Revista de Informática Teórica e Aplicada - RITA - ISSN 2175-2745
Vol. 30, Num. 01 (2023) 11-23

RESEARCH ARTICLE

Multi-Objective, Multi-Armed Bandits: Algorithms for Repeated Games and Application to Route Choice

Multi-Armed Bandits Multiobjetivo: Algoritmos para Jogos Repetidos e Aplicações em Escolha de Rota

Candy A. Huanca-Anquise[1], Ana L. C. Bazzan[1]*, Anderson R. Tavares[1]

Abstract: Multi-objective decision-making in multi-agent scenarios poses multiple challenges. Dealing with multiple objectives and non-stationarity caused by simultaneous learning are only two of them, which have been addressed separately. In this work, reinforcement learning algorithms that tackle both issues together are proposed and applied to a route choice problem, where drivers must select an action in a single-state formulation, while aiming to minimize both their travel time and toll. Hence, we deal with repeated games, now with a multi-objective approach. Advantages, limitations and differences of these algorithms are discussed. Our results show that the proposed algorithms for action selection using reinforcement learning deal with non-stationarity and multiple objectives, while providing alternative solutions to those of centralized methods.
Keywords: Multi-objective decision-making — Multi-objective route choice — Reinforcement learning — Repeated games — Multiagent systems — Multi-armed Bandit algorithms

Resumo: Tomada de decisão considerando múltiplos objetivos em cenários multiagente coloca múltiplos desafios. Entre eles, pode-se mencionar o fato de ter que lidar com mais de um objetivo, bem como a não estacionariedade que é causada por aprendizado simultâneo; ambos os desafios têm sido endereçados separadamente. Neste trabalho são propostos algoritmos de aprendizado por reforço que lidam com ambos os desafios em conjunto. Tais algoritmos são aplicados em um problema de escolha de rotas, onde os motoristas precisam selecionar uma ação em uma formulação de estado único, objetivando minimizar tanto o tempo de viagem quanto o pedágio. Desta forma, nós lidamos com jogos repetidos, agora sob uma abordagem multiobjetivo. Vantagens, limitações e diferenças dos algoritmos são discutidas. Nossos resultados mostram que os algoritmos de aprendizado por reforço propostos para seleção de ação lidam tanto com a não estacionariedade quanto com os múltiplos objetivos, provendo soluções alternativas aos métodos centralizados.
Palavras-Chave: Tomada de decisão multiobjetivo — Escolha multiobjetivo de rota — Aprendizado por reforço — Jogos repetidos — Sistemas multiagente — Algoritmos Bandits

[1] *Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS), Porto Alegre - Rio Grande do Sul, Brazil*
*__Corresponding author__: bazzan@inf.ufrgs.br
DOI: http://dx.doi.org/10.22456/2175-2745.122929 • **Received:** 18/03/2022 • **Accepted:** 26/12/2022

1. Introduction

Decision-making using reinforcement learning (RL) is turning increasingly popular in multi-agent systems. Particularly, many tasks are more naturally described by means of multiple, possibly conflicting objectives. This poses a challenge when more than few agents interact and when such interplay is modeled by a repeated game. Recall that repeated games are useful to formalize RL tasks that relate to single-state scenarios. While a single-state formulation somehow means a simplified modeling, it is useful in real-world problems such as route choice, in which, usually, a single choice is made in the entire decision-making process. One popular approach to deal with single-state settings is through multi-armed bandit algorithms (henceforth MABs), which can efficiently deal with the exploration/exploitation dilemma [1]. For this class of RL problems, some algorithms have been proposed, such as the UCB (Upper Confidence Bound) family [1, 2, 3], learning automata [4], and even simplifications of the popular Q-learning (QL), as for instance in [5].

In route choice and traffic assignment problems, multiple drivers must select a route to travel from an origin to a destination. Usually, a single objective is considered, namely minimizing their travel times. However, frequently, there are other objectives to be considered, as for instance, toll. Bi-

objective approaches were proposed in [6, 7, 8]. However, these are centralized and do not involve learning. In contrast, we assume that each agent or driver performs its optimization process locally, by means of (multi-objective) RL, in a decentralized way.

Other characteristics of the route choice problem is that it involves many (potentially thousands) of agents, and that there is competition for resources. These pose challenges to RL methods, even if only one objective is considered. Specifically, the fact that there are many agents learning simultaneously causes the environment to be non-stationary.

In short, a realistic, decentralized RL-based approach to route choice involves the following issues: (i) agents learning simultaneously result in non-stationarity; (ii) there are potentially two or more objectives to be optimized; (iii) the underlying learning task is modeled as a repeated game, where there is only one state.

In the literature, existing works deal with these problems separately, focusing on a single-agent scenario with multiple objectives or tackling only the non-stationarity issue. Proposals like Pareto UCB1 [9] and Pareto Q-learning [10] deal with a multi-objective scenario but focus on a single agent (though in different ways). Pareto UCB1 is a multi-armed bandits algorithm, while Pareto Q-learning is an extension of Q-learning and, as such, considers a set of states rather than a single state. Further, Pareto Q-learning has issues to deal with non-stationarity. Non-stationarity was tackled by other members of the UCB family, such as the discounted UCB [2] and sliding-window UCB [3]. However, these do not deal with more than one objective. Meanwhile, works in multi-objective decision-making via reinforcement learning are recent [11, 12]. Most existing proposals for extending those popular algorithms for the multi-objective case concentrate on single-agent scenarios.

Therefore, most approaches concentrate on either the multi-objective case in a single-agent scenario or non-stationarity in a single-objective setting. There is a need of further investigation when we deal with multi-objective RL (MORL) and multiple agents that compete for scarce resources and make the environment non-stationary for each other. The present paper proposes and analyze extensions of existing algorithms for multi-objective decision-making in a multi-agent system, i.e., where there is inherent non-stationarity due to multiple learners acting simultaneously.

Specifically, the contributions of our work are manifold. First, we introduce extensions to Pareto UCB1 [9] (which, originally, only works in scenarios with a single agent), to cope with the severe non-stationarity of a multi-agent setting. Non-stationarity is addressed by other UCB-based algorithms, such as the aforementioned discounted UCB and sliding-window UCB. They do not deal with multiple objectives though. Besides, it is unclear whether these two variants of the UCB can indeed handle the kind of non-stationarity due to 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 successfully deal with changes that happen in large intervals of time, which is not the same type of non-stationarity that arises due to multiple learners.

Another contribution is a modification of the Pareto Q-learning algorithm. While this algorithm originally deals with more than one objective, it has issues when dealing with rapidly changing environment. We introduce extensions in order to enable its use in this kind of environment, in which the learning tasks are modeled as repeated games since the agents perform action selection in a single state.

The remainder of this paper is organized as follows. The next section discusses the background concepts and gives an overview on the related work. Section 3 details the proposed algorithms; for their evaluation, a proof of concept scenario is discussed in Section 4. Concluding remarks and future directions are given in Section 5.

2. Background and related work

In this section, we briefly introduce some relevant concepts on traffic assignment and route choice (including multi-objective variants), as well as RL. The former is necessary as it relates to the case study we discuss in Section 4.

2.1 Conventional and multi-objective traffic assignment

A traffic network can be modeled as a graph $G = (V, L)$, where V is the set of vertices that represent the intersections, and L is the set of links that describes the segments connecting a pair of vertices. There are origin-destination (OD) pairs and each one of them has a certain demand for trips, which then translates into flows on the various links. The traffic assignment problem (TAP) aims at computing the flow per link l, so that the individual (agent) travel time is minimized. It is assumed that agents (drivers) are rational, thus each selects the route with the least perceived individual cost, in order to travel from its origin to its destination.

One solution for this optimization problem is via the method of successive averages (MSA), which 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. For details on these methods, a textbook such as [13] can be consulted.

As mentioned, this assumes that each traveler aims at minimizing only its travel time. However, as aforementioned, route choices may depend on multiple cost functions, not just one. Traditionally, multi-objective traffic assignment is modeled by using a linear combination of the various objectives, as proposed, for instance, by Dial in [14] for a bi-objective assignment. Such a linear combination has some drawbacks. Some efficient routes are missed, as discussed in [8]. The key point is that algorithms that use a linear combination only identify supported solutions at extreme points, while there may be other efficient solutions that are not considered in that group but could be preferred by some users.

Hence, it is necessary to have alternative solutions. One is based on the aforementioned MSA. One issue that arises is that, in a multi-objective scenario, **there is a set of efficient routes instead of just one**. For the particular application of route choice we deal here, this is illustrated, with examples, at the end of Section 4.

To determine how users will choose their preferred route, it can be assumed that all of them are equally attractive. This leads to an equal share assignment (EQS); see [7] for more details. EQS may be unrealistic, but it is a good start point for iterative methods to compute an assignment based on multiple objectives. In order to address the limitations of an EQS-like traffic assignment for the bi-objective case, [7] has proposed a method called time surplus maximization. This is defined as the maximum time minus the actual time. This means that, given a set of routes, the one with the least negative time surplus will be the preferred one, for each traveler. This method requires information about the value of time, i.e., extra information, which is not always available.

Finally, we mention that other authors have also addressed various forms of multi-objective traffic assignment: in [6], the authors formulate heuristic solutions to compute the assignment; [15] extends their previous bi-objective user equilibrium model based on time surplus maximisation ([7]) in order to incorporate the concept of travel time budget to model how users might react to uncertainty induced by day-to-day variability in travel time caused by traffic incidents; [16] deals with edges whose costs are non-monotonic, also accounting for emission of gases (as a second objective).

In Section 4, we use some of these approaches as comparison to our results. We recall, however, that both the time surplus maximization and those heuristics are centralized approaches.

2.2 Solving the single objective TAP by means of RL

As an alternative to the MSA, the (single objective) TAP can also be solved in a decentralized fashion by letting agents select one in a set of routes. Normally, the task of the agent is to select one in a set of k shortest routes connecting its origin to its destination; such set can be computed by the agent via the k-shortest paths algorithm proposed in [17], which requires no information about other agents choices, as the input to the algorithm is only the graph G (i.e., the network topology).

Once each agent knows k routes (actions), RL is then used to let agents select routes in G that will lead to minimal costs for each agent. This can be done by means of multi-armed bandit algorithms or by a single-state formulation of QL. Normally, in both cases, the goal is to reach the Nash or user equilibrium. Each agent is rewarded by its own travel time taken to go from its origin to its destination, i.e., a single objective underlies the reward function. This kind of learning task is associated with repeated games; the only state is the node where the decision on which route to take is made (normally this node is the origin).

Examples of RL-based approaches for the TAP are: (i) Ramos et al. [18], where 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; (ii) Oliveira et al. [19], where the authors compare stateless QL to multi-armed bandit algorithms, concluding that none of the variants of the UCB algorithm (see next section) outperform QL, possibly due to the highly non-stationary nature of the route choice problem; (iii) departing from repeated games, [20] formulated the TAP as a stochastic game, where the states are the vertices in which the agent finds itself, actions are the selection of links that leave a vertex, and the reward is a function of the travel time in each link.

Apart from RL, other artificial intelligence methods have also been used to model single-objective route choice: [21] proposed a fuzzy-rule based routing, which includes qualitative decisions, whereas [22] proposed a fuzzy-based method to take the uncertainties of the travelers into account.

Finally, in the bi- or multi-objective front, we recall the aforementioned works by [14, 6, 8, 7], which, as said, are centralized.

2.3 RL algorithms for single and multi-objective action selection

Single-state learning tasks (such as route choice) are formalized as repeated games and can be solved by MABs algorithms such as UCB [1] and its variants. Next, we give just a brief overview on them and note that the terms arm and action are used interchangeably.

2.3.1 Upper Confidence Bound (UCB) family of algorithms

The UCB family of algorithms addresses the multi-armed bandit problem, where we have a set K of gambling arms, i.e., actions. Each play of an arm yields rewards, drawn from a fixed distribution. These are independent and identically distributed according to an unknown law with an unknown expectation vector. Rewards across actions are also independent.

In UCB1, the simplest algorithm, the arm 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. The first one is the current average reward. The second term is a padding function related to a one-sided confidence interval of the average reward [1]. The second term grows with the total number of actions the player has taken but decreases with the number of times this particular action has been tried. 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 quickly explore unknown actions, by initially playing each arm once, before engaging in selecting the more promising action.

The discounted UCB algorithm [2] was proposed to deal with non-stationarity. It is an adaptation of the UCB and works by averaging all past rewards while using a discount factor to give more weight to more recent observations. Another variant of UCB is sliding-window UCB ([3] *apud* [23]). It also addresses the non-stationary case, but averages the past rewards using only the last w plays.

It is worth mentioning that these two variants were created primarily to deal with a single agent learning in a non-stationary environment. 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.

2.3.2 Pareto UCB1 (PUCB1)

The PUCB1 algorithm was proposed in [9] as an extension of the standard UCB1 algorithm for the multi-objective, multi-armed bandit 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 arm, the sum of two terms is calculated: the average reward vector $\bar{R}(a)$, for arm a, and a padding function c_t corresponding to the size of a one-sided confidence interval of the average reward. This is shown in Eq. 1, where $|O|$ denotes the number of objectives, $\mathscr{A}^*$ is the set of Pareto optimal set of arms, n_e is the number of plays and $N(a)$ is the number of times arm a was played.

$$c_t = \sqrt{\frac{2\ln(n_e \sqrt[4]{|O||\mathscr{A}^*|})}{N_t(a)}} \tag{1}$$

Aiming to maximize all objectives, in each episode, the Pareto set $\mathscr{A}'$ is found, such that $\forall \ell \notin \mathscr{A}'$, there exists an arm $a \in \mathscr{A}'$ that dominates arm ℓ, as stated by Eq. 2. Then, an arm is selected uniformly at random from $\mathscr{A}'$.

$$\bar{R}(\ell) + c_t(\ell) \not\succ \bar{R}(a) + c_t(a) \tag{2}$$

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

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 selected actions to be eventually chosen against the arms with a high average reward.

On a related note, apart from PUCB1 there are other algorithms that have been proposed in the MORL literature, which use a multi-objective MAB setting. Roijers et al. [24] proposed two algorithms using online MORL with user interaction to learn the utility function of the user. A strategy was proposed by [25] to discretize the continuous action space according to samples that were Pareto-dominating from the objective space in order to approximate the Pareto front. In [26] Pareto ε-dominance is applied, which assumes the existence of a set of representative actions as a good approximation of a large Pareto optimal set of actions. Drugan et al. [27] focus on removing sub-optimal arms (actions) given a fixed budget of arm pulls. These works employ different approaches to solve the multi-objective MAB problem which are far from our aims in the present paper, such as interaction with an user, continuous set of arms (rather than discrete), an alternative Pareto dominance relation, or a budget for arm pulls. Furthermore, they focus on a single-agent setting.

2.4 Pareto Q-learning (PQL)

Pareto Q-learning [10] (PQL) integrates the Pareto dominance relation into a MORL approach. Based on Q-learning [28], PQL was not designed for repeated games. However, we include it for sake of comparison with algorithms of the UCB family (see Section 4.4).

PQL considers both the average reward vector and the set of expected future discounted reward vectors in order to compute the so-called Q-sets, which are composed of vectors. As PQL was initially defined for a state-based case (thus formulated as stochastic games), the set of expected future discounted reward vectors relies on a function, called *ND* (from non-dominated), that finds those vectors that correspond to the possible future states, and which are not Pareto-dominated.

Eq. 3 shows how the Q-sets are calculated. $\bar{R}(s,a)$ denotes the average reward vector and $ND_t(s,a)$ is the set of non-dominated vectors in the next state s, which is reached by performing action a at time step t. $\bar{R}(s,a)$ is added to each element of $\gamma\, ND_t(s,a)$. When action a is selected at state s, both terms are updated. $\bar{R}(s,a)$ is updated according to Eq. 4, 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. 5, using the non-dominated vectors in the $\tilde{Q}_{set}$ of every action a' in the next state s'.

$$\tilde{Q}_{set}(s,a) = \bar{R}(s,a) \oplus \gamma\, ND_t(s,a) \tag{3}$$

$$\bar{R}(s,a) = \bar{R}(s,a) + \frac{\vec{R} - \bar{R}(s,a)}{N(s,a)} \tag{4}$$

$$ND_t(s,a) = ND(\cup_{a'} \tilde{Q}_{set}(s',a')) \tag{5}$$

PQL learns the entire Pareto front, finding multiple Pareto optimal solutions, 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 PQL is not directly useful in rapidly changing environments in which the presence of multiple agents learning simultaneously cause non-stationarity, as for instance the route choice domain we use ahead.

Our contribution (called mPQL, detailled ahead) is an algorithm that is adapted from PQL in order to deal with multiple objectives in a non-stationary environment.

2.5 Overview

The first five lines of Table 1 summarize the gaps in the research around the related algorithms presented before. Note that only PQL and PUCB1 deal with multi-objective learning. However, neither were designed to handle non-stationarity, be it because there are multiple agents, or due to a change on the environment dynamics itself. The latter is, for example, the case of both discounted UCB and sliding-window UCB, which were formulated primarily to deal with single-agent scenarios. Hence, the obvious gap refers to the ability to deal non-stationary cases, when caused by multiple agents learning simultaneously, together with the case where there are multiple objectives. The next section then describes the algorithms we propose to address such gap.

3. Proposed algorithms

Recall that the aforementioned PUCB1 does not deal with non-stationarity (or with a multi-agent scenario for that matter). To handle non-stationarity, we developed the discounted Pareto UCB (DPUCB) and the sliding-window Pareto UCB (SWPUCB) algorithms based on ideas borrowed from the discounted and the sliding-window variants of UCB, so that we maintain those features from PUCB1 that are able to handle multiple objectives.

To address multi-objective problems, both DPUCB and SWPUCB calculate variants of the average reward vector and padding function from PUCB1, and keep using the Pareto dominance relation to find a set of efficient solutions from which they choose an action uniformly at random. In the case of mPQL, changes were made to PQL to deal with a non-stationary environment by enabling the use of previous knowledge, while adapting it to focus on single-state settings.

The last three rows of Table 1 show the features the proposed algorithms share and how they address issues the algorithms presented previously did not.

Before detailing the algorithms we propose here (sections ahead), we briefly discuss the general formulation and representation. A multi-agent, multi-objective RL problem, formulated as a repeated game can be defined as a single-state multi-agent Markov decision process (MMDP), in which $\mathscr{G}$ is the set of agents, A is the set of actions, and $\mathscr{R}(a) : A \to \mathbb{R}^{|O|}$ is the reward function defined over the set O of objectives. Note that $\mathscr{R}(a)$ is specific for each agent; however, for clarity, we drop the index that indicates the particular agent.

In particular, as the present paper uses a route choice scenario for illustrating the performance of the proposed algorithms, the aforementioned MMDP is instantiated as follows. A is a set K containing the $|K|$ shortest routes that take each agent from origin o to its destination d. The reward function is then $\mathscr{R}(a) : K \to \mathbb{R}^{|O|}$. We evaluate two objectives: travel time and flow-independent toll. Eq. 6 shows the definition of $\mathscr{R}(a)$, 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.

$$\mathscr{R}(a) = \vec{R} \tag{6}$$

Lastly, we recall that the route choice problem is highly non-stationary, given that the actions by an agent affect the travel time of the others.

3.1 Discounted Pareto UCB (DPUCB)

DPUCB takes the discount factor γ from the discounted UCB algorithm [2] to average past rewards and give more weight to recent observations in order to deal with non-stationarity. This discount factor is introduced in the calculations of the average reward vector $\bar{R}_t$ and the padding function c_t defined in Eq. 1 from the Pareto UCB1 algorithm, where u_b is the upper confidence bound of agent i.

At episode t, for action a, the upper bound in Eq. 7 is composed of the discounted average reward vector $\bar{R}_t(\gamma, a)$, as stated in Eq. 8 (where I_m is the action selected in episode m), and the discounted padding function $c_t(\gamma, a)$ defined by Eq. 9. As rewards are upper-bounded by B, its value was set to 1 in Eq. 9 since PUCB1 assumes the rewards are defined on the interval $[0, 1]$.

$$u_b = \bar{R}_t(\gamma, a) + c_t(\gamma, a) \tag{7}$$

$$\begin{aligned} \bar{R}_t(\gamma, a) &= \frac{1}{N_t(\gamma, a)} \sum_{m=1}^{t} \gamma^{t-m} \vec{R}_m(a) \mathbb{1}_{\{I_m = a\}} \\ N_t(\gamma, a) &= \sum_{m=1}^{t} \gamma^{t-m} \mathbb{1}_{\{I_m = a\}} \end{aligned} \tag{8}$$

$$c_t(\gamma, a) = 2B \sqrt{\frac{2 \ln \left(n_t(\gamma) \sqrt[4]{|O||\mathscr{A}^*|} \right)}{N_t(\gamma, a)}}, \quad n_t(\gamma) = \sum_{i=1}^{|K|} N_t(\gamma, i) \tag{9}$$

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

$$\bar{R}_t(\gamma, \ell) + c_t(\gamma, \ell) \not\succ \bar{R}_t(\gamma, a) + c_t(\gamma, a) \tag{10}$$

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

To further deal with the non-stationarity that arises due to multiple agents learning simultaneously, DPUCB employs a random initialization phase that ensures that not all agents chose the same action at each episode. Agents are also prevented from choosing already selected actions during this phase.

Table 1. Comparison of algorithms.

	Deals with non-stationarity	*Multi-objective*	*Single-state settings* (i.e., repeated game)
UCB [1]	–	–	√
PUCB1 [9]	–	√	√
Discounted UCB [2]	√	–	√
Sliding-window UCB [3]	√	–	√
PQL [10]	–	√	–
DPUCB	√	√	√
SWPUCB	√	√	√
mPQL	√	√	√

In Eq. 7, the exploitation part of the algorithm is defined by the discounted average reward $\bar{R}$ from by Eq. 8. The exploration is determined by the discounted padding function c_t from Eq. 9. The denominator $N_t(\gamma,a)$ represents the discounted number of times action a was played. Each episode in which action a is not played, $N_t(\gamma,a)$ 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 $\mathscr{A}'$ is computed according to Eq. 10 by finding those actions whose associated upper bound (Eq. 7) is not Pareto-dominated by another action (line 8). An action is selected randomly from $\mathscr{A}'$ at line 9. The agent's average reward vector $\bar{R}(\gamma,a)$ is updated using Eq. 8 (line 11). Afterwards, the current episode is updated at line 13, and the previous steps repeat until n_e episodes have passed.

Algorithm 1 Discounted + Sliding window Pareto UCB

1: **procedure** DISCOUNTEDSLIDINGWINDOWPARETOUCB($\mathscr{G}$, $|K|$, γ, w, n_e) ▷ $\mathscr{G}$ 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 $g \in \mathscr{G}$ **do**
5: **if** $t < |K|$ **then**
6: Select randomly an action that has not been taken yet
7: **else**
8: Find the Pareto set $\mathscr{A}'$ as stated in Eq. 10 if not using w; otherwise, use Eq. 14.
9: Select action a uniformly at random from $\mathscr{A}'$
10: **end if**
11: Update $\bar{R}_t$ of g according to Eq. 8 if not using w; otherwise, use Eq. 12.
12: **end for**
13: $t \leftarrow t+1$
14: **end while**
15: **end procedure**

3.2 Sliding Window Pareto UCB (SWPUCB)

Although the original sliding-window UCB algorithm does not employ the discount factor γ, SWPUCB considers it to average past rewards using the last w plays while also giving more weight to recent observations. The formulation of SWPUCB is essentially the same of DPUCB, while introducing the parameter w in the calculations of $\bar{R}_t$ and c_t. This is done by making Eq. 8 and Eq. 9 require w and by setting $\gamma = 1$ and $s = t-w+1$ in them. Changes are needed in some terms of those two equations: now we denote the average reward vector and the padding function by $\bar{R}_t(\gamma,w,a)$ and $c_t(\gamma,w,a)$, respectively. The upper bound of SWPUCB is defined by Eq. 11.

$$u_b = \bar{R}_t(\gamma,w,a) + c_t(\gamma,w,a) \tag{11}$$

$$\begin{aligned} \bar{R}_t(\gamma,w,a) &= \frac{1}{N_t(\gamma,w,a)} \sum_{m=t-w+1}^{t} \gamma^{t-m} \vec{R}_m(a)\, \mathbb{1}_{\{I_m=a\}} \\ N_t(\gamma,w,a) &= \sum_{m=t-w+1}^{t} \gamma^{t-m}\, \mathbb{1}_{\{I_m=a\}} \end{aligned} \tag{12}$$

$$\begin{aligned} c_t(\gamma,w,a) &= B\sqrt{\frac{2\ln\left(n_t(\gamma,w)\sqrt[4]{|O||\mathscr{A}^*|}\right)}{N_t(\gamma,w,a)}} \\ n_t(\gamma,w) &= \sum_{i=1}^{|K|} N_t(\gamma,w,i) \end{aligned} \tag{13}$$

These changes are extended to Eq. 10, which is now re-

presented by Eq. 14.

$$\bar{R}_t(\gamma,w,\ell)+c_t(\gamma,w,\ell) \not\succ \bar{R}_t(\gamma,w,a)+c_t(\gamma,w,a) \quad (14)$$

If one wants to use SWPUCB without discount, it suffices to set $\gamma = 1$.

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 $\mathscr{A}'$ is found at line 8 according to Eq. 14. Besides, the average reward vector $\bar{R}_t(\gamma,w,i)$ is updated applying the changes already mentioned to Eq. 8 and represented by Eq. 12.

We remark that SWPUCB is a generalization of the DPUCB algorithm (Section 3.1), where we can obtain DPUCB from SWPUCB with an infinite sliding window and a discount factor greater than zero. We opted to discuss the two as separate algorithms because drivers using DPUCB and SWPUCB perform differently, as shown in Section 4.

3.3 Modified Pareto Q-learning (mPQL)

As mentioned, we modified the PQL algorithm to better deal with non-stationary environments and to enable its application to action selection problems. This modified PQL is denoted by mPQL. Eq. 3 considers the simple mean reward. For mPQL to work in non-stationary environments, we use an exponential average (see Eq. 15). More recent rewards are exponentially more important than older ones, which is a good feature to deal with non-stationarity environments. As there is only one state, Eq. 3 changes and the new update rule is the same as that of Q-learning, but the rewards and Q-values are vectors instead of scalars, as shown in Eq. 15, where $\vec{R}$ is the observed reward vector. In fact, mPQL simplifies the procedure to update the Q-sets of each action and changes the Q-sets for Q-vectors denoted by $\vec{Q}$. Q-vectors are updated according to Eq. 15.

$$\vec{Q}(a) = \alpha\vec{R}+(1-\alpha)\vec{Q}(a) \quad (15)$$

4. Evaluation

Even in the transportation engineering literature, there are not many works that deal with bi or multi-objective TAP. Thus, finding benchmarks or even scenarios to compare with is not an easy task.

In what follows, as a proof of concept, we recur to a four-node network, formulated in [7] and depicted in Fig. 1, for which we know the solution, i.e. the optimal solutions for the bi-objective traffic assignment.

In that figure, links in red are toll-free; links 3 and 8 have a toll $\tau = 1$; $\tau = 15$ for link 2; and $\tau = 20$ for link 1. The demand from vertex o to d is 10,000 vehicle per hour, and there are six possible routes from o to d (the authors in [7] excluded routes such as 3-5-6-7). We note that all six routes are efficient when there is no flow. Details about the six routes are described in [7]. The cost functions associated with each link follow the BPR[1] family of cost functions: $T_l(f_l) =$

[1] So called due to the Bureau of Public Roads, [29].

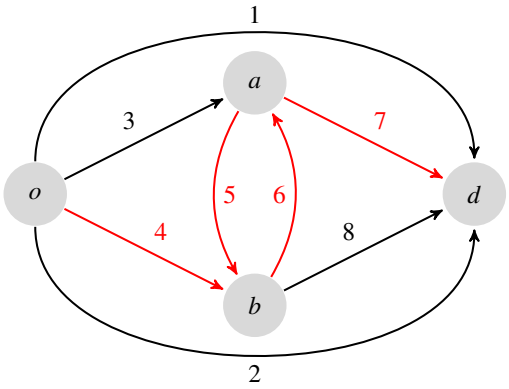

Figure 1. Four-node network (adapted from [7, 30]); red links are toll-free.

Table 2. Route characteristics of the four-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

$T_l^0(1+a(f_l/C_l)^b)$, where $a = 0.15$, $b = 4$. T_l^0 (free flow travel time, or FFTT, at link l) and C_l (l's capacity) are as defined in [7]. The tolls and FFTTs of the six routes are given in Table 2.

The algorithms we propose (see Section 3 for DPUCB, SWPUCB and mPQL) are compared with the assignment yielded by two methods: EQS (where the assignment is done equally among all efficient routes), and PUCB1. Note that we are including mPQL for comparison purposes, even though its original formulation, PQL, it is not a MABs algorithm.

Travel time and average toll are used as metrics to compare the approaches. To this aim, Table 3 reports the values of average travel time and average toll for both objectives. The values shown in Table 3, as well as the results of each algorithm, are explained in more detail in the following subsections.

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 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. The reward vector $\vec{R}$, calculated by Eq. 6, is composed by the negative values of travel time and toll of the corresponding route (action) chosen by the agent, i.e., the negative of $\vec{R}$. Thus we follow the standard practice of maximizing rewards.

Table 3. Four-node network. Average travel time and toll, for each algorithm. Values are means over 30 repetitions (std. dev. is also given). Best values in each criterion are highlighted in bold.

Algorithm	*Avg. toll*	*Avg. travel time*
EQS	7.60	56.88
PUCB1	**6.25 ± 0.01**	94.31 ± 0.55
DPUCB ($\gamma = 0.77$)	9.72 ± 0.02	**35.94 ± 0.11**
SWPUCB ($\gamma = 1$)	8.52 ± 0.10	48.06 ± 1.04
SWPUCB ($\gamma = 0.77$)	8.44 ± 0.04	45.44 ± 0.54
mPQL	7.59 ± 0.09	57.05 ± 1.30

We have implemented the EQS assignment following [8]. EQS iteratively divides the flow equally among all efficient routes. Obviously, for the reasons discussed before, we do not expect EQS to have a good performance, but it serves as a baseline.

Indeed, the EQS assignment does not perform well in this network. This method results in an allocation where all routes are efficient and, thus, they have the same flow.

4.1 PUCB1

Recall that 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 [9], 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. These values are 0 and 20, as it can be observed in Table 2. PUCB1 shares the initialization phase of DPUCB, described in Section 3.1.

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 (whose values are flow independent), but travel time depends on the flow (thus, on the decision of the whole set of agents). Hence, the distribution of values for travel times may vary across time.

Note also that PUCB1 has the highest running 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.

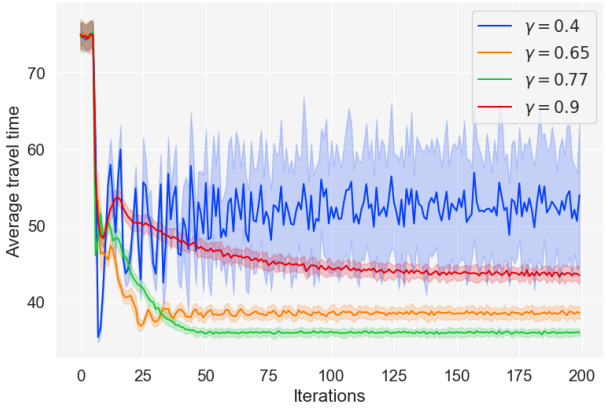

Figure 2. Average travel time of DPUCB for different values of γ.

In short, the PUCB1 yielded the worst performance, thus stressing our point that a novel approach was needed. Next we discuss the performance of our algorithms, which were detailed in Section 3, namely DPUCB, SWPUCB and mPQL.

4.2 DPUCB

As just mentioned, normalization poses problems when PUCB1 is used. For that reason, we do not include normalization in DPUCB and SWPUCB.

For the experiments using these two algorithms, the discount factor used was $\gamma = 0.77$. This value was obtained after extensive tests. Figure 2 shows the average travel time when different values of γ are used.

For the sake of information, we note that when γ is too low, there are great oscillations in the average travel time and average toll. This may be caused by a frequent change of the set (and thus, the number) of efficient routes. A low γ quickly increases the exploration term (Eq. 9), 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 (see Table 2) 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.

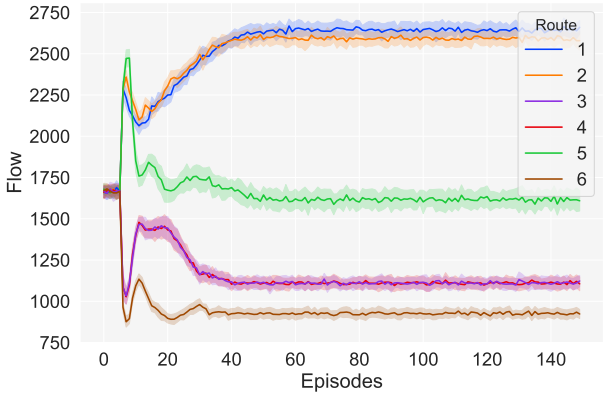

Figure 3. Route flows obtained by using DPUCB with $\gamma = 0.77$.

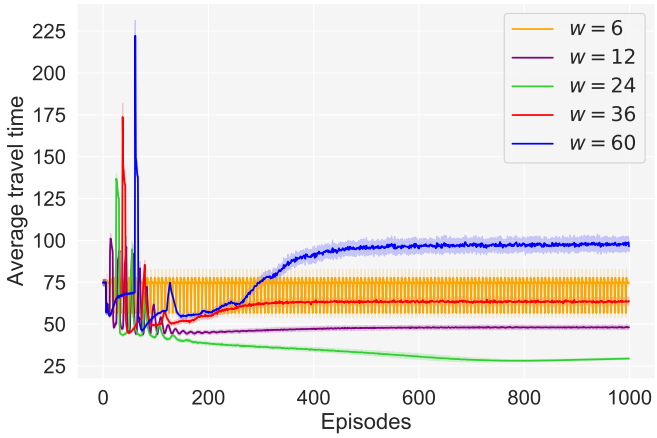

Figure 4. Average travel time for non-discounted SWPUCB with different values of w.

Note that the padding function from Eq. 9, that determines exploration, will produce low values as it depends on the ln function. As we work with non-normalized rewards, 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 presented in Figure 3, that shows the flow distribution obtained with DPUCB. 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 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 one and the least preferred route.

4.3 SWPUCB

As aforementioned, SWPUCB requires a further parameter, when compared to DPUCB, namely the window size w. Therefore, to study the effect of w alone, initially we kept $\gamma = 1$ and made experiments changing the value of w. Given that this network has six routes, we tested values of w that are multiples of six. Figures 4 and 5 show the results. 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, most curves start to converge or stabilize.

An exception is the case when $w = 6$, which shows oscillations from beginning to end. The reason is that this 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, we decided to use $w = 12$ for the experiments regarding SWPUCB.

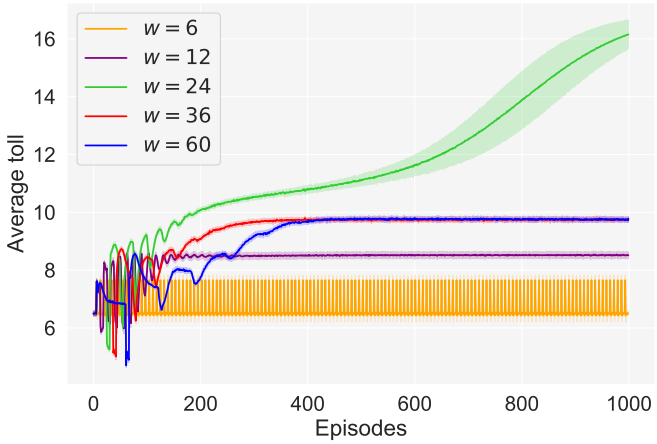

Figure 5. Average toll for non-discounted SWPUCB and different values of w.

Given this selection, the next step is to analyze how SWPUCB with $\gamma = 1$ and $w = 12$, compares to the other algorithm, not only in terms of the final values shown in Table 3, 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.

In short, while using $w = 12$ produces oscillations in the

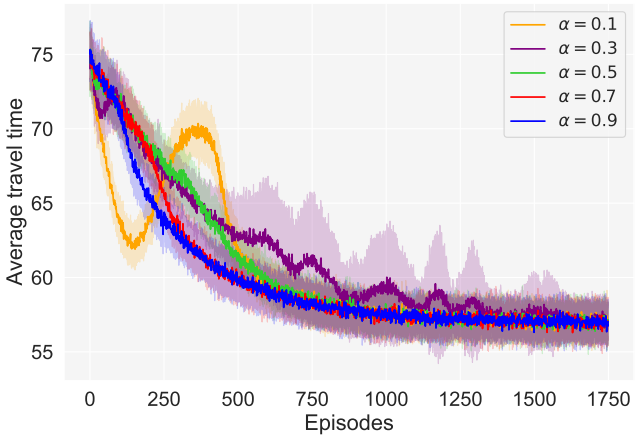

Figure 6. Average travel time of mPQL for different values of α.

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, we took the same $\gamma = 0.77$ used in DPUCB and kept the window size $w = 12$. 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. Even so, note that discounting rewards helps, as SWPUCB with discount achieves a lower average travel time and average toll than non-discounted SWPUCB, as shown in Table 3.

4.4 mPQL

The modified Pareto Q-learning was tested using different values of α in Eq. 15, namely 0.1, 0.3, 0.5, 0.7 and 0.9, while using the decaying ε-greedy method to balance exploitation and exploration. The parameter ε was initially set to 1, while a decay rate was applied to promote exploration for about 70% of the episodes. This ensures that the initialization does not produce biased results.

Different values of α tended to converge to very similar solutions (even if the oscillation pattern along episodes differs), as depicted in Figure 6, which shows the average travel time along episodes when using the aforementioned values of α. The main difference lies in the number of episodes required to converge. Higher values of α reduce this value.

It is worth mentioning that the solutions obtained by mPQL are approximately the same as those expected by the EQS assignment (compare the EQS and mPQL lines of Table 3). This happens because of the uniformly random selection, which promotes an equal number of agents in each efficient route. The exploration in mPQL also does not give preference to specific routes. As exploration decays, there are less agents switching routes and the algorithm reaches a point where the efficient routes have (almost) the same flow and the current assignment is a multi-objective user equilibrium solution (MUE), where the agents have no incentives to switch routes. Therefore, by adding certain random elements, mPQL reaches different solutions, but these are very similar in terms of average travel time, average toll and flow per route. It is that randomness that makes possible to produce solutions very similar to that of EQS, as mPQL follows a similar process to allocate flow but does not always make an evenly distribution of the agents between the efficient routes.

4.5 Discussion

Experiments show that most of the decentralized algorithms provide alternative solutions to those produced by centralized algorithms like EQS. Such solutions fall under MUE conditions [6], where no individual trip maker 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. Different values for them may favor one objective over the other. When SWPUCB uses a discount factor, 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.

The modified Pareto Q-learning (mPQL) faces difficulties as the value of the learning rate α does not influence its final result, which tends to be almost unique and equal to that of EQS. Unlike DPUCB and SWPUCB, mPQL is not sensible to variations of the parameters, i.e., mPQL outputs essentially similar traffic allocations. As such, the learning rate does not influence the final solution. This happens because of the uniformly random selection, which happens also during exploration, and that leads to an equal number of agents in each efficient route.

While the UCB 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 gives incentives to agents choosing less explored routes, while the exploration in mPQL does not give preference to specific routes. The discount factor in the UCB algorithms helps to deal with the non-stationarity, as giving more importance to recent rewards lets the agents adapt to the changing environment.

Figures 4 and 5 show how different average toll and average travel time can be obtained by varying the window size w parameter. Analogously, using other values for other parameters of the UCB family can result in other solutions. Also, DPUCB and SWPUCB tend to converge more quickly when there is a discount factor.

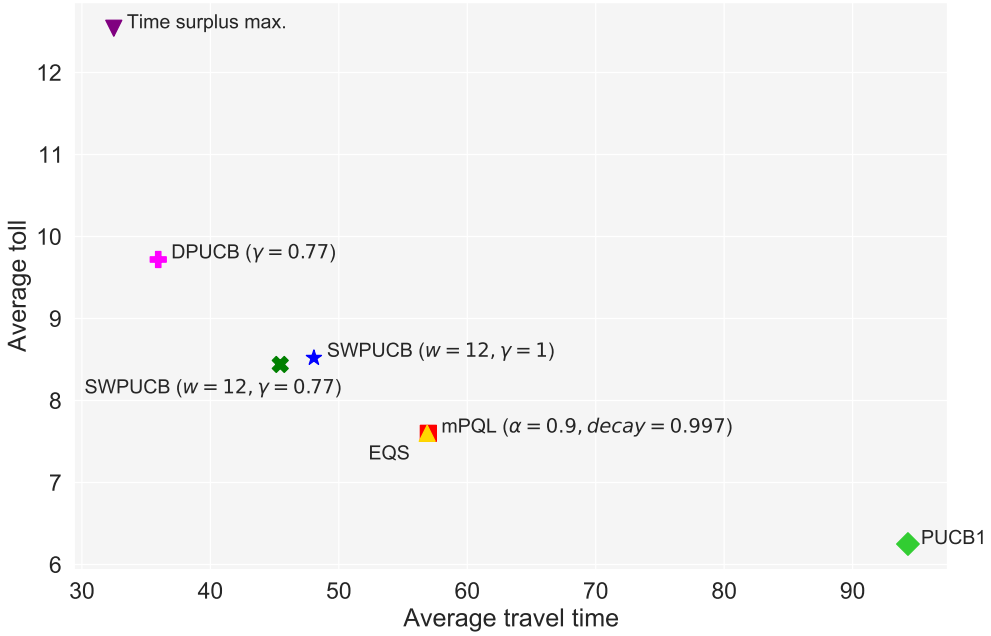

Figure 7. Average travel time and average toll of the algorithms presented in Table 3.

Overall, DPUCB and SWPUCB offer more flexibility to find different alternative solutions, but DPUCB is slightly more advantageous. Meanwhile, mPQL tends to find a solution that is similar to a centralized method (EQS).

Finally, Figure 7 shows the Pareto front formed by the results presented in Table 3. Recall that mPQL and EQS have almost the same result, which is why they overlap in the figure. 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 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 since they are deterministic. Though time surplus 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.

5. Conclusion

Multi-objective decision-making in a multi-agent system is a challenging topic as it not only deals with multiple possibly conflicting objectives, but the adaptation of an agent makes the environment non-stationary scenario for the others.

While there are works that address multiple objectives and non-stationarity, they have focused on only one of these topics. There is a need for methods that can deal with such still open challenges jointly.

In this paper we deal specifically with decision-making and learning in repeated games. For single-objective case, this is normally tackled by the UCB family of MABs algorithms.

The algorithms proposed in this work – SWPUCB, DPUCB and mPQL – address both of the issues, extending existing algorithms that approach those problems in separate ways. SWPUCB, DPUCB and mPQL were applied to the route choice problem while considering two objectives: to minimize travel time and toll, and compared to a centralized algorithm as well as a MABs algorithm that deals with multiple objectives (PUCB1).

Recall that, in the multi-objective case, there is a set of efficient solutions (as opposed to a single one in the case a single objective such as travel time is considered). Hence, there is a need for another means to compare the output of the algorithms. Indeed, an assessment based on solely average travel time or average toll would be inconclusive. Other criteria for such a comparison is, for instance, the variety of solutions that allows a better coverage of the Pareto front of efficient solutions that SWPUCB and DPUCB offer thanks to variation of their parameters values. It must be noted that DPUCB performs better as it does not require the additional parameter w and its discount factor γ can downplay w.

mPQL does not perform as well as the UCB algorithms since it converges to solutions very similar to that of the centralized EQS algorithm. Moreover, the use of an exponential average scheme does not significantly influence results. The main difference between mPQL and DPUCB and SWPUCB lies in how the agents explore.

In summary, the proposed algorithms address both non-stationarity and multiple objectives by taking and combining strengths from algorithms that deal with only one of those is-

sues. DPUCB and SWPUCB offer varied alternative solutions to those of centralized methods in a route choice problem due to changes in their parameters, but DPUCB is more advantageous due to the discount factor having more relevance than the window size parameter. Meanwhile, though mPQL also deals with non-stationarity and multiple objectives, its solutions are very similar to that of a centralized method.

As multiple objective decision-making is likely going to become more relevant in the route choice problem in the future, one could consider battery autonomy, or further objectives that are pertinent to the driver agents. Among these additional objectives, those that are flow-dependent would be more challenging. Other possible future directions to be explored are the addition of information provided by the user to guide the learning process, and to consider dynamically changing the value along time for some of the parameters, such as γ.

Ana Bazzan is partially supported by CNPq (grant 304932/2021-3). This work was partially funded by the Brazilian agencies MCTI/MC/CGI and São Paulo Research Foundation (FAPESP, grant 2020/05165-1), and by CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil) - Finance Code 001. Ana Bazzan is also grateful to the German Federal Ministry of Education and Research (BMBF), Käte Hamburger Kolleg Cultures des Forschens/ Cultures of Research.

Authors contributions

Candy A. Huanca-Anquise has performed the literature research and written the section on background and related work, under supervision of Ana L.C. Bazzan and Anderson R. Tavares. She has also written the code for the various algorithms reported in this paper, and designed and carried out the experiments (again, under supervision of the other two authors). All authors have collaborated in the analysis of the results, as well as on the writting.

References

[1] AUER, P.; CESA-BIANCHI, N.; FISCHER, P. P. finite-time analysis of the multiarmed bandit problem. *Machine Learning*, Netherlands, v. 47, n. 2, p. 235–256, May 2002.

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

[3] GARIVIER, A.; MOULINES, E. On upper-confidence bound policies for switching bandit problems. In: *Algorithmic Learning Theory*. Berlin, Germany: Springer Berlin Heidelberg, 2011. p. 174–188.

[4] NARENDRA, K. S.; THATHACHAR, M. A. L. *Learning Automata: An Introduction*. Upper Saddle River, USA: Prentice-Hall, 1989.

[5] CLAUS, C.; BOUTILIER, C. The dynamics of reinforcement learning in cooperative multiagent systems. In: *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. Menlo Park, USA: American Association for Artificial Intelligence, 1998. p. 746–752.

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

[7] 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*, United Kingdom, v. 57, p. 342–360, November 2013.

[8] WANG, J. Y. T.; RAITH, A.; EHRGOTT, M. Tolling analysis with bi-objective traffic assignment. In: *Multiple Criteria Decision Making for Sustainable Energy and Transportation Systems*. Berlin, Germany: Springer Berlin Heidelberg, 2010. p. 117–129.

[9] DRUGAN, M. M.; NOWE, A. Designing multi-objective multi-armed bandits algorithms: A study. In: *The 2013 International Joint Conference on Neural Networks (IJCNN)*. New York, USA: IEEE, 2013. p. 1–8.

[10] MOFFAERT, K. V.; NOWé, A. Multi-objective reinforcement learning using sets of pareto dominating policies. *The Journal of Machine Learning Research*, New York, USA, v. 15, n. 1, p. 3483–3512, January 2014.

[11] RADULESCU, R. et al. Multi-objective multi-agent decision making: a utility-based analysis and survey. *Autonomous Agents and Multi-Agent Systems*, Netherlands, v. 34, n. 10, p. 1–52, January 2020.

[12] HAYES, C. F. et al. A practical guide to multi-objective reinforcement learning and planning. *Arxiv preprint arXiv:2103.09568*, [S. l.], p. 1–41, March 2021.

[13] ORTÚZAR, J. de D.; WILLUMSEN, L. G. *Modelling transport*. 4. ed. Chichester, United Kingdom: John Wiley & Sons, 2011.

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

[15] EHRGOTT, J. Y. T. W. and Matthias. *Journal of Multi-Criteria Decision Analysis*, United Kingdom, v. 25, n. 1-2, p. 3–15, March 2018.

[16] TIDSWELL, J. et al. Minimising emissions in traffic assignment with non-monotonic arc costs. *Transportation Research Part B: Methodological*, United Kingdom, v. 153, p. 70–90, November 2021.

[17] YEN, J. Y. Finding the k shortest loopless paths in a network. *Management Science*, USA, v. 17, n. 11, p. 712–716, July 1971.

[18] RAMOS, G. de O.; SILVA, B. C. da; BAZZAN, A. L. C. Learning to minimise regret in route choice. In: *Proc. of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017)*. New York, USA: IFAAMAS, 2017. p. 846–855.

[19] 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*. New York, USA: IEEE, 2018. p. 1–8.

[20] 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)*. New York, USA: IEEE, 2016. p. 5288–5295.

[21] PEETA, S.; YU, J. W. A hybrid model for driver route choice incorporating en-route attributes and real-time information effects. *Networks and Spatial Economics*, Netherlands, v. 5, n. 1, p. 21–40, March 2005.

[22] HENN, V. Fuzzy route choice model for traffic assignment. *Fuzzy Sets and Systems*, Netherlands, v. 116, n. 1, p. 77–101, November 2000.

[23] CHEN, Y.; SU, S.; WEI, J. A policy for optimizing sub-band selection sequences in wideband spectrum sensing. *Sensors*, Basel, Switzerland, v. 19, n. 19, p. 1–17, September 2019.

[24] ROIJERS, D. M.; ZINTGRAF, L. M.; NOWÉ, A. Interactive thompson sampling for multi-objective multi-armed bandits. In: *Algorithmic Decision Theory*. Germany: Springer International Publishing, 2017. p. 18–34.

[25] MOFFAERT, K. V. et al. Multi-objective x-armed bandits. In: *2014 International Joint Conference on Neural Networks (IJCNN)*. New York, USA: IEEE, 2014. p. 2331–2338.

[26] DRUGAN, M.; NOWE, A. Epsilon-approximate pareto optimal set of arms identification in multi-objective multi-armed bandits. In: . Brussels, Belgium: BENELEARN 2014 - 23rd annual Belgian-Dutch Conference on Machine Learning, 2014. p. 73–80.

[27] 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)*. New York, USA: IEEE, 2014. p. 1–8.

[28] WATKINS, C. J. C. H.; DAYAN, P. Q-learning. *Machine Learning*, Hingham, USA, v. 8, n. 3, p. 279–292, May 1992.

[29] United States Bureau of Public Roads. *Traffic Assignment Manual*. Washington, USA: United States Bureau of Public Roads, 1964.

[30] ANQUISE, C. A. H. *Multi-objective reinforcement learning methods for action selection: dealing with multiple objectives and non-stationarity*. Dissertação (Mestrado) — Instituto de Informática, UFRGS, Porto Alegre, 2021.