next state of s that is reached through action a at time step t. $\bar{R}(s,a)$ is added to each element of $\gamma ND_t(s,a)$. When the action a in state s is selected, both terms are updated. $\bar{R}(s,a)$ is updated according to Eq. 2.5, where $\vec{R}$ is the new reward vector and $N(s,a)$ is the number of times action a was selected in s. $ND_t(s,a)$ is updated as shown in Eq. 2.6, using the non-dominated vectors in the $\tilde{Q}_{set}$ of every action a' in the next state s'.

$$\tilde{Q}_{set}(s,a) = \bar{R}(s,a) \oplus \gamma ND_t(s,a) \tag{2.4}$$

$$\bar{R}(s,a) = \bar{R}(s,a) + \frac{\vec{R} - \bar{R}(s,a)}{N(s,a)} \tag{2.5}$$

$$ND_t(s,a) = ND(\cup_{a'} \tilde{Q}_{set}(s',a')) \tag{2.6}$$

PQL learns the entire Pareto front, finding multiple Pareto optimal solutions (policies), provided that each state-action pair is sufficiently sampled. This algorithm is not biased by the Pareto front shape (algorithms that find a single policy and use scalarization can not sample the entire Pareto front if it is non-convex) or a weight vector (it guides the exploration to specific parts of the search space).

Note that it is assumed that the environment is deterministic, hence PQL is not directly useful for environments in which the presence of multiple agents learning simultaneously causes non-stationarity.

2.3.2.3 Other methods

There are other algorithms that have been proposed in the MORL literature, mostly focusing on a single-agent scenario, such as (MOFFAERT et al., 2014), where they focused on sampling actions that were Pareto-dominating from the objective space in a MAB setting, proposing a strategy to discretize the continuous action space according to such samples and approximating the Pareto front.

In (DRUGAN; NOWÉ, 2014b), an approach was proposed to identify the Pareto optimal set of actions and the minimum subset of scalarization functions that optimize such set, in order to avoid the need to use MAB to find the entire Pareto front when it is not evident which scalarization functions should be used.

An hypervolume-based MORL algorithm was proposed in (MOFFAERT; DRUGAN; NOWÉ, 2013), where the hypervolume unary indicator is used as action selection mechanism. This algorithm was compared to two scalarization-based MORL algorithms: linear scalarization and Chebyshev Function, outperforming the first and performing similarly to the second without needing to prioritize objectives based on user input.

In (ROIJERS; ZINTGRAF; NOWÉ, 2017), two algorithms were proposed (Utility-MAP UCB and Interactive Thompson Sampling) using online MORL with user interaction to learn the utility function of the user in a multi-objective MAB setting. They showed that these algorithms approximate the regret of UCB and Thompson Sampling, while Interactive Thompson Sampling outperforms Utility-MAP UCB.

2.4 Traffic assignment problem (TAP)

A road network can be modeled as a graph $G = (V, L)$, where V is the set of vertices that represent intersections, and L is the set of directed arcs (links) that describe road segments that connect a pair of vertices. There is a demand for trips between origin-destination (OD) pairs. The traffic assignment problem (TAP) aims at assigning these trips to the links of the road network, according to the traveler's route choice criteria (ORTÚZAR; WILLUMSEN, 2011).

2.4.1 Single-objective case

Central to the notion of the TAP is the idea of a rational traveler who wants to make a trip from an origin to a destination vertex, choosing the route with the least perceived individual cost. Several factors may influence this decision, such as travel time, distance, monetary cost, e.g., toll, consumption, battery, emission, etc. (ORTÚZAR; WILLUMSEN, 2011).

In the single-objective case, it is assumed that all drivers aim at minimizing a single objective: travel time. One solution for this optimization problem is via the method of successive averages (MSA). MSA is an iterative algorithm that calculates the current flow on a link as a linear combination of the flow on the previous iteration and an auxiliary flow produced by an all-or-nothing (AON) assignment in the present iteration (ORTÚZAR; WILLUMSEN, 2011). An AON assignment is a typical approach to trip assignment under no congestion, where all drivers are assigned to the shortest route.

As unrealistic as it is, AON is often the basis for iterative methods such as MSA, i.e., the first step. The linear combination is given by Eq. 2.7, where f_l^{t+1} is the current flow in link l, $\tilde{f}_l^t$ is the auxiliary flow and f_l^t is the previous flow in l.

$$f_l^{t+1} \leftarrow \omega^t \tilde{f}_l^t + (1 - \omega^t) f_l^t, \quad \omega^t \leftarrow 1/(t+1) \tag{2.7}$$

Regarding RL, the TAP can be solved in a decentralized fashion by letting agents (drivers) select routes using RL, be it by means of the bandit algorithms or by stateless QL. Normally, in this formulation, the goal is to reach the Nash or user equilibrium, as stated by Wardrop (WARDROP, 1952): "under equilibrium conditions traffic arranges itself in congested networks such that all used routes between an OD pair have equal and minimum costs while all those routes that were not used have greater or equal costs". This way, no agent can improve its own travel time by unilaterally switching routes. Each agent is rewarded by its travel time, i.e., a single objective underlies the reward function. Recall that this kind of learning task is associated with repeated games, in which there is a single state and actions are the selection of routes.

In (RAMOS; SILVA; BAZZAN, 2017), a regret-minimization method was proposed, in which the performance of each route in comparison to the best one experienced is used as an estimation of the regret.

In (OLIVEIRA et al., 2018), the authors compared stateless QL to MAB algorithms, concluding that several variants of the UCB algorithm do not outperform

QL, possibly due to the highly non-stationary nature of the route choice problem.

Departing from repeated games, (BAZZAN; GRUNITZKI, 2016) formulated the TAP as a stochastic game, where the states are the vertices and actions are the links that leave a vertex, and where the reward is as aforementioned.

2.4.2 Multi-objective case

While MSA assumes that each traveler aims at minimizing only its travel time, in fact, travelers' route choices depend on multiple cost functions, not just one. Traditionally, multi-objective traffic assignment is modeled by using a linear combination of the cost functions. For instance, as early as 1979, such a formulation was proposed by Dial (DIAL, 1979) for a bi-objective assignment.

However, using a generalized cost function has some drawbacks. It assumes the travelers make their choices based on a cost function which is effectively single-objective, as it is composed of a sum of components with different weights. This generalized cost function causes some efficient paths to be missed. As shown in (WANG; RAITH; EHRGOTT, 2010), shortest paths algorithms can only find supported efficient solutions at extreme points. There might be other efficient solutions at non-extreme points that could appeal to some travelers if they made their decision based on multiple criteria, rather than using a generalized cost function.

Given multiple objectives, it is necessary to have alternative solutions for methods such as the MSA. One issue here is that, in a multi-objective scenario, there is a set of efficient paths rather than a single shortest path, given that travelers are not only trying to improve their travel times, but other costs as well. To determine how users will choose their preferred path, it can be assumed that all of them are equally attractive. This means an equal share (EQS) assignment, as proposed in (WANG; RAITH; EHRGOTT, 2010), where an equal number of trips is assigned to each efficient path. The EQS assignment replaces the AON assignment in the original MSA. Again, while this may be unrealistic, it is a good start point for iterative methods.

Other works that deal with the multi-objective TAP are the aforementioned (DIAL, 1979), which uses a linear combination of the cost functions; (RAITH et al., 2014), which proposes heuristic solution algorithms based on the MSA and path equilibration; (WANG; EHRGOTT, 2013), which works with a bi-objective approach by using a time surplus to determine which path will be chosen by the individual, and (WANG; RAITH; EHRGOTT,

2010), which also is applied to a bi-objective scenario and proposes traffic assignment procedures to achieve a bi-objective equilibrium (the EQS assignment is among them). All of these works are centralized approaches.

However, it must be noted that centralized approaches imply a central authority is in charge of the process of assigning routes to drivers. These centralized methods present unrealistic assumptions, such as the existence of such central authority, a complete observation of network conditions and a communication channel between drivers and the authority. In real life, and from a driver's point of view, these assumptions may not be satisfied. Considering that, a multi-agent systems can provide a robust modeling of individual decision-making of the drivers in the road network.

2.5 Overview of algorithms

As mentioned, the TAP can be formulated as a single-state, action selection fashion. Thus, algorithms such as UCB and its variants (see Section 2.3.1) can be used. However, it is worth mentioning that these two variants (DUCB and SWUCB) were created primarily to deal with a single agent learning in a non-stationary environment with a single-objective. This differs from the non-stationarity that arises from simultaneous learning by many agents. While the latter is also attributed to the environment, its nature is such that changes occur much more frequently, thus leading to a more challenging environment.

As this work uses a MAS to model the multi-objective TAP problem, the main concerns lies in the subsequent non-stationarity and multiple objectives. While the former can be handled by the DUCB and the SWUCB, two algorithms were considered to deal with the latter: PUCB1 and PQL.

The first 5 lines of Table 2.1 summarize the gaps in the aforementioned algorithms, which refer to the inability to deal with both issues: multiple agents causing non-stationarity and multiple objectives. The next chapter then introduces the algorithms proposed in this work to address both problems. The features of these algorithms correspond to the last three lines of Table 2.1.

Table 2.1 – Comparison of algorithms

	Non-stationarity	*Multi-objective*
UCB	–	–
PUCB1	–	√
DUCB	√	–
SWUCB	√	–
PQL	–	√
DPUCB	√	√
SWPUCB	√	√
mPQL	√	√

Source: Author.

3 PROPOSED ALGORITHMS

In this chapter, algorithms to address non-stationarity and multiple objectives are detailed, based on methods discussed in Chapter 2. The first algorithms, described in Section 3.1, are UCB-based and combine aspects from PUCB1, DUCB and SWUCB. The third algorithm, detailed in Section 3.2, is a modification of the PQL method.

Before detailing the algorithms, the general formulation and representation is presented. A multi-agent, multi-objective RL problem, formulated as a repeated game, can be defined as a stateless multi-agent Markov decision process (MMDP), in which $\mathcal{G}$ is the set of agents, A is the set of actions, and $\mathcal{R}(a) : A \rightarrow \mathbb{R}^{|O|}$ is the reward function defined over the set O of objectives. Each element of the reward vector corresponds to an element of O.

The aforementioned MMDP is instantiated as follows for the route choice problem. A is a set K containing routes that take each agent from origin o to its destination d. The reward function is then $\mathcal{R}(a) : K \rightarrow \mathbb{R}^{|O|}$. Two objectives are evaluated: travel time and flow-independent toll. Eq. 3.1 shows the definition of $\mathcal{R}(a)$, where $a \in K$ is a route and $\vec{R}$ is a reward vector whose elements are the travel time and toll of route a. The travel time of route a is calculated as the sum of the travel times of its links. Its toll is calculated analogously.

$$\mathcal{R}(a) = \vec{R} \tag{3.1}$$

3.1 Discounted Pareto UCB (DPUCB) and Sliding window Pareto UCB (SWPUCB)

DPUCB takes a discount factor γ from the DUCB algorithm (see Section 2.3.1) to average past rewards and give more weight to recent observations. This discount factor is introduced in the calculations of the average reward vector and the padding function defined in Eq. 2.2 from PUCB1 (see Section 2.3.2.1). DPUCB also employs the Pareto dominance relation from Eq. 2.3 to determine the set of efficient solutions.

To deal with the non-stationarity that arises due to multiple agents learning simultaneously, DPUCB employs a random initialization phase where each agent selects each action once randomly, i.e., agents are prevented from choosing already selected actions. This ensures that not all agents choose the same action in each episode.

At episode t, for action a, the upper bound in Eq. 3.2 is composed of the discounted

average reward vector $\bar{R}_t(\gamma, a)$, as stated in Eq. 3.3 (where I_m is the action selected in episode m), and the discounted padding function $c_t(\gamma, a)$ defined by Eq. 3.4. As rewards are upper-bounded by B, its value was set to 1 in Eq. 3.4 since PUCB1 assumes the rewards are defined on the interval $[0, 1]$.

$$u_b = \bar{R}_t(\gamma, a) + c_t(\gamma, a) \tag{3.2}$$

$$\begin{aligned} \bar{R}_t(\gamma, a) &= \frac{1}{N_t(\gamma, a)} \sum_{m=1}^{t} \gamma^{t-m} \vec{R}_m(a) \mathbb{1}_{\{I_m = a\}} \\ N_t(\gamma, a) &= \sum_{m=1}^{t} \gamma^{t-m} \mathbb{1}_{\{I_m = a\}} \end{aligned} \tag{3.3}$$

$$c_t(\gamma, a) = 2B \sqrt{\frac{2 \ln \left(n_t(\gamma) \sqrt[4]{|O||\mathcal{A}^*|} \right)}{N_t(\gamma, a)}}, \quad n_t(\gamma) = \sum_{i=1}^{|K|} N_t(\gamma, i) \tag{3.4}$$

Like in PUCB1, in each episode, the Pareto set $\mathcal{A}'$ is found, such that $\forall \ell \notin \mathcal{A}'$, there exists an action $a \in \mathcal{A}'$ that dominates ℓ:

$$\bar{R}_t(\gamma, \ell) + c_t(\gamma, \ell) \not\succ \bar{R}_t(\gamma, a) + c_t(\gamma, a) \tag{3.5}$$

Finally, an action is randomly selected from $\mathcal{A}'$.

In Eq. 3.2, the exploitation part of the algorithm is defined by the discounted average reward $\bar{R}$ from Eq. 3.3. The exploration is determined by the discounted padding function c_t from Eq. 3.4. The denominator $N_t(\gamma, a)$ represents the discounted number of times action a was selected. At each episode in which action a is not selected, $N_t(\gamma, a)$ decreases, causing the value of the padding function c_t to increase. This increment makes action a more likely to be chosen, as the algorithm aims to maximize u_b.

DPUCB is presented in Algorithm 1. Note that, in this case, the parameter w is not required. For each agent, the initialization phase (lines 5 and 6) guarantees that each action is selected once in random order. After this phase, the Pareto set $\mathcal{A}'$ is computed according to Eq. 3.5 by finding those actions whose associated upper bound (Eq. 3.2) is not Pareto-dominated by another action (line 8). An action is selected randomly from $\mathcal{A}'$ at line 9. The agent's average reward vector $\bar{R}(\gamma, a)$ is updated using Eq. 3.3 (line 11).

Algorithm 1 Discounted + Sliding window Pareto UCB

1: **procedure** DISCOUNTEDSLIDINGWINDOWPARETOUCB($\mathcal{G}$, $|K|, \gamma, w, n_e$) ▷ $\mathcal{G}$ is the set of agents; $|K|$, the number of actions; γ, the discount factor; w, the window size and n_e, the number of episodes
2: $t \leftarrow 1$
3: **while** $t \leq n_e$ **do**
4: **for** each agent $g \in \mathcal{G}$ **do**
5: **if** $t < |K|$ **then**
6: Select randomly an action that has not been taken yet
7: **else**
8: Find the Pareto set $\mathcal{A}'$ as stated in Eq. 3.5 if not using w; otherwise, use Eq. 3.9.
9: Select action a uniformly at random from $\mathcal{A}'$
10: **end if**
11: Update $\bar{R}_t$ according to Eq. 3.3 if not using w; otherwise, use Eq. 3.7.
12: **end for**
13: $t \leftarrow t + 1$
14: **end while**
15: **end procedure**

Afterwards, the current episode is updated at line 13, and the previous steps repeat until n_e episodes have passed.

In the case of SWPUCB, some modifications must be made to consider just the most recent plays. As aforementioned, SWUCB also tackles non-stationarity by averaging past rewards considering only the last w plays (episodes). This is done by making Eq. 3.3 and Eq. 3.4 require w and by setting $\gamma = 1$ and $m = t - w + 1$ in them. Changes are needed in some terms of those two equations: now we denote the average reward vector and the padding function of action a by $\bar{R}_t(\gamma, w, a)$ and $c_t(\gamma, w, a)$, respectively. The upper bound of SWPUCB is defined by Eq. 3.6.

$$u_b = \bar{R}_t(\gamma, w, a) + c_t(\gamma, w, a) \tag{3.6}$$

The average reward vector $\bar{R}_t(\gamma, w, a)$ and the padding function $c_t(\gamma, w, a)$ are calculated using Eq. 3.7 and 3.8, respectively.

$$\begin{aligned} \bar{R}_t(\gamma, w, a) &= \frac{1}{N_t(\gamma, w, a)} \sum_{m=t-w+1}^{t} \gamma^{t-m} \vec{R}_m(a) \mathbb{1}_{\{I_m = a\}} \\ N_t(\gamma, w, a) &= \sum_{m=t-w+1}^{t} \gamma^{t-m} \mathbb{1}_{\{I_m = a\}} \end{aligned} \tag{3.7}$$

$$c_t(\gamma, w, a) = B\sqrt{\frac{2 \ln\left(n_t(\gamma, w)\sqrt[4]{|O||\mathcal{A}^*|}\right)}{N_t(\gamma, w, a)}}, \quad n_t(\gamma, w) = \sum_{i=1}^{|K|} N_t(\gamma, w, i) \tag{3.8}$$

These changes are extended to Eq. 3.5, which is now represented by Eq. 3.9.

$$\bar{R}_t(\gamma, w, \ell) + c_t(\gamma, w, \ell) \not\succ \bar{R}_t(\gamma, w, a) + c_t(\gamma, w, a) \tag{3.9}$$

SWPUCB is also presented in Algorithm 1, essentially following the same steps of DPUCB, though it requires an additional parameter w for line 8 and line 11. The Pareto set $\mathcal{A}'$ is found at line 8 according to Eq. 3.9, and the average reward vector $\bar{R}_t(\gamma, w, a)$ is updated according to Eq. 3.7 at line 11.

It is worth emphasizing that the original SWUCB algorithm does not employ a discount factor γ. However, the proposed formulation does consider such a discount factor, in order to average past rewards using the last w plays while also giving more weight to recent observations. If one wants to use SWPUCB without discount, it suffices to set $\gamma = 1$.

3.2 Modified Pareto Q-learning (mPQL)

The original PQL algorithm (Section 2.3.2.2) was modified to deal with non-stationary environments and with the fact that it is being applied to single-state problems. This modified PQL is denoted by mPQL.

Note that Eq. 2.4 from PQL implies a learning rate α equal to 1. For mPQL to work in non-stationary environments, α must set to values that are less than 1, to enable using prior knowledge.

Recall that, in PQL, updating $\tilde{Q}_{set}(s, a)$ involves calculating $ND_t(s, a)$, which is the set of non-dominated vectors in the next state of s that is reached through action a at time step t. However, the model used in this work implies a single-state. Consequently, there is not next state and it is not possible to calculate the set of non-dominating vectors $ND_t(s, a)$. This allows for a simplified procedure to update the Q-sets of each action. In fact, Q-sets are now replaced by Q-vectors denoted by $\vec{Q}$, as PQL works with a set of vectors for each state-action pair, but mPQL works with just a vector for each action. The new update rule is the same as that of QL, but the rewards and Q-values are vectors

Algorithm 2 Modified Pareto Q-learning

1: **procedure** MODIFIEDPARETOQLEARNING($\mathcal{G}$, α, n_e) ▷ $\mathcal{G}$ is the set of agents; α, the learning rate and n_e, the number of episodes
2: $t \leftarrow 1$
3: **while** $t \leq n_e$ **do**
4: **for** each agent $g \in \mathcal{G}$ **do**
5: **if** t=1 **then**
6: Initialize $\vec{Q}$'s randomly
7: **else**
8: Find the Pareto set Q' based on the $\vec{Q}$'s
9: Select action a uniformly at random from Q'
10: Take action a and observe reward vector $\vec{R}$
11: Update $\vec{Q}(a)$ according to Equation (3.10)
12: **end if**
13: **end for**
14: $t \leftarrow t + 1$
15: **end while**
16: **end procedure**

instead of scalars, as shown in Eq. 3.10, where $\vec{R}$ is the observed reward vector.

$$\vec{Q}(a) = \alpha\vec{R} + (1 - \alpha)\vec{Q}(a) \tag{3.10}$$

The steps of mPQL are outlined in Algorithm 2. The Q-vector $\vec{Q}$ of each action is initialized randomly at line 6. In each episode, the Pareto set Q' is found based on the $\vec{Q}$'s at line 8. An action from that set is selected randomly (line 9). This random way of choosing is inherited from PQL to treat every non-dominated solution equally. To balance exploration and exploitation in the action selection, the decaying ε-greedy method can be used. After the action is executed, an agent receives a reward vector $\vec{R}$ (line 10), which is used to update the $\vec{Q}$ of the chosen action according to a vector-oriented extension of the general updating rule of QL (line 11), represented by Eq. 3.10.

3.3 Summary

In this chapter, three algorithms were introduced: DPUCB, SWPUCB and mPQL. All of them deal with both non-stationarity and multiple objectives.

DPUCB and SWPUCB share the multi-objective approach of PUCB1 and use certain elements from DUCB and SWUCB, respectively, to address non-stationarity. SWPUCB can use a discount factor too, though.

mPQL is based on PQL. While PQL is already a MORL algorithm, modifications were needed to apply it to a stateless setting with non-stationarity.

4 EXPERIMENTAL RESULTS

This chapter presents the results from experiments realized to evaluate the algorithms proposed in Chapter 3. Section 4.1 details the settings and scenario used in the experiments. Section 4.2 presents the methods used in the experiments, as well as explanations of their results. A general discussion of the results is presented in Section 4.3. Other metrics for comparison are discussed in Section 4.4.

4.1 Scenario and settings

Several works were consulted that deal with bi or multi-objective TAP. However, many of the scenarios used in those works were not suitable to compare with due to some issues, such as lack of information (some details about the networks were not included), size (they were extremely small or large) and use of algorithms that require additional information (the algorithms proposed in Chapter 3 work does not require this), among other issues. Moreover, the main objective of this work is to deal with non-stationarity and multiple objectives in a MAS, rather than comparing the proposed algorithms to others.

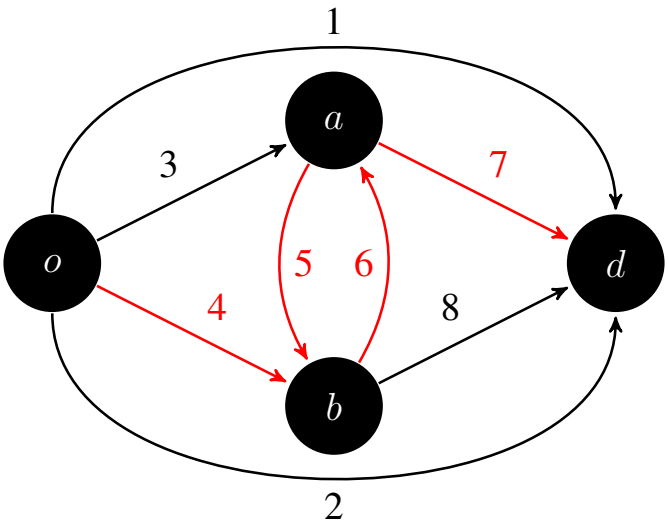
In what follows, the scenario used is a 4-node network, formulated in (WANG; EHRGOTT, 2013) and depicted in Fig. 4.1, for which the solution is known. The demand from vertex o to d is 10000 vehicles per hour, and there are six possible routes or paths from o to d (the authors in (WANG; EHRGOTT, 2013) excluded paths such as 3-5-6-7). It must be noted that all six routes are efficient when there is no flow. The cost functions associated with each link follow the BPR family of cost functions, shown in Eq. 4.1, where $a = 0.15$ and $b = 4$. Free flow travel time, or FFTT, and tolls of the routes are given in Table 4.1.

Recall that the travel time of a route is calculated as the sum of the travel times of its links, and its toll is calculated analogously.

$$T_l(f_l) = T_l^0(1 + a\,(f_l/C_l)^b) \tag{4.1}$$

As a metric to compare the approaches, a table that reports the values for each objective is shown. All plots and tables shown ahead report mean values, as well as standard deviations, calculated over 30 repetitions of the same setting (except if the method is deterministic). All experiments were carried out on a PC with a processor

Figure 4.1 – 4-node network. Red links are toll-free.



Source: Adapted from (WANG; EHRGOTT, 2013).

Table 4.1 – Routes characteristics of the 4-node network.

Route	*Links*	*FFTT*	*Toll*
1	1	18.0	20
2	2	22.5	15
3	3–7	36	1
4	4–8	36	1
5	3–5–8	26.4	2
6	4–6–7	54	0

Source: Adapted from (WANG; EHRGOTT, 2013).

Intel Core i7-8700 3.20 GHz and 31 Gb of RAM under Ubuntu 18.04 operating system. The algorithms were implemented using Python 3.7.

Recall that the action set A of each agent is formed by a set of $|K|$ routes. Also, the reward vector $\vec{R}$, calculated by Eq. 3.1, is the negative travel time and toll of the route chosen by the agent, as the standard practice of maximizing rewards is followed.

4.2 Methods used in experiments

Those algorithms presented in Chapter 3 are compared with the assignment yielded by two centralized methods: EQS assignment and the time surplus maximization (Section 2.4.2), as well as PUCB1. Table 4.2 shows the average travel time and average toll reached by each algorithm. This information is used mainly to illustrate the different solutions each algorithm provide and how each algorithm works.

Table 4.2 – 4-node network. Average travel time and toll, for each algorithm.

Algorithm	*Avg. toll*	*Avg. travel time*
EQS	6.50	74.69
Time surplus max.	12.54	32.48
PUCB1	6.25 ± 0.01	94.31 ± 0.55
DPUCB ($\gamma = 0.77$)	9.72 ± 0.02	35.94 ± 0.11
SWPUCB ($w = 12, \gamma = 1$)	8.52 ± 0.10	48.06 ± 1.04
SWPUCB ($w = 12, \gamma = 0.77$)	8.44 ± 0.04	45.44 ± 0.54
mPQL ($\alpha = 0.9, decay = 0.997$)	7.60 ± 0.01	56.97 ± 0.10

Source: Author.

4.2.1 Centralized methods

The EQS assignment was implemented following (WANG; RAITH; EHRGOTT, 2010), as these values are not reported in (WANG; EHRGOTT, 2013) for this particular network. The EQS assignment iteratively divides the flow equally among all efficient routes. Obviously, it is not expected for EQS to have a good performance, but it serves as a baseline.

It must be noted that the EQS assignment does not perform well in this network. When there is no flow, all routes are efficient. Therefore, the same flow is assigned to all of them at the beginning. Given that there are 10000 agents and 6 routes, each route has $\sim$ 1666.67 agents. As toll is flow-independent, the efficiency in following episodes is determined by travel time. However, after calculating the new travel times, all routes remain efficient. Then, the flow same flow (~ 1666.67) is allocated in each route again. The process of calculating travel time and allocating flow in the efficient routes repeats in each episode, so this allocation does not change over time and the six routes remain with the same number of agents.

The EQS assignment ends up performing an assignment that has average toll and travel time as shown in the first line of Table 4.2.

Particularly, for this network, there is an **alternative approach** of the EQS assignment that produces another reasonable result, instead of the practical result explained in previous paragraphs. Taking into account that routes 3 and 4 are equivalent (see Table 4.1), they could be represented as just one route. This way, in theory, there would be 5 routes instead of 6. These 5 routes are efficient when there is no flow, thus the total flow would be divided equally between them. Table 4.3 shows this new distribution of flow. The flow assigned to the route that represents routes 3 and 4 would be divided

equally between those two, meaning that routes 3 and 4 would have 1000 agents each.

Table 4.3 – Alternative flow distribution of EQS.

Routes	Route 1	Route 2	Route 3+Route 4	Route 5	Route 6
Flow	2000	2000	2000	2000	2000

Source: Author.

After calculating the new travel time, these 5 routes continue being efficient. Therefore, the flow distribution would not change in subsequent episodes.

This alternative approach of EQS would lead to a solution with an average travel time of 56.88 and an average toll of 7.60. These values are included to later show the relation they share with the values obtained by mPQL.

However, it must be noted that, if all the routes in the network were different (if there were no equivalent routes), the practical and the alternative results of the EQS assignment would be equal.

The second line of Table 4.2 also shows the results reported in (WANG; EHRGOTT, 2013), where the authors use their proposed method (time surplus maximization). This method, by design, produces low travel time since it considers extra information, namely, how travelers value their time. This, however, causes the toll to be the highest. It must be noted that time surplus maximization requires extra information that neither of the other algorithms demands. While it may be not fair to compare it to the other methods, the results of time surplus maximization are included because it is the known solution presented in (WANG; EHRGOTT, 2013) for this network.

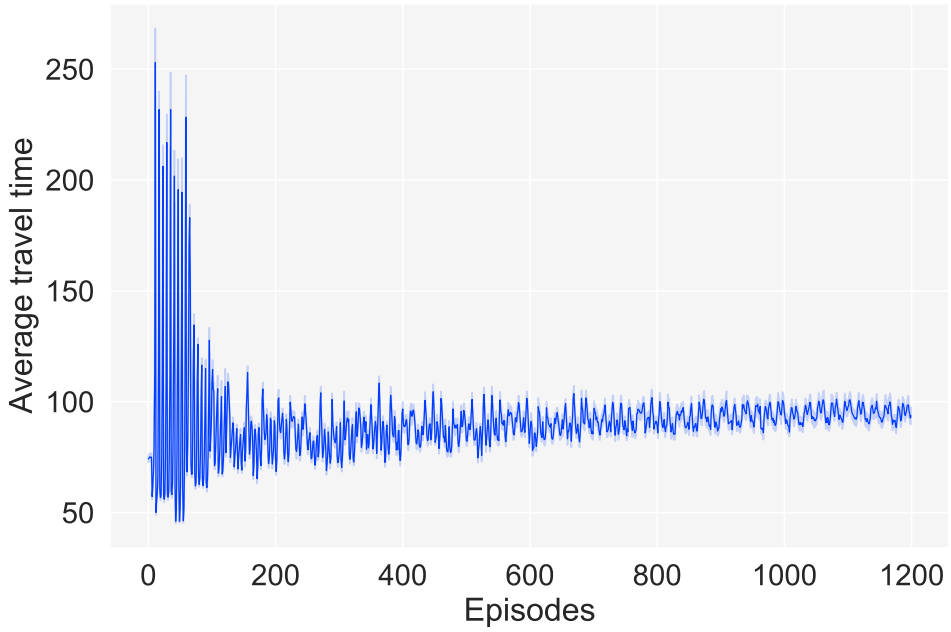
4.2.2 PUCB1

This algorithm does not require parameters to be tuned, but rewards need to be normalized, if one wishes to comply with the proof regarding the upper-confidence bound of PUCB1 given in (DRUGAN; NOWÉ, 2013), which is based on a support of the rewards in $[0, 1]^{|O|}$, where O is the objective set.

For travel time, the min and max values used for normalization are the travel time of the fastest route with a single agent and the travel time of the slowest route when all agents are assigned to it, respectively. For toll, those min and max values are the minimum and maximum toll of the six routes. PUCB1 shares the initialization phase of DPUCB.

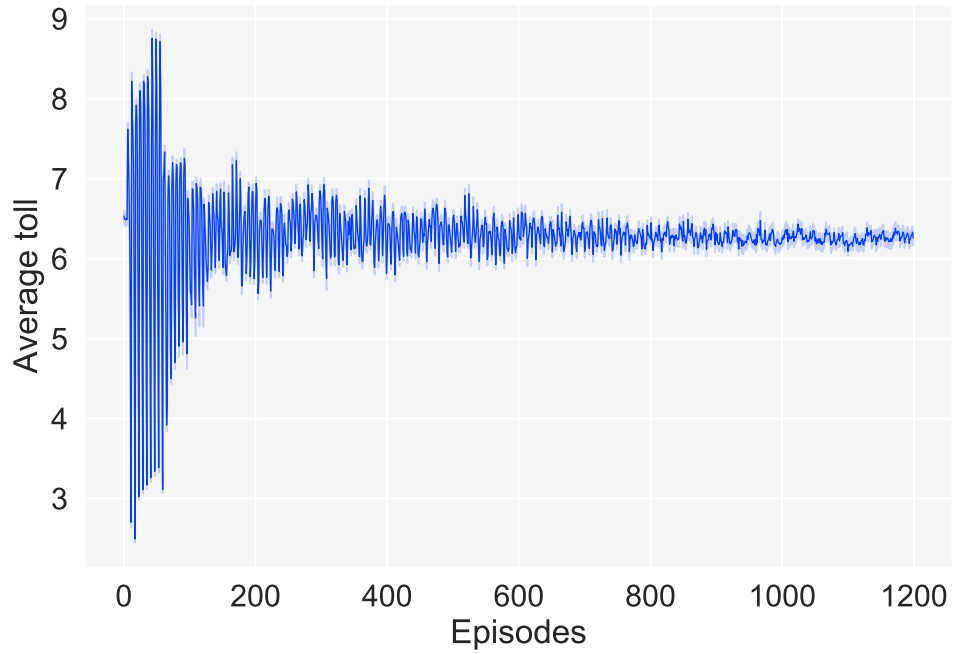
Figures 4.2 and 4.3 show the average travel time and average toll of PUCB1, respectively.

Figure 4.2 – Average travel time of PUCB1.



Source: Author.

Figure 4.3 – Average toll of PUCB1.



Source: Author.

While normalization certainly plays a role in the performance of PUCB1, the second – and most important – issue regarding this algorithm is the fact that PUCB1

does not fully deal with non-stationarity. Like UCB1, this algorithm assumes the reward vectors of the routes are identically distributed. This means that if an agent chooses route k at episode t and then chooses it again at episode $t + 1$, the reward vectors received at both episodes should be drawn from the same probability distribution. That is the case of toll, but travel time depends on the decision of the other agents as well and its distribution may vary across time.

As a further note, it must be remarked that, for the sake of tests, PUCB1 was also run without normalization. This yielded an average toll of 6.69 and an average travel time of 71.14. The results produced less oscillations and the repetitions were more varied (i.e., there was a higher standard deviation), unlike PUCB1 with normalization. As it can be observed in figures 4.2 and 4.3, the repetitions of PUCB1 with normalization are very similar in average and there is a low standard deviation that is barely perceived.

Note also that PUCB1 takes the most time among all algorithms of the UCB family; moreover, it yields the most unbalanced result, as it achieves the highest average travel time and the lowest average toll. This is due to PUCB1 converging to values that do not deviate too much from the initial sampling of average travel time and average toll due to normalization. The min and max values used to normalize travel time provide a too wide interval, mapping the normalized travel time rewards to a very narrow interval. Consequently, the differences between the normalized rewards are too low, and the algorithm relies more on the normalized toll values, which are less affected by such a mapping/normalization.

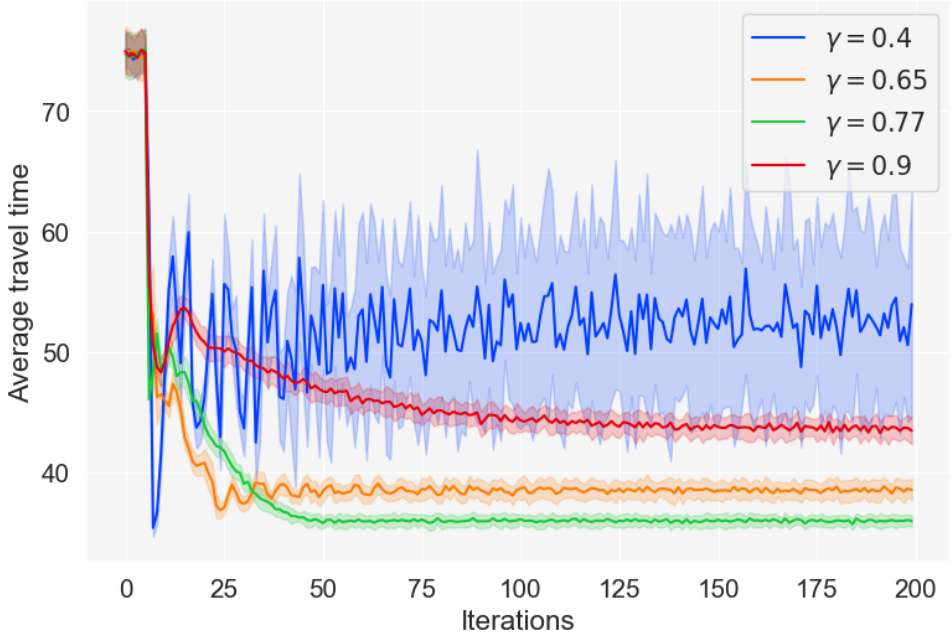
In short, the PUCB1 yielded the worst performance, thus stressing the point that a novel approach was needed. Next, it is discussed the performance of the algorithms proposed in this work, which were detailed in Chapter 3, namely DPUCB, SWPUCB and mPQL.

4.2.3 DPUCB

As just mentioned, normalization proved to be more of a hindrance than a help when it was used in PUCB1. For that reason, DPUCB and SWPUCB do not include it.

For the experiments, the discount factor used was $\gamma = 0.77$. This value was obtained after extensive tests. Figure 4.4 shows the average travel time when different values of γ are used. It must be noted that it seems that when γ is too low, there are great oscillations in the average travel time and average toll. This is caused due to a frequent

Figure 4.4 – Average travel time of DPUCB using different values of γ.



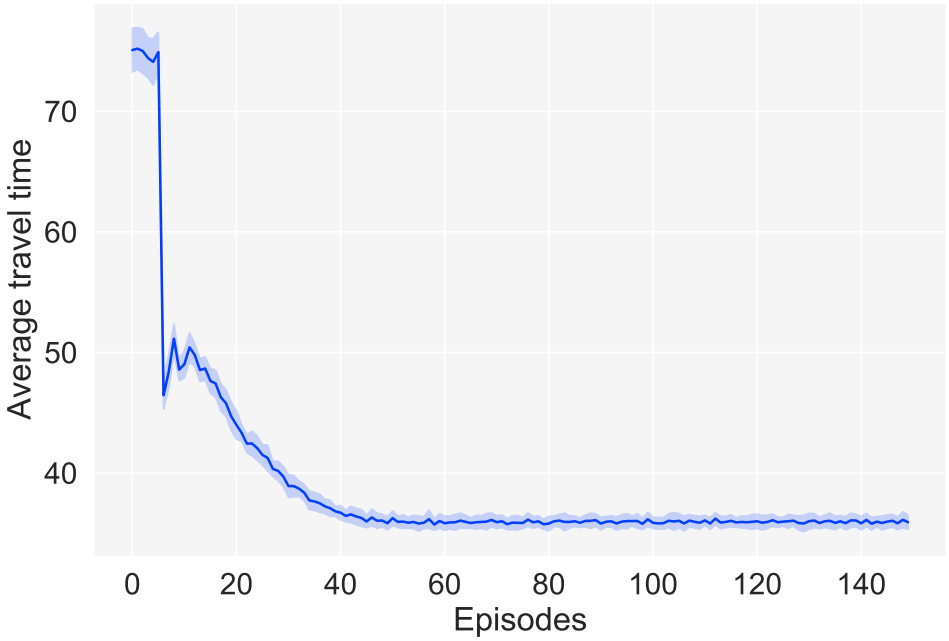
Source: Author.

change of the set (and thus, the number) of efficient routes. A low γ increases quickly the exploration term (Eq. 3.4), which means that a route that has not been selected as much as the others, eventually will become the most, or one of the most, attractive. Therefore, when there are just one or two of such attractive routes, traffic congestion occurs, while a higher number of efficient routes produces the opposite effect. When γ is high, e.g., $\gamma = 0.9$, the exploration term increases slowly and also slows down convergence. As such, initial rewards have a weight close to that of more recent ones. Therefore, most of the routes are efficient, as it was the case in the initial episodes.

A compromise is achieved with $\gamma = 0.77$. It ensures the fastest routes 1, 2 and 5 have more probability to be chosen in the first episodes after initialization. Hence the decrease in the average travel time, while the average toll increases as these routes have the highest tolls. Figures 4.5 and 4.6 show the average travel time and average toll of DPUCB when $\gamma = 0.77$, respectively.

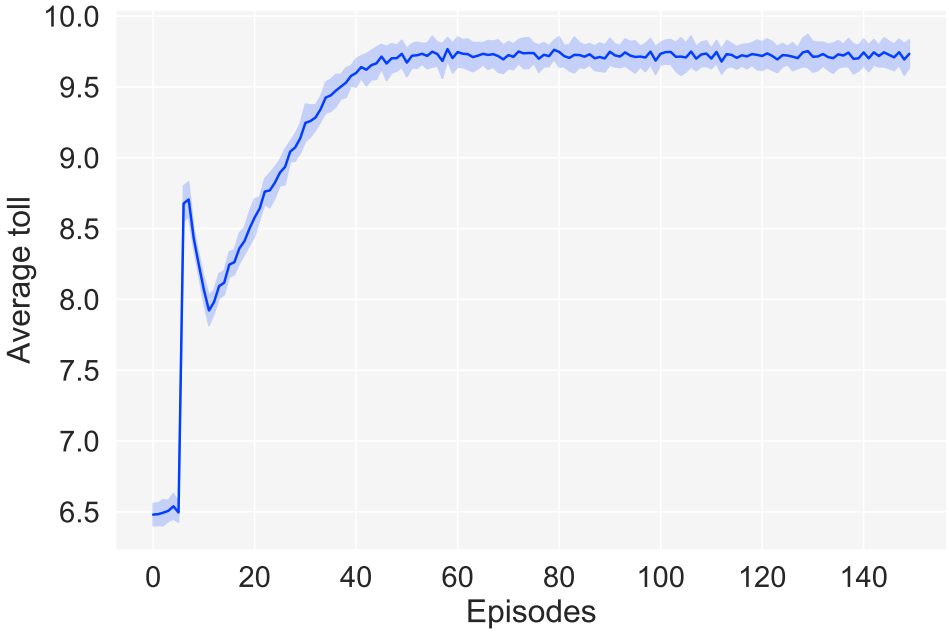
Note that the padding function from Eq. 3.4, that determines exploration, will produce low values as it depends on the function ln. As non-normalized rewards are used, travel time is not affected as much as toll since it is in a higher scale. Particularly, routes 3, 4, 5 and 6 are more affected since their tolls are very low when compared to those of routes 1 and 2. This leads to travel time having more influence on the agents' decisions. As such, routes 1 and 2, which share no links with other routes, are the most preferred routes as they are the fastest ones. The remaining routes maintain an order according to

Figure 4.5 – Average travel time of DPUCB when $\gamma = 0.77$.



Source: Author.

Figure 4.6 – Average toll of DPUCB when $\gamma = 0.77$.



Source: Author.

their FFTTs: route 5 is the third most chosen, while routes 3 and 4 have the same flow (as they have the same toll and FFTT). Finally, route 6 is the slowest and least preferred one.

4.2.4 SWPUCB

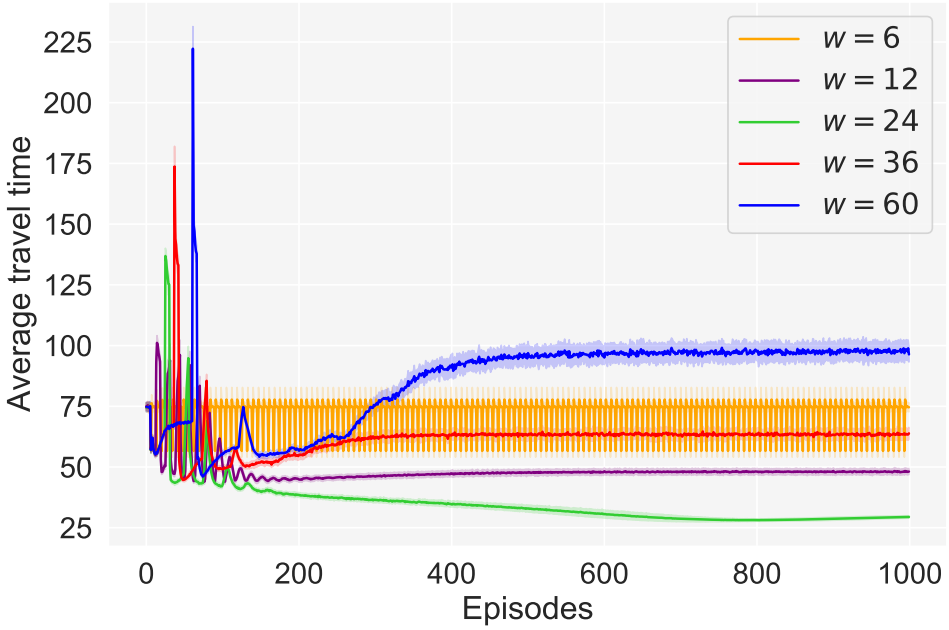
SWPUCB requires a further parameter, when compared to DPUCB, namely the window size w. Therefore, γ was kept as 1 and experiments changing the value of w were made. Given that this network has six routes, values of w that are multiples of six were tested. Figures 4.7 and 4.8 shows a comparison of average travel time and average toll, respectively, using different values of w. Note that there is a period (roughly between episodes 0 and 100), in which almost all cases show that exploration is more intense, thus having visible oscillations. After those, most curves start to converge or stabilize.

An exception is the case when $w = 6$, which shows oscillations from beginning to end. The window is too short to allow agents to learn something. It is observed that other values of w produce varied effects, such as increasing the average toll and decreasing the average travel time. When $w = 12$ the average toll also increases while the average travel time decreases, as agents begin to select faster routes more frequently, with the consequent increase in tolls. A higher value ($w = 24$) produces the same effect on a higher scale, affecting mainly the average toll as it grows considerably. Note also that the deviations are higher (see green curve). Finally, when the window size is large (36 and 60), the opposite effect occurs and the average travel time increases, depending on how high the value of w is ($w = 60$ leads to a higher average travel time than $w = 36$), with decrease in the toll.

As using $w = 12$ Pareto-dominates most of the other cases, i.e., has a lower average travel time and average toll, while not incurring such a high increase in average toll as $w = 24$ does, $w = 12$ was selected for the experiments regarding SWPUCB.

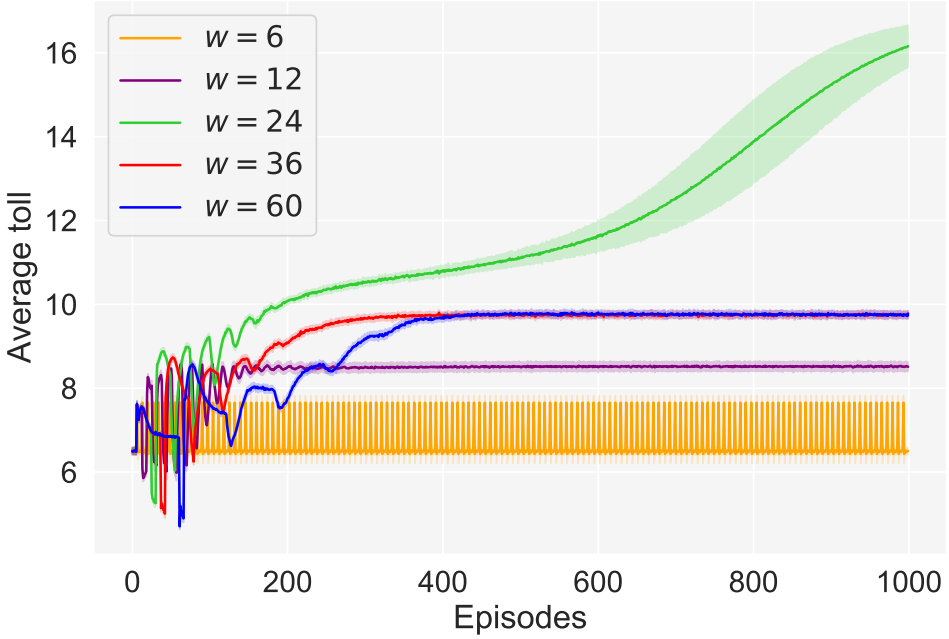
Given this selection, the next step is to analyze how SWPUCB with $\gamma = 1$ and $w = 12$, compares to the other algorithms, not only in terms of the final values shown on Table 4.2, but also in terms of oscillations and how these value change along episodes. When $w = 12$, the agents have a short window to acquire experience. This affects their decisions and produces initial oscillations, because the number of times a route has been chosen inside that window frame may be low, taking into account there are six possibly efficient routes to choose from. After about 200 episodes, the agents have chosen each route enough times to prefer one.

Figure 4.7 – Average travel time with non-discounted SWPUCB and different values of w.



Source: Author.

Figure 4.8 – Average toll with non-discounted SWPUCB and different values of w.



Source: Author

In short, while using $w = 12$ produces oscillations in the initial episodes, it was selected because it Pareto-dominates the most of the window sizes.

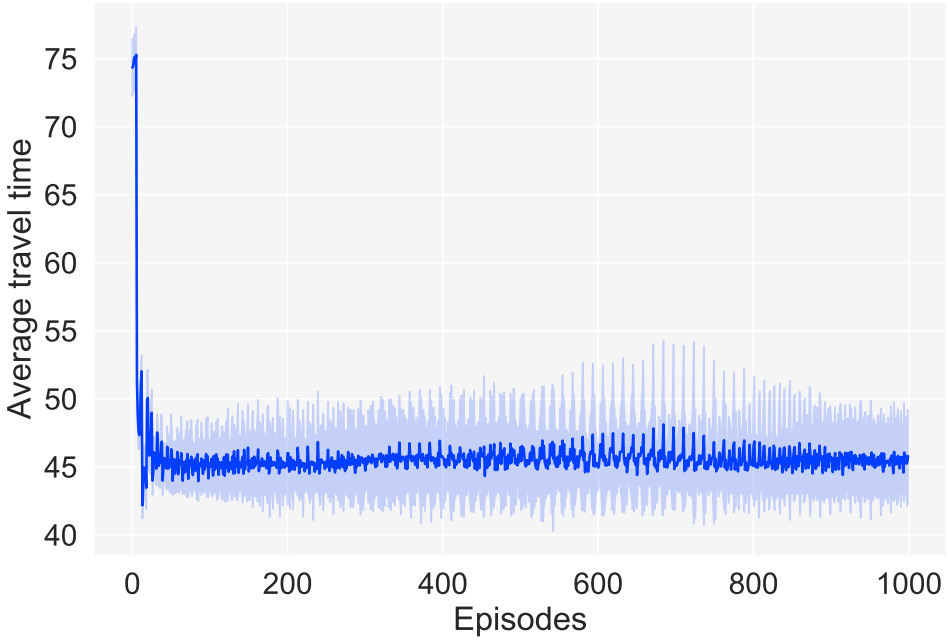
Next, the value of the discount γ was also varied. In the discounted SWPUCB, γ plays an important role, especially when w is high. Depending on the value of γ, the window size may become less relevant, i.e., the higher the w, the less relevant this window becomes. This happens because least recent rewards (inside the window) may have almost zero weight if γ is low. For comparison purposes, the value of γ used in DPUCB was used (0.77), and the window size w remained as 12. Figures 4.9 and 4.10 show the average travel time and average toll of SWPUCB with these values, respectively. Using $\gamma < 1$ accelerates the convergence when compared to the non-discounted SWPUCB. If w were higher, more rewards could be considered and the discounted SWPUCB could approximate more the results of DPUCB and reduce the oscillations, as they are caused because the window is small enough to leave out some information that causes the agent to switch routes. Even so, note that discounting rewards helps, as SWPUCB with discount achieves a lower average travel time and average toll than non-discounted SWPUCB, shown in Table 4.2.

4.2.5 mPQL

mPQL was tested using different values of α (0.1, 0.3, 0.5, 0.7 and 0.9), while using the decaying ϵ-greedy method to balance exploitation and exploration. The decay was set to promote exploration for about 70% of the episodes. This ensures that there is enough exploration to avoid biased results due to the initialization phase.

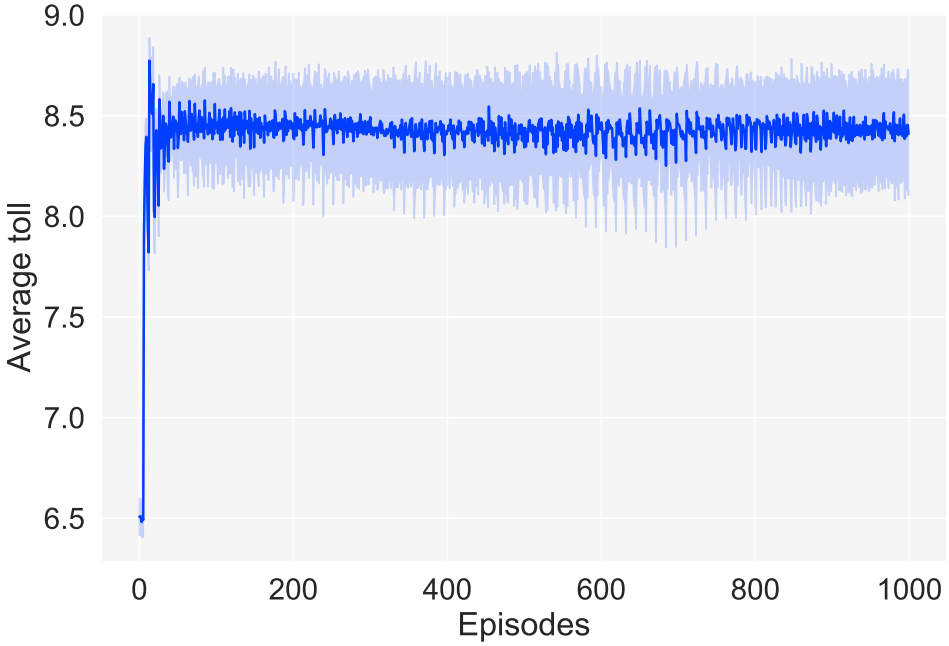
However, different values of α tended to reach very similar solutions after converging. Figures 4.11 and 4.12 prove this point by showing a comparison of average travel time and average toll with different values of α. All curves converge to very similar values, though they differ during the learning process. The main difference lies in the number of episodes required to stabilize or smooth these curves. Higher values of α reduce that number. Because of this, in Table 4.2 it is presented the average travel time and average toll of mPQL when $\alpha = 0.9$ and the decay is 0.997. A higher value of α means that the agents are learning faster and acquiring knowledge that leads to a flow distribution that, though changes over time, does not experience large variations in average travel time and average toll as, for example, the cases when $\alpha = 0.1$ or $\alpha = 0.3$. When α is low, the agents take more time to learn about their environment and can also

Figure 4.9 – Average travel time of discounted SWPUCB when $w = 12$ and $\gamma = 0.77$.



Source: Author.

Figure 4.10 – Average toll of discounted SWPUCB when $w = 12$ and $\gamma = 0.77$.



Source: Author.

be biased due to initialization, thus switching routes yields less stable average travel time and average toll. However, as observed, all values of α converge to similar values.

The solutions obtained by mPQL with different values of α are approximately the same as the result of the **alternative approach** of the EQS assignment that has an average travel time of 56.88 and an average toll of 7.60 [1]. These values are very similar to those of mPQL, displayed in the last line of Table 4.2. This happens because of the random uniform selection, which promotes an approximately equal number of agents in each efficient route (e.g., the flow distribution in the first episode when $\alpha = 0.9$ is [1675, 1667, 1660, 1666, 1669, 1663]), and the randomness in exploration. While the random uniform selection emulates, to some degree, an equal share assignment, exploration lets the agent acquire knowledge about the routes. As exploration decays, the agents come to know that there are routes that are equivalent (specifically, routes 3 and 4). While these routes are efficient, they are equally attractive, so half the agents selects route 3 and the other half selects route 4. This results in a flow distribution similar to that of the alternative approach of the EQS assignment.

4.3 Discussion

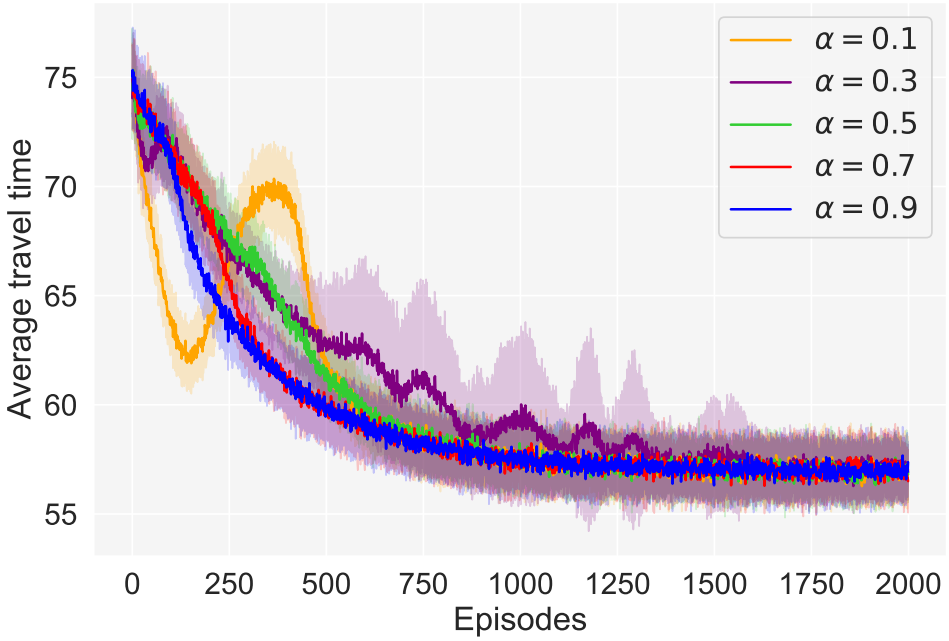
Experiments show that most of the decentralized algorithms provide alternative solutions to those produced by centralized algorithms like Equal share assignment (EQS) and Surplus maximization. Such solutions fall under multi-objective user equilibrium (MUE) conditions (RAITH et al., 2014), where no individual agent can improve at least one of their objectives without worsening any of the others by unilaterally switching routes.

Pareto UCB1, applied to a multi-agent scenario, does not deal well with non-stationarity due to giving the same importance to previous and current rewards.

DPUCB and non-discounted SWPUCB rely on a discount factor γ and a window size w, respectively. Different values for them may favor one objective over the other. When SWPUCB works with discount, it has both parameters to tune. w has a less relevant role when it is increased, as γ can downplay the weights of the least recent rewards inside the window.

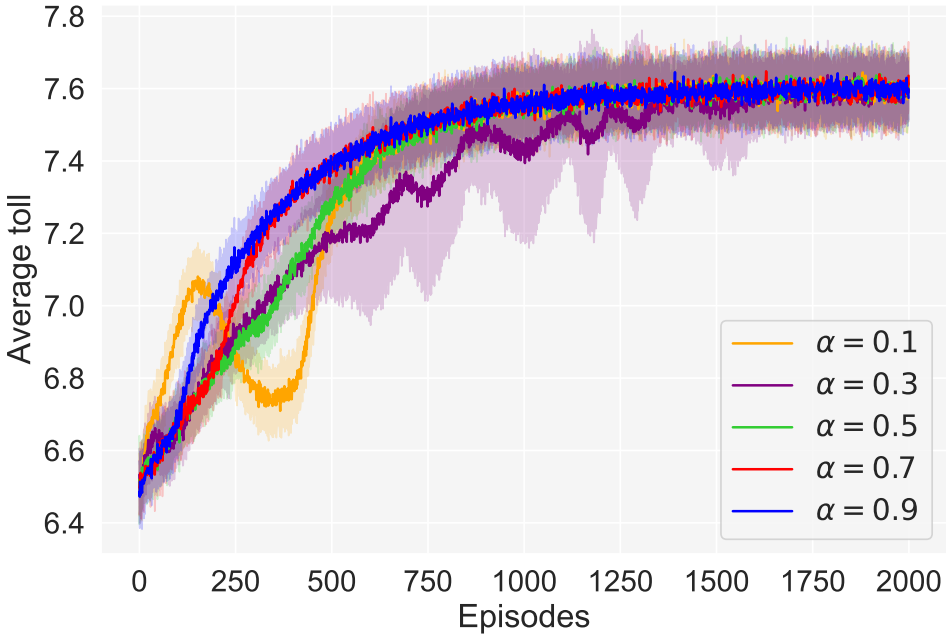
[1] Note that these values are not the same that are displayed in Table 4.2. Table 4.2 shows the values of the original EQS assignment, while here the comparison is made between mPQL and the alternative approach of the EQS assignment explained in Section 4.2.1.

Figure 4.11 – Average travel time of mPQL using different values of α and $decay = 0.997$.



Source: Author.

Figure 4.12 – Average toll of mPQL using different values of α and $decay = 0.997$.



Source: Author.

The modified Pareto Q-learning (mPQL) faces difficulties as the value of α does not significantly influence its final result, which tends to be the same as that of the EQS assignment. There are only small variations in average travel time and average toll produced by different values of α. It must be noted that the result of an alternative approach of the EQS assignment was used to compare the results of mPQL. However, this is a particular case due to some features of the network used. In general, mPQL should tend to reach a similar solution to that of the EQS assignment.

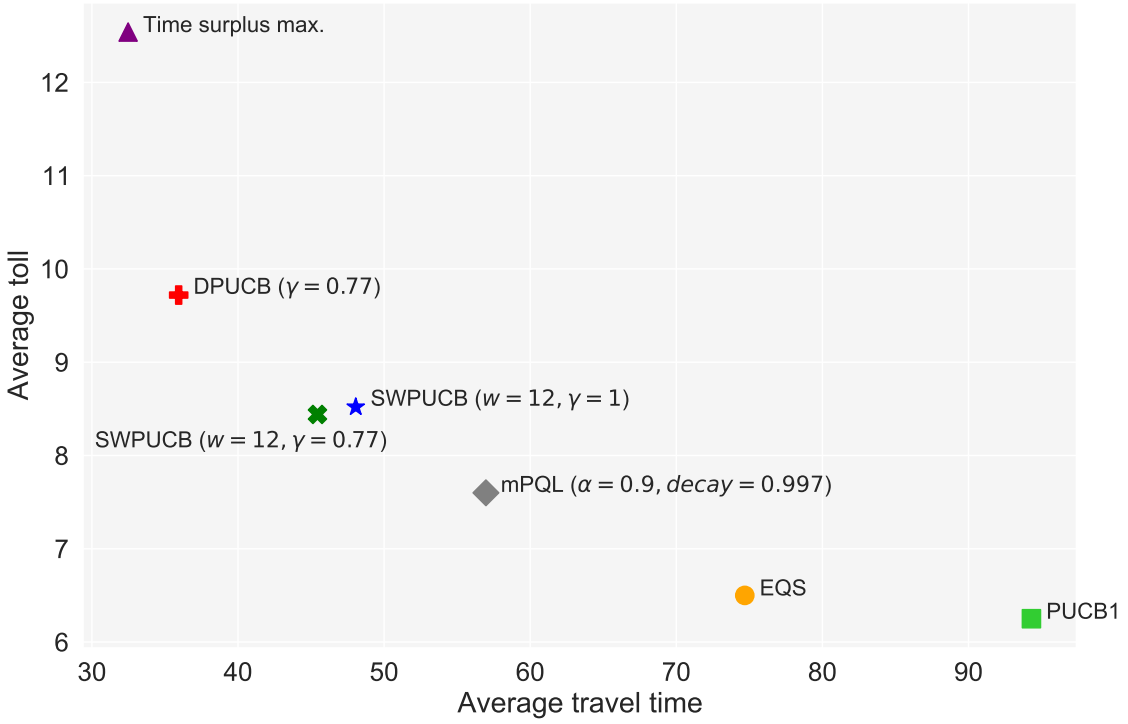
Unlike DPUCB and SWPUCB, variations of the parameters of mPQL does not produce notably different flow distributions. As such, the learning rate does not influence the final solution. This happens because of the random uniform selection and exploration.

While the UCB-based algorithms use the same rule to select randomly a route between those that are efficient, their padding function and the discount factor (and the window size to a lesser extent, as explained before) provide a variety of different solutions. The padding function encourages choosing less explored routes, while the exploration in mPQL does not give preference to specific routes. The discount factor in the UCB-based algorithms helps to deal with the non-stationarity, as giving more importance to recent rewards lets the agents adapt to the changing environment. For example, Figures 4.7 and 4.8 show how different average toll and average travel time can be obtained by varying the window size w parameter of the non-discounted SWPUCB. Analogously, using other values for other parameters of the UCB-based algorithms can result in other solutions. Also, DPUCB and SWPUCB tend to converge more quickly when there is discount involved.

Overall, DPUCB and SWPUCB offer more flexibility to find different alternative solutions, but DPUCB is more advantageous. Meanwhile, mPQL tends to find solutions that could be reached easier by using a centralized method (EQS).

Finally, Figure 4.13 shows the Pareto front formed by the results presented in Table 4.2. Every result reached by the algorithms, except non-discounted SWPUCB, is an efficient solution. Still, as it was pointed before, the UCB-based algorithms can produce different results when their parameters vary (for an example, see Figure 4.14) and another combination of values could yield an efficient solution for non-discounted SWPUCB. This enables a better coverage of the Pareto front of efficient solutions. However, recall that mPQL can reach different solutions, but all of them tend to be very similar, so it would cover a very small area of the Pareto front. On the other hand, results of centralized algorithms, such as EQS, will not change as they are deterministic. Though time surplus

Figure 4.13 – Average travel time and average toll of the algorithms presented in Table 4.2.



Source: Author.

maximization may yield another result if the additional information it uses changes, that is beyond the scope in this work, as the algorithms proposed do not require such information.

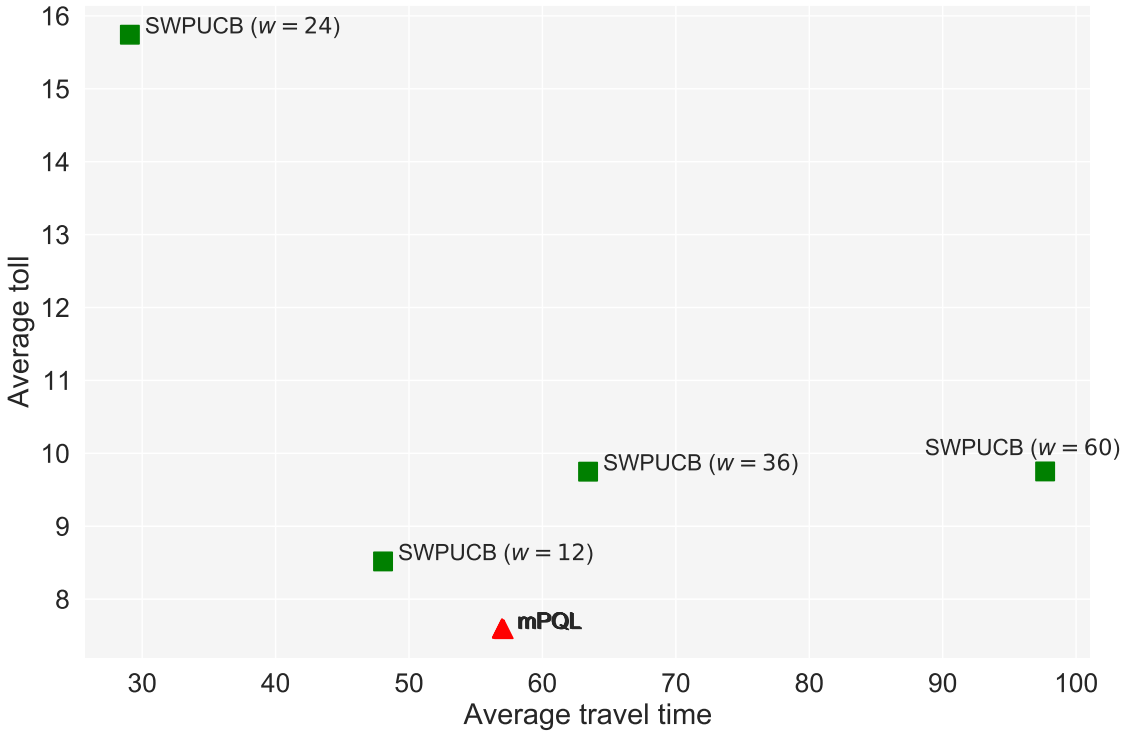
4.4 Other metrics for comparison

Since the problem is multi-objective, a comparison such as that presented in Table 4.2 and illustrated in Figure 4.13 only shows a dimension of the algorithms' results based on the average travel time and average toll. There are other criteria that can be useful for comparison purposes. In this section, some examples of these criteria will be presented.

4.4.1 Coverage of the Pareto front

Coverage of the Pareto front demonstrates how useful an algorithm can be by providing varied solutions to the user. As the algorithms proposed do not use scalarization nor require additional information, it is not possible to prioritize one objective over the other. As stated in Section 2.2, a set of solutions should be calculated so the agent can

Figure 4.14 – Average travel time and average toll of mPQL with different values of α and same decay, and non-discounted SWPUCB with different values of w.



Source: Author.

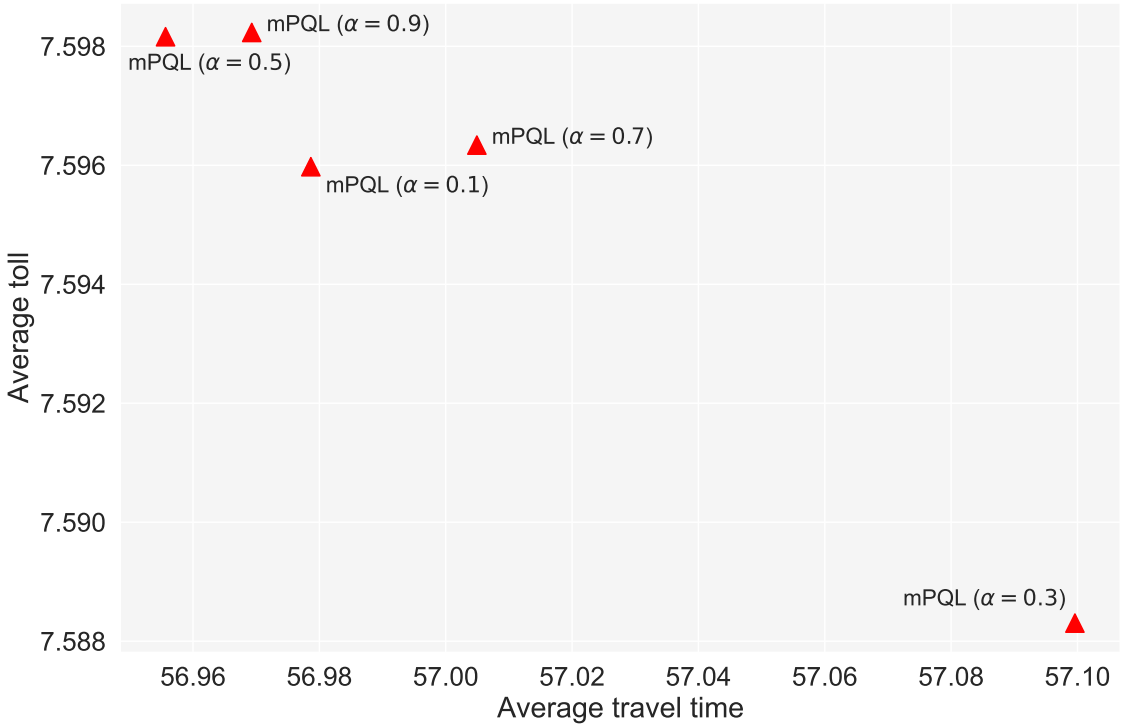
select one of them arbitrarily or based on some information (such as weights). A good coverage of the Pareto front provides a better set of solutions the agent can choose from.

Figure 4.14 shows the solutions found by mPQL with different values of α and same decay, and the non-discounted SWPUCB with different window sizes ($w = 6$ is not included because it does not converge). Half the solutions of SWPUCB are Pareto-dominating and vary widely from each other, as one favors lesser average toll and the other one favors more a low value of average travel time. However, mPQL covers such a small area of the Pareto front that its solutions, represented by triangles, overlap. Figure 4.15 expands that area to visualize the solutions of mPQL. As explained in previous sections, this demonstrates how mPQL tends to reach the same single solution of the EQS assignment, though the randomness present in mPQL allows for very small variations that are not significantly different between them.

Then, by considering coverage of the Pareto front, non-discounted SWPUCB would be a better method than mPQL. A better coverage could be achieved if other values of w and γ were used in SWPUCB.

Note that deterministic methods with no additional information, such as EQS, produce a single unique solution that does not change as there is no parameters to be

Figure 4.15 – Average travel time and average toll of mPQL con different values of α and same decay (0.997).



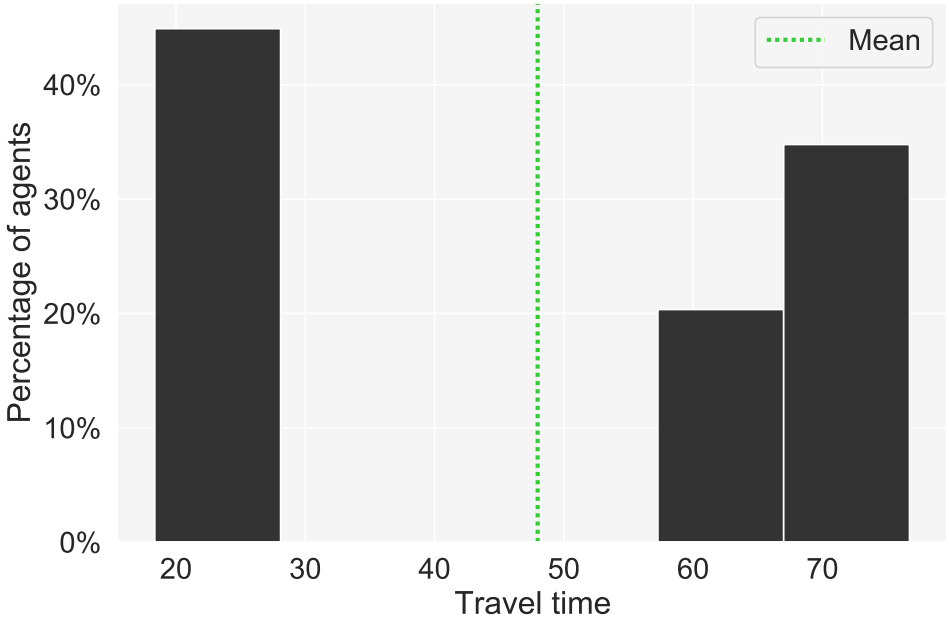
Source: Author.

tuned. The methods proposed in this work also reach a single solution but, while none of them include additional information, different parameter settings allow for exploration of other results that can be part of the Pareto front of efficient solutions.

4.4.2 Fairness

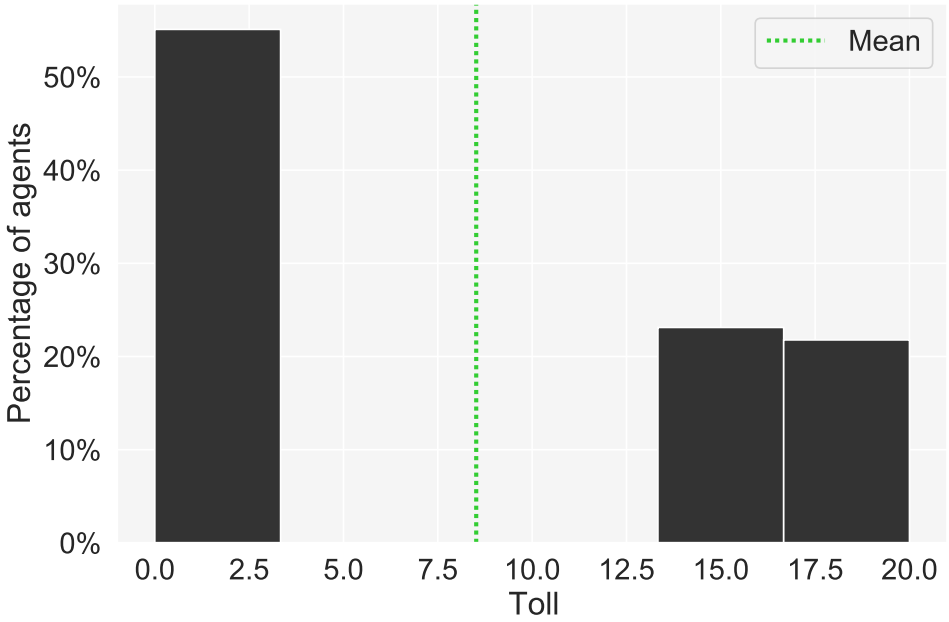
Fairness relates to the travel time experienced by an agent compared to the other agents. A fair solution would consider that all agents experiment the same travel time when there is a single objective (to minimize travel time). However, in this case, there is another objective: to minimize toll. Thus, evaluating fairness involves a less straightforward idea. Considering that only efficient routes are attractive for the agents, a low travel time implies a higher toll. The same applies to the opposite case: a high travel time implies a low toll. Taking the solution provided by non-discounted SWPUCB with $w = 12$, the travel time and toll of all agents were divided in 6 intervals or bins, and it was calculated the percent of agents that belong to each bin. In addition to these histograms, Figures 4.16 and 4.17 show the mean of travel time and toll, respectively. These figures show that the idea previously explained applies here.

Figure 4.16 – Percentage of agents for 6 different intervals of travel time, obtained by non-discounted SWPUCB with $w = 12$. The mean of travel time is also included.



Source: Author.

Figure 4.17 – Percentage of agents for 6 different intervals of toll, obtained by non-discounted SWPUCB with $w = 12$. The mean of toll is also included.



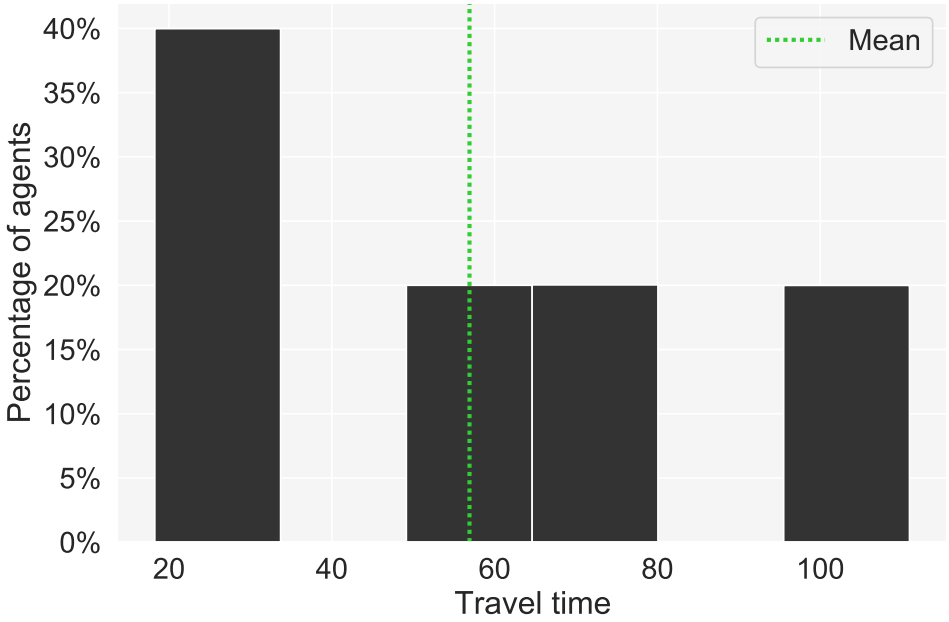
Source: Author.

Figure 4.16 shows a remarkable difference between agents with high and low travel times, and Figure 4.17 shows that this difference continues in the case of tolls. While about 45% of agents experience low travel times (first bin of Figure 4.16), they also pay high tolls (represented by the two last bins in Figure 4.17). The other ∼55% of agents have high travel times, but they pay low tolls.

Consider now the solution reached by mPQL with $\alpha = 0.5$ and $decay = 0.997$. Likewise, Figures 4.18 and 4.19 show histograms and the mean of travel time and toll, respectively. Figure 4.18 shows that there is not a difference as big as that depicted in Figure 4.16 when considering travel time, though the histograms of toll of both algorithms are similar. In this case, about 40% of agents experience low travel times, while the travel times of the remaining ∼60% are distributed in a wide interval composed of 4 bins, though only 3 bins have more than zero agents, as illustrated in Figure 4.18. Those ∼60% of agents pay low tolls, and the rest pays a significantly higher quantity, as shown in Figure 4.19. This could be interpreted as less fair than the result of non-discounted SWPUCB, since about 60% of agents whose travel times are in the aforementioned wide interval, varying from ∼50 to 110, pay almost the same toll when there is a big difference between their travel times.

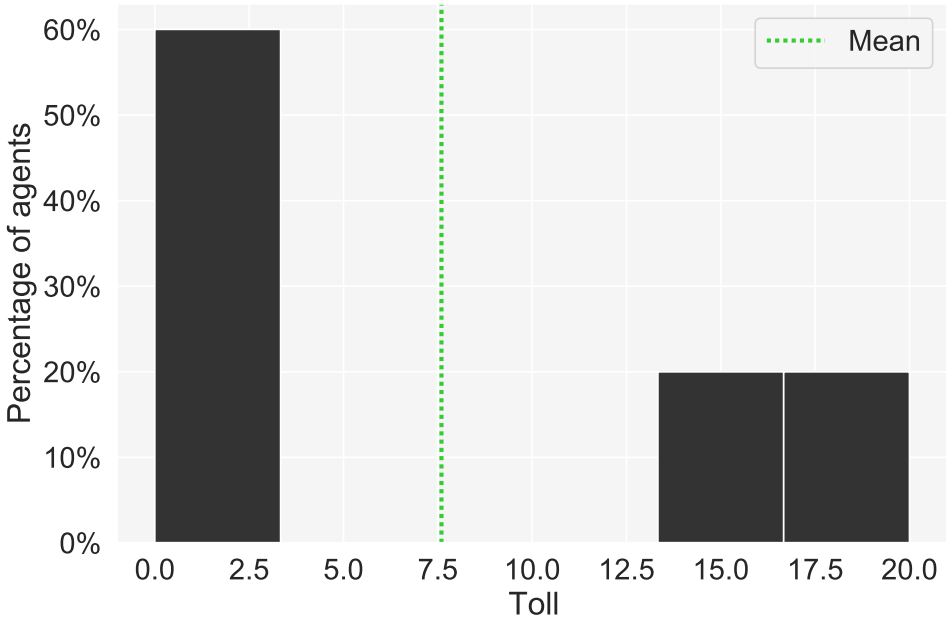
Means (or average values) help to illustrate this unfairness. In the case of non-discounted SWPUCB, we can observe that it trades a relatively high average travel time for a low average toll. The average toll is low because it is lesser than 10, which is the middle point of the X-axis in Figure 4.17. This quantity (10) is the average value that an agent could expect to pay, considering that the minimum toll is 0 and the maximum is 20. The average travel time is high because the opposite happens in Figure 4.16: the average travel time is greater than the middle point of the X-axis. This results in a sort of balance where more than 50% of agents experiences high travel times but also those more than 50% of agents pay low tolls. Meanwhile, in the case of mPQL, both means illustrated in Figures 4.18 and 4.19 are low (lesser than the middle point in their respective X-axis). More than 50% of agents experience low travel times and more than 50% of agents pay low tolls. Therefore, there must be agents that pay less than the average toll and experience travel times lower than the average travel time, which could be considered unfair, since their low tolls should imply travel times higher than the average.

Figure 4.18 – Percentage of agents for 6 different intervals of travel time, obtained by mPQL with $\alpha = 0.5$ and $decay = 0.997$. The mean of travel time is also included.



Source: Author.

Figure 4.19 – Percentage of agents for 6 different intervals of toll, obtained by mPQL with $\alpha = 0.5$ and $decay = 0.997$. The mean of toll is also included.



Source: Author.

5 CONCLUSION

Multi-objective decision-making in multi-agent settings present many challenges for multi-armed bandit research, such as non-stationarity due the presence of simultaneous learners. Research so far has addressed multiple objectives and non-stationarity separately. There is a need for methods that can deal with such still open challenges jointly. In this work, novel methods were proposed to combine the strengths of both research fronts, namely DPUCB, SWPUCB and mPQL.

DPUCB and SWPUCB are based on PUCB1, discounted UCB and sliding-window UCB. PUCB1 is a method designed to deal with multiple objectives in a multi-armed bandit setting, but oriented towards a single agent, while discounted UCB and sliding-window UCB address non-stationarity using a discount factor and a window, respectively, to give more importance to more recent rewards. Combining these approaches, it is possible to deal with both problems, multiple objectives and non-stationarity. DPUCB follows the idea of discounted rewards, while SWPUCB adds the sliding-window and includes the option to work with a discount.

mPQL derives from PQL, an state-based algorithm that works with multiple objectives but that assumes that the environment is deterministic. As the presence of many agents makes this assumption unrealistic, PQL was modified to address non-stationarity while also changing it to work in a single-state setting.

These algorithms were applied to a route choice problem that involves thousands of agents. These not only compete for a scarce resource (a route), but also have to minimize travel time and toll expenditure. The proposed methods were compared to centralized methods, as well as to a previous approach for MABs problems. The experimental results show that using a discount factor is the best approach when addressing non-stationarity and multiple objectives. Recall that the proposed DPUCB does not require the setting of a window size and, still, performs as well as SWPUCB with discount. mPQL also reaches a solution, but this solution is almost the same of a centralized solution to the TAP.

As there is more than one objective, the need for other means to compare algorithms arises, since a judgment based on solely average travel time and average toll would be inconclusive. Other criteria make possible to compare the algorithms, such as fairness or the coverage of the Pareto front of solutions that SWPUCB and DPUCB offers thanks to the variation of their parameters. mPQL falls short in this aspect, as its

solutions tend to be very similar, which does not allow to explore other efficient solutions that may interest the agents.

In general, it was demonstrated that the algorithms proposed deal with non-stationarity and multiple objectives in a multi-agent setting by using elements of algorithms that were proposed to address only one of those issues.

The future of multi-objective decision-making is likely to play a major role in route choice since, besides time and toll, one could consider battery autonomy, or further objectives that are relevant for the driver agents (or for autonomous vehicles). Among these additional objectives, those that are flow-dependent would provide a more challenging problem. Another possibility that was not explored is the addition of information provided by the user. There are many algorithms that work with that kind of data to guide the process of learning. This could help to reach a satisfying solution without the need to execute several times the proposed algorithms with different parameters. The inclusion of such information is also an interesting challenge as the the proposed algorithms focus in a direct Pareto approach, that considers a vector reward, and not the straightforward scalarization that can incorporate weights provided by the user to indicate preferences, or other kind of information to prioritize objectives.

Future work is also related to the scenarios. While there are many networks focused on single objective TAP, scenarios for a multi-objective TAP are not as widely known. The application of DPUCB and SWPUCB, and even mPQL, in a large network with multiple OD pairs and multiple objectives could give more insight about how different network features impact on their performance and what other details and/or drawbacks exists but could not be observed while using the 4-node network that was employed in this work.

The application of the proposed algorithms to other scenarios can also provide large and complex environments to deal with. Techniques like those applied in (DRUGAN; NOWÉ, 2014a) and (DRUGAN; NOWÉ; MANDERICK, 2014), which are focused in MAB algorithms and that remove sub-optimal arms (actions), can also be used in DPUCB and SWPUCB. The phase of initialization could also be improved, as currently each action is selected once during this phase. As the action of an agent influences the other agents, a more thorough sampling of combinations of the joint actions of the agents could be beneficial to the decision-making process.

Finally, since mPQL does not perform as well as DPUCB and SWPUCB, using additional data could be useful to reach different solutions and not cover just a very

small area of the Pareto set of efficient solutions. There were suggestions to use data like standard deviation and mean of the Q-values and the padding function from the PUCB1 algorithm. Some tests were run, but their results were not conclusive enough. Still, there is room for improvement in mPQL and the inclusion of other kind of data could help in this regard.

REFERENCES

AUER, P.; CESA-BIANCHI, N.; FISCHER, P. Finite-time analysis of the multiarmed bandit problem. **Machine Learning**, v. 47, n. 2/3, p. 235–256, 2002. ISSN 08856125.

BAZZAN, A. L. C.; GRUNITZKI, R. A multiagent reinforcement learning approach to en-route trip building. In: **2016 International Joint Conference on Neural Networks (IJCNN)**. [S.l.: s.n.], 2016. p. 5288–5295. ISSN 2161-4407.

BELLMAN, R. A markovian decision process. **Journal of Mathematics and Mechanics**, Indiana University Mathematics Department, v. 6, n. 5, p. 679–684, 1957.

DIAL, R. B. A model and algorithm for multicriteria route-mode choice. **Transportation Research Part B: Methodological**, v. 13, n. 4, p. 311–316, 1979.

DRUGAN, M. M.; NOWÉ, A. Designing multi-objective multi-armed bandits algorithms: A study. In: **Proc. of the International Joint Conference on Neural Networks (IJCNN)**. [S.l.: s.n.], 2013. p. 1–8.

DRUGAN, M. M.; NOWÉ, A. Epsilon-approximate pareto optimal set of arms identification in multi-objective multi-armed bandits. In: **BENELEARN 2014 - 23rd annual Belgian-Dutch Conference on Machine Learning**. [S.l.: s.n.], 2014. p. 73–80.

DRUGAN, M. M.; NOWÉ, A. Scalarization based pareto optimal set of arms identification algorithms. In: **2014 International Joint Conference on Neural Networks (IJCNN)**. [S.l.: s.n.], 2014. p. 2690–2697.

DRUGAN, M. M.; NOWÉ, A.; MANDERICK, B. Pareto upper confidence bounds algorithms: An empirical study. In: **2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)**. [S.l.: s.n.], 2014. p. 1–8.

GARIVIER, A.; MOULINES, E. On upper-confidence bound policies for switching bandit problems. In: KIVINEN, J. et al. (Ed.). **Algorithmic Learning Theory**. [S.l.]: Springer, 2011. p. 174–188.

KOCSIS, L.; SZEPESVÁRI, C. Discounted UCB. In: **Proc. of the 2nd PASCAL Challenges Workshop**. Venice: [s.n.], 2006. v. 2.

MOFFAERT, K. V.; DRUGAN, M. M.; NOWÉ, A. Hypervolume-based multi-objective reinforcement learning. In: PURSHOUSE, R. C. et al. (Ed.). **Evolutionary Multi-Criterion Optimization**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 352–366. ISBN 978-3-642-37140-0.

MOFFAERT, K. V. et al. Multi-objective x-armed bandits. In: **2014 International Joint Conference on Neural Networks (IJCNN)**. [S.l.: s.n.], 2014. p. 2331–2338.

MOFFAERT, K. van; NOWé, A. Multi-objective reinforcement learning using sets of pareto dominating policies. **J. Mach. Learn. Res.**, v. 15, n. 1, p. 3483–3512, jan. 2014. ISSN 1532-4435.

NOWÉ, A.; VRANCX, P.; HAUWERE, Y.-M. D. Game theory and multi-agent reinforcement learning. In: ____. **Reinforcement Learning: State-of-the-Art**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 441–470. ISBN 978-3-642-27645-3.

OLIVEIRA, T. B. F. de et al. Comparing multi-armed bandit algorithms and Q-learning for multiagent action selection: a case study in route choice. In: **2018 International Joint Conference on Neural Networks, IJCNN**. IEEE, 2018. p. 1–8. Disponível em: <https://doi.org/10.1109/IJCNN.2018.8489655>.

ORTÚZAR, J. d. D.; WILLUMSEN, L. G. **Modelling transport**. 4. ed. Chichester, UK: John Wiley & Sons, 2011. ISBN 978-0-470-76039-0.

RADULESCU, R. et al. Multi-objective multi-agent decision making: a utility-based analysis and survey. **Autonomous Agents and Multi-Agent Systems**, v. 34, 04 2020.

RAITH, A. et al. Solving multi-objective traffic assignment. **Annals of Operations Research**, Springer, v. 222, n. 1, p. 483–516, 2014.

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

ROBBINS, H. Some aspects of the sequential design of experiments. **Bulletin of the American Mathematical Society**, v. 58, n. 5, p. 527–535, 1952.

ROIJERS, D. M. et al. A survey of multi-objective sequential decision-making. **J. Artificial Intelligence Research**, AI Access Foundation, El Segundo, CA, USA, v. 48, n. 1, p. 67–113, out. 2013. ISSN 1076-9757.

ROIJERS, D. M.; ZINTGRAF, L. M.; NOWÉ, A. Interactive thompson sampling for multi-objective multi-armed bandits. In: ROTHE, J. (Ed.). **Algorithmic Decision Theory**. Cham: Springer International Publishing, 2017. p. 18–34. ISBN 978-3-319-67504-6.

SUTTON, R. S.; BARTO, A. G. **Reinforcement learning: An introduction**. Second. [S.l.]: The MIT Press, 2018.

THOMPSON, W. R. Likelihood that one unknown probability exceeds another in view of the evidence of two samples. **Biometrika**, v. 25, n. 3, p. 285–294, 1933.

WANG, J. Y.; EHRGOTT, M. Modelling route choice behaviour in a tolled road network with a time surplus maximisation bi-objective user equilibrium model. **Transportation Research Part B: Methodological**, v. 57, p. 342–360, 2013.

WANG, J. Y. T.; RAITH, A.; EHRGOTT, M. Tolling analysis with bi-objective traffic assignment. In: EHRGOTT, M. et al. (Ed.). **Multiple Criteria Decision Making for Sustainable Energy and Transportation Systems**. [S.l.: s.n.], 2010. p. 117–129. ISBN 978-3-642-04045-0.

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

WATKINS, C. **Learning from Delayed Rewards**. Tese (Doutorado) — University of Cambridge, 1989.

Aprendizado por reforço multiobjetivo para a seleção de ações: lidando com objetivos múltiplos e não-estacionariedade

RESUMO

A maioria dos problemas de decisão do mundo real envolve critérios múltiplos e, geralmente, conflitantes. A tomada de decisão multiobjetivo implica um planejamento baseado em um modelo para encontrar a melhor política para resolver tais problemas. Se esse modelo for desconhecido, o aprendizado através da interação fornece os meios para se comportar no ambiente. Além disso, a tomada de decisão multiobjetivo em um sistema multi-agente apresenta muitos desafios não resolvidos. Entre eles, objetivos múltiplos e não estacionariedade causados pela presença de aprendizes simultâneos têm sido tratados separadamente até agora. Neste trabalho, são propostos algoritmos que abordam ambos os problemas ao aproveitar pontos fortes de diferentes métodos. Esses algoritmos são aplicados a um cenário de escolha de rotas formulado como um *multi-armed bandit problem*, focando assim na seleção de ações. Neste problema de escolha de rotas, os motoristas devem selecionar uma rota com o objetivo de minimizar o tempo de viagem e o pedágio. Os algoritmos propostos extraem pontos fortes de trabalhos que abordam apenas uma das questões: não estacionariedade ou objetivos múltiplos. Ao combiná-los, esses algoritmos podem lidar com os dois assuntos em questão. Os métodos usados para desenvolver os algoritmos propostos são um conjunto de algoritmos baseados no *Upper-Confidence Bound* (UCB) e o algoritmo *Pareto Q-learning* (PQL). Os algoritmos baseados em UCB são o *Pareto UCB1* (PUCB1), o *Discounted UCB* (DUCB) e o *Sliding-window UCB* (SWUCB). O PUCB1 lida com objetivos múltiplos, enquanto que o DUCB e o SWUCB abordam o problema de não estacionariedade de formas diferentes. O DUCB usa um fator de desconto para dar mais importância às recompensas mais recentes, enquanto que o SWUCB usa uma janela para considerar apenas as últimas ações selecionadas. O PUCB1 foi estendido para incluir características do DUCB e do SWUCB, sendo propostos os algoritmos DPUCB e SWPUCB, respectivamente. O DPUCB apenas considera o fator de desconto, enquanto que o SWPUCB pode usar ele além da janela mencionada antes. No caso de PQL, por ser um método baseado em estados que foca em mais de um objetivo, algumas mudanças foram feitas para tratar um problema focado na seleção de ações considerando apenas um estado e usando uma taxa de aprendizado para usar o conhecimento prévio. Este

algoritmo foi chamado de mPQL.

Os resultados obtidos a partir de uma comparação em um cenário de escolha de rotas com dois objetivos (minimizar o tempo de viagem e o toll) mostram que os algoritmos propostos lidam com a não estacionariedade e objetivos múltiplos. Um dos algoritmos usados na comparação foi o EQS *assignment*, que é centralizado e cujo resultado está relacionado ao resultado do mPQL. O PUCB1 também foi usado para comparar, mostrando um desempenho ruim devido a que ele não lida com a não-estacionariedade. O DPUCB e o SWPUCB têm um melhor desempenho e chegam a resultados que podem variar segundo os valores de seus parâmetros. Mas foi observado que o fator de desconto tem uma maior relevância que o tamanho da janela. A diferença do DPUCB e do SWPUCB, o mPQL produz resultados que tendem a ser muito similares ao do EQS *assignment*. Então, variações na taxa de aprendizado só influencia nos episódios antes da convergência, mas todos seus resultados são muito semelhantes. A similitude com o EQS *assignment* acontece pela seleção aleatória uniforme de ações e a exploração.

Como há múltiplos objetivos, comparar diferentes métodos não é tão simples como no caso de apenas um objetivo. Outros critérios de comparação foram tratados: a cobertura da fronteira de Pareto e o *fairness*. O primeiro está relacionado à variedade de soluções que os algoritmos encontram e como podem cobrir diferentes partes da fronteira de Pareto de soluções eficientes. O segundo está relacionado à ideia de que agentes que pagam um toll alto deveriam ter um tempo de viagem baixo e vice-versa. Em relação à cobertura da fronteira de Pareto, O DPUCB e o SWPUCB podem chegar a resultados que cobrem diversas áreas, mas o mPQL tende a ter o mesmo resultado do EQS *assignment*, cobrindo apenas uma pequena parte.

Em conclusão, são propostos algoritmos que lidam com os problemas de objetivos múltiplos e a não-estacionariedade em conjunto, baseados na combinação de outros métodos que tratam esses problemas por separado, em quanto que usar um fator de disconto é a melhor abordagem. As limitações, diferenças e vantagens destes algoritmos são discutidas.

Palavras-chave: Tomada de decisões multiobjetivo, Escolha de rotas multiobjetivo, Aprendizado por reforço.