A Multiobjective Reinforcement Learning Approach to Trip Building

Guilherme Dytz dos Santos[1], Ana L. C . Bazzan[1,*]

[1]Computer Science, Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, RS, Brazil

Abstract

Using reinforcement learning to support agents in making decisions that consider more than one objective poses challenges. We formulate the problem of multiple agents learn to travel from A to B in a traffic network as a reinforcement learning task in which we take into account: non-stationarity, more than one objective, and a a stochastic game based model. To solve this task, we propose an adaptation of an existing algorithm (PQL), which deals with more than one objective. We have applied the proposed method to a scenario in which hundreds of agents have to learn how to travel from their origins to their destinations, aiming at minimizing their travel times, as well as the carbon monoxide vehicles emit. Results show that the adapted algorithm is able to find efficient routes for the agents.

Keywords

reinforcement learning (RL), multiobjective RL, routing, route choice

1. Introduction

Decision-making using multiobjective reinforcement learning (RL) is turning popular in multi-agent systems. The reason is that many tasks are more naturally described by means of multiple, possibly conflicting objectives. Although this poses more challenges to RL, especially when more than few agents interact, such formulation is useful in real-world problems such as making decision regarding trips in traffic networks. In this domain, multiple drivers must learn to reach their destinations, starting at their origin locations. Depending on the formulation of the RL task, agents construct their routes by making decisions about which link to follow, once they find themselves at given locations. Usually, for such learning task, a single objective is considered, namely minimizing their travel times. However, frequently, there are other objectives to be considered. For instance, there are works that optimize two objectives – toll and travel time – such as [1, 2, 3]. However, these are centralized and do not involve RL.

Route choice using travel time and toll by means of RL is addressed by [4]; however this work deals with agents selecting among k pre-computed routes that take the agents from their origins to their destinations. This means that the RL task involves only one state (stateless RL), namely the origin node, where an agent makes a decision (select one of the k routes). Then the agent follows the selected route without making further decisions during the trip. For this

ATT'22: Workshop Agents in Traffic and Transportation, July 25, 2022, Vienna, Austria
gdsantos@inf.ufrgs.br (G. D. d. Santos); bazzan@inf.ufrgs.br (A. L. C. . Bazzan)
http://www.inf.ufrgs.br/~bazzan (A. L. C. . Bazzan)
0000-0002-2803-9607 (A. L. C. . Bazzan)

kind of problem, the work described in [4] extends a Bandit algorithm like UCB [5], in order to account for multiple objectives and for multiple learning agents.

In contrast to the aforementioned works, in the present paper, each agent or driver not only performs its optimization process locally (in a decentralized way), by means of multi-objective RL (MORL), but also it builds its trip by making decisions at each intersection (that function as states). Hence, the underlying learning task is state-based. Other characteristics of the problem we deal with is that it involves many agents competing for resources. This also poses 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 routing 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 stochastic game, where states are the intersections of the networks, where decisions about which link to take are made.

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 method. Its evaluation in 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 RL, as well as on traffic assignment, route choice, and routing (including multi-objective variants).

2.1. Reinforcement Learning

In RL, an agent learns how to act in an environment, by receiving a feedback (reward) that measures how its action has affected the environment. The agent does not a priori know how its actions affect the environment, hence it has to learn this by trial and error, which characterizes an exploration phase. Such phase may be very noisy in multiagent scenarios, given that all agents are learning simultaneously and hence have to adapt to others' exploration processes. However, agents should not only explore; in order to maximize the rewards of their actions, they also have to exploit the gained knowledge. Thus, there must be an exploration-exploitation strategy that is to be followed by the agents. One of these strategies is ε-greedy, where an action is randomly chosen (exploration) with a probability ε, or , with probability 1-ε, the best known action is chosen, i.e., the one with the highest value so far (exploitation).

In the exploitation phase, at each interaction, it is assumed that an agent has sensors to determine its current state and can then make an action. The reward perceived or received from the environment is used to update its policy, i.e., a mapping from states to actions. This policy can be generated or computed in several ways. For the sake of the present paper, we concentrate on a model-free, off-policy algorithm called Q-learning or QL [6] and its extensions. QL estimates the so-called Q-values using a table to store the experienced values of performing a given action when in a given state. The value of a state s_t and action a_t at time t is updated based on Eq. 1, where $\alpha \in [0, 1]$ is the learning rate, $\gamma \in [0, 1]$ is the discount factor, s_{t+1} is the

next state and r_t is the reward received when the agent moves from s_t to s_{t+1} after selecting action a_t in state s_t.

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

RL can be employed to compute how drivers learn how to go from A to B in a transportation network. The next section thus introduces this learning task.

2.2. How to Travel from A to B?

In this section, we first discuss the differences among the various terms that are related to this problem. Then, we introduce the concepts related to the definition of an urban network (topology), as well as the demand side. Finally, we discuss the need for multiobjective approaches.

Given a road network and the demand for its use (number of trips per unit of time), the task of finding out how each unit of such demand realizes its travel needs can be solved in different ways, although all seek to compute the so-called user equilibrium, i.e., the condition in which no user is better off by changing its route. We recall that, in the literature, demand can be expressed as trip, traveller, user, agent, or vehicle, depending on the community.

In the transportation community this problem is normally known as traffic assignment problem (TAP), which is solved in a centralized fashion, mostly via a macroscopic approach using a volume-delay function (VDF), which estimates the travel time as a function of the volume of trips; see Chapter 10 of [7] for details. In the computer science community, that task is frequently solved by means of RL in a decentralized way, in the sense that agents learn how to reach their destinations, by performing experiments (exploration) until they learns their respective ways (which is equivalent to the condition under the user equilibrium). This tasks admits two variants: in the first, an agent knows a set of precomputed routes (or is able to compute them), and its task is to choose a route among them (again by means of experimenting with several selections of actions or routes). Once such choice is done, an agent travels the route, without changing its mind during the trip. The second variant departs from the assumption that agents are given a set of routes. Instead, agents make choices during the trip (e.g., at each intersection, or at each important point of decision), thus building the actual route while traveling. In terms of RL, these two variants have a key difference as well. While the latter is a classical RL task with a set of states (e.g., each decision points), and a set of actions, the former belongs to the stateless RL front, where agents basically need to select an action at the origin node (the sole state). Moreover, this learning task resembles Bandit algorithms such as UCB [5]. Henceforth, we refer to the former as route choice (as the agent simply chooses a route and remain on it), while the latter is referred here as routing to convey the idea that the trip is build during the actual routing task.

Regarding approaches that were already proposed to tackle this problem, in the literature, there are various approaches to solve the TAP; here we refer only to those that address more than one objective such as [8, 3, 2, 1]. Others can be found in [9]. Regarding route choice, the reader is referred to [10, 11]; in particular, multiobjective route choice is tackled by [4]. All these works use a macroscopic approach in the sense that VDF's are used to provide a reward for each agent based on the usage of a given route. Routing is less common in the literature but

some works appear in [12, 13, 14, 15]. While the former employs a VDF to compute rewards (thus, a macroscopic approach), the last two use a microscopic simulator, where vehicles realize the actual driving movements. As such, rewards (e.g., travel times) are given by the simulator itself and correspond to a more realistic reward (as opposed to the estimation provide by a VDF).

This said, next we briefly review the concept of transportation network and the demand that uses it. Formally, a traffic network can be modeled as a graph $G = (V, L)$, where V is the set of vertices or nodes that represent the intersections, and L, the set of links that describe the segments connecting a pair of vertices. In the network, trips are distributed among a set of origin-destination (OD) pairs; each corresponding to a certain demand for trips. These then translates into flows on the various links.

In all methods it is assumed that agents (drivers) are rational, thus each selects the route or link with the least perceived individual cost, in order to travel from its origin to its destination. Several factors may influence this decision, such as travel time, distance, monetary cost (e.g., toll), fuel consumption, battery, emission of pollutants, etc.

Traditionally, multi-objective traffic assignment is modeled by using a linear combination of the various objectives, as proposed, for instance, by Dial in [8] for a bi-objective assignment. Such a linear combination has some drawbacks. Some efficient routes are missed, as discussed in [3]. 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 issue that arises is that, in a multi-objective scenario, there is a **set** of efficient solutions instead of just one. Different solutions have been proposed in the transportation and optimization communities; see for instance [8, 3, 2, 1]. We remark that all these methods rely on a set of (pre-computed) k shortest paths and they are centralized. Thus, they are not useful in a multiagent environment where each agent should learn by its own experience using RL, nor they fit a state-based RL task, where agents learn at each intersection.

When employing RL to the problem of how agents should travel from A to B, there are two possible approaches. The first one relates closely to the TAP in the sense that k shortest routes are computed for agents in each OD pair, although it is not centralized as it is the case with the aforementioned methods. Examples of this approach are [10, 11] among others.

The second variant relax the assumption that each agent selects one of the k shortest routes (and then travel on it, without deviating from the selected route). Rather, agents make choices *en route*, i.e., they decide which link to follow once they find themselves in each vertex. This is the approach followed in [16, 14] and also in the present paper.

2.3. Multiobjective Approach to Q-learning (PQL)

In this section we discuss how to use QL when agents need to deal with more than one objective. Besides the discussion ahead, we refer readers to other approaches; see [17, 18].

In order to extend the aformentioned QL algorithm, [19] proposed Pareto Q-learning (PQL), which integrates the Pareto dominance relation into a MORL approach. PQL considers both the immediate 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. In PQL, 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. 2 shows how Q-sets are calculated. $\bar{R}(s,a)$ denotes the immediate 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 a is selected at s, both terms are updated. $\bar{R}(s,a)$ is updated according to Eq. 3, 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. 4, using the non-dominated vectors in the $\tilde{Q}_{set}$ of every action a' in the next state s'.

$$\tilde{Q}_{set}(s,a) = \bar{R}(s,a) \oplus \gamma\, ND_t(s,a) \tag{2}$$

$$\bar{R}(s,a) = \bar{R}(s,a) + \frac{\vec{R} - \bar{R}(s,a)}{N(s,a)} \tag{3}$$

$$ND_t(s,a) = ND(\cup_{a'} \tilde{Q}_{set}(s',a')) \tag{4}$$

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 cannot 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 assumes that the environment is deterministic, hence it is not directly useful for 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 aPQL, detailed ahead) is an adaptation of PQL to deal with multiple objectives and with multiple agents (thus, a non-stationary environment).

3. Methodology

3.1. Markov Decision Process Definition

Before presenting the details of the proposed method, some considerations to the Markov decision process (MDP) underlying the learning task should be addressed. In an MDP, we define a set of states S, a set of actions A, a reward function $R : S \times A \rightarrow \mathbb{R}$, and a probabilistic state transition $T(s,a,s') \rightarrow [0,1]$, with $s \in S$ being the state the agent is currently in, $a \in A$ the action the agent takes and $s' \in S$ being the state it might end up, taking action a in state s.

In the routing learning task, each intersection within a traffic network is a state s, and the actions are the links an agent might take when in s (i.e., links that leave the intersection). In short, agents build their routes by making decisions about which link to follow, when finding themselves in a given intersection.

Regarding the environment, the transition model is implemented by a microscopic traffic simulator itself, which moves vehicles along the network. As for the rewards, since we deal with a multiobjective scenario, the rewards returned by the simulator are assembled in a vector; this way, the reward function definition is $R : S \times A \rightarrow \mathbb{R}^n$, where n is the number of objectives

to be optimized. As detailed in Section 3.3, in the present paper we deal with agents aiming at optimizing two objectives: travel time and carbon monoxide emission. However, the proposed method is general and can be used if more objectives are taken into account.

3.2. Time Steps and Episodes

In a traditional RL environment, at each time step, each agent finds itself in state s, chooses action a, receives a reward and transitions to a state s'. In the learning task at hand, time steps are controlled by the microscopic simulator and correspond to an unit of time such as one second. This has important consequences. First, not all time steps count as decision-making steps; these only happen when an agent is at an intersection. The rest of the time steps are used (e.g., see plots ahead) just as clock units. Second, different agents find themselves either in decision-making states, or are moving along a link (thus, not updating their value functions), or have reached their destinations. In the latter case, an episode is finished *for that particular agent*. Since travel times and destinations are different for most of the agents, episodes are *not synchronous*. Therefore, in general, we do not refer to episodes; rather, we use the clock units or time steps of the simulator to depict time. To control the simulation time, we introduce a parameter called maximum number of steps, M.

3.3. Reward Calculation

In this section we detailed how we deal with multiple objectives in the reward function.

In the exploration phase, due to uncoordinated action selection by the agents, congestion may occur, which impacts the reward of the agents, while also inserting a lot of noise in the learning process, which then leads to slow convergence. In order to speed up this process, and avoid agents wandering around the network performing experimentation, two hyper-parameters were introduced. The first one is a bonus b that is added to the reward when an agent reaches its destination. The second is a penalty p_d that is subtracted to the reward when an agent is in a dead-end node, i.e., a node from which it cannot reaches its destination at all.

Given that the reward accounts for more than one objective (and thus is represented by a vector of size n, where n is the number of objectives to optimize), both the bonus b and the penalty p_d are added (subtracted in the case of p_d) to the corresponding scalar value, as stated in Eq. 5 (for the bonus) and Eq. 6 (for the penalty).

$$\vec{r} = \vec{r} + b \cdot \vec{1} \tag{5}$$

$$\vec{r} = \vec{r} - p_d \cdot \vec{1} \tag{6}$$

A further hyper-parameter of our model resembles a kind of toll on emissions of pollutants. It is designed based on the assumption that certain links of the network should be spared from high emission levels; this may be the case of links close to parks, hospitals, residential areas, etc. This parameter is the penalty p_e; it acts similarly to the aforementioned penalty p_d, with the difference that p_e is subtracted only to the element of the reward vector that refers to the emission. Let $\hat{e}_i$ be an i-th basis vector, where i is the position of the emission element of the

reward vector. Then the penalty p_e is used to update the reward as in Eq. 7. Algorithm 1 summarizes how the reward is calculated.

$$\vec{r} = \vec{r} - p_e \cdot \hat{e}_i \tag{7}$$

Algorithm 1: Compute Reward

Data: b, p_d, p_e, $agent$
Result: $\vec{r}$

$\vec{r} \leftarrow -[agent.last\_egde\_travel\_time, agent.last\_egde\_emission]$;
$\vec{r} \leftarrow \vec{r} + b \cdot \vec{1}$;
$\vec{r} \leftarrow \vec{r} - p_d \cdot \vec{1}$;
$\vec{r} \leftarrow \vec{r} - p_e \cdot \hat{e}_i$;

3.4. Action Selection: Adapted Pareto Q-Learning (aPQL)

Originally, in PQL (i.e., as proposed in [19]), every pair state-action (s, a) is associated with a set of non-dominated points that represents the Pareto front. Q-sets are computed as in Eq. 2 and are used to compute a hypervolume of the Pareto front. This is then used to determine the best action. In our case, this was not effective given that the Q-sets tend to contain few points only, thus computations based on hypervolume tend to be ineffective. Therefore, we had to modify the action selection strategy. Instead of using the hypervolume of the Pareto front to determine the best action, we let each agent select randomly (with uniform probability, at each decision point, and independently of one another) which objective is to be optimized at that given step. This seems to be a fair substitution for the hypervolume and avoids biasing towards one objective. Once a particular objective is randomly drawn, then an agent uses the ε-greedy strategy to select an action, given the Q-set. Note that the approach continues to be multiobjective, as all objectives have their values updated after a given action was selected.

Given this adaptation in the action selection part of the approach proposed in [19], henceforth we refer to ours as aPQL (adapted PQL). The remainder of aPQL follows basically the same procedures underlying PQL.

3.5. The Whole Picture

Algorithm 2 shows how our method works for each agent. The required parameters are: M (maximum number of time steps); b, p_d and p_e (bonus, the destination, and the emission penalty discussed in Section 3.3); the discount factor γ as well as the ε (ε-greedy exploration strategy).

Algorithm 2: aPQL Learning Procedure for Each Agent

```
   Data: M, b, p_d, p_e, γ, ε
1  step ← 0;
2  agent.start();
3  while step < M do
4      if agent has reached a node then
5          if agent crossed an emission toll link then
6              p'_e ← p_e;
7          else
8              p'_e ← 0;
9          end
10         if reached an ending node then
11             case reached its destination do
12                 r⃗ = compute_reward(b, 0, p'_e, agent);
13             end
14             case reached a dead-end do
15                 r⃗ = compute_reward(0, p_d, p'_e, agent);
16             end
17             agent.update_non_dominated_and_avg_rewards(r⃗, γ);
18             agent.restart();
19         else
20             r⃗ = compute_reward(0, 0, p'_e, agent);
21             Q_set = compute_Q_set(agent.current_state());
22             agent.update_non_dominated_and_avg_rewards(r⃗, γ);
23             if random() > ε then
24                 chosen_obj = agent.random_objective();
25                 action = arg max_a Q_set[a][chosen_objective];
26             else
27                 action = agent.random_action();
28             end
29         end
30     end
31     step ← step + 1;
32 end
```

The procedure is as follows: the agent and the counter that accounts for the simulation time step are both initialized. The simulation then runs until this counter reaches the value M.

The `if` statement in lines 4 - 30 checks if the state of an agent is a node/intersection. If it is not, nothing happens (this means the agent is traveling in a link); otherwise, depending on whether or not the agent has crossed a link in which an emission penalty should be imposed (lines 5 - 9), variable p'_e is set.

Next, the agent needs to know if it is in an ending node (`if` statement in lines 10 - 18), i.e.,

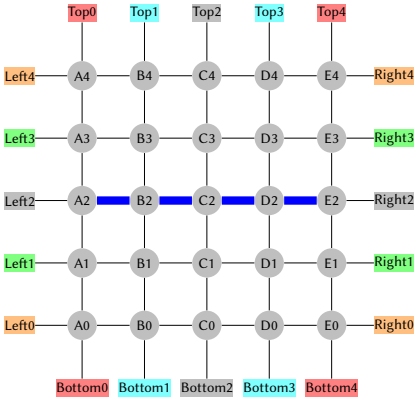

Figure 1: 5x5 Grid Network (as in [14])

OD-Pairs			Demand
Bottom0|Top4	/	Top4|Bottom0	102
Bottom1|Top3	/	Top3|Bottom1	86
Bottom3|Top1	/	Top1|Bottom3	86
Bottom4|Top0	/	Top0|Bottom4	102
Left0|Right4	/	Right4|Left0	102
Left1|Right3	/	Right3|Left1	86
Left3|Right1	/	Right1|Left3	86
Left4|Right0	/	Right0|Left4	102

Table 1: Demand (nb. of trips) per OD-pair

in a destination or in a dead-end node. Recall that a dead-end node is one from which it is not possible to reach a destination at all, an information provided by SUMO. In both cases, Algorithm 1 is used to compute the reward, as stated in Section 3.3: in the case of a destination node (line 12), a non-zero bonus is given; in the case of a dead-end (line 15), the reward includes a non-zero penalty.

The agent can then update its non-dominated policies for the last action using the Q-sets of possible next states, and the average rewards (line 17). After these updates, the agent (re)starts a new trip (an episode for this particular agent), thus it is reinserted at its origin node.

The `else` statement in lines 19 - 29 deals with the agent that reached a node that is not its destination (or a dead-end). In this case, the reward computation in line 20 neither a bonus nor a penalty apply. Thus both b and p_d are set to zero.

Both methods in lines 17 and 22, as well as the Q-set computation in line 21 are performed as proposed in the original PQL algorithm.

4. Experiments and Results

4.1. Network and Demand

To test the approach, we used a 5×5 grid network depicted in Figure 1. The demand corresponding to the OD (origin-destination) matrix is shown in in Table 1. For each OD pair, the number of trips refers to both directions. For example, for the first line of the table, half of the vehicles travel from Bottom0 to Top4 and half from Top4 to Bottom0.

Each link shown in Figure 1 is two-way. The ticker links (in blue, connecting nodes A2 to E2) are the ones in which we want to have less emission. Thus, each agent using those links receive a penalty p_e (as discussed in Section 3.3) once it travels through them.

4.2. Methods and Parameters

The method proposed was compared against three others: dynamic user assignment (DUA), which is an iterative method for computing the user equilibrium provided by SUMO; the standard

QL optimizing only the agents' travel time (QLTT); and the standard QL optimizing only the CO emission (QLCO).

All parameters discussed in Section 3 were set as follows: maximum simulation steps $M = 50,000$; since we use normalized (ranging from $[-1, 0]$) values for the reward, all values regarding reward computation were set as $b = 1$, $p_d = 1$ and $p_e = 1$, guaranteeing that the rewards values received are always within $[-1, 1]$, as are the rewards. For all learning methods, the discount factor $\gamma = 0.9$ and a fixed value $\varepsilon = 0.05$ were set. Lastly, for the methods based on the standard QL (QLTT and QLCO), a learning rate $\alpha = 0.5$ was used. We remark that other values for these parameters were tried, without significant improvement.

Its is also worth mentioning that, due to the stochastic nature of all methods we used for comparison, each of them were repeated 30 times. In the plots ahead, we show the mean values (and their standard deviations, as shadows).

4.3. Results and Analysis

The comparison among the aforementioned methods was performed using values related to three different metrics: travel time (Figure 2a), CO emission (2b) and speed (2c). These values were collected from each link in the network. Plots in Figure 2 show a moving average (over a window of 600 steps) of the corresponding value, over all links.

In order to investigate performances for a part of the network only (those links indicated in blue in Figure 1), Figure 3 also depicts the three metrics, however values there are averaged only over those particular links (not for the entire network, as in Figure 2).

Results plotted in Figure 2 show a clear advantage of the aPQL method: it outperforms the others, showing lower travel time, lower CO emission, and higher speed (already after approximately 20,000 steps). These results seem to indicate that the aPQL method have a better usage of the links in the network. On a related note, one might assume that higher speeds would lead to higher CO emission. However, a more decisive factor is acceleration/deceleration. If the speed is constant, this seems to not affect emissions in a significant negative way.

Another issue is that one also observes a better distribution of vehicle in the network, which may be due to the fact that the aPQL takes the advantage of both the optimization of travel time and CO emission. The penalty imposed in the links A2 to E2 in Figure 1 causes both methods that optimize the objectives independently to behave in a different way. Optimizing travel time (QLTT) alone leads agents to a solution that can be seen in Figure 3(b), where no considerable reduction to CO emission is observed when we look at the red curve that represents QLTT. Conversely, optimizing only the CO emission causes the agents not to take the links in which they are penalized. In fact, since they completely stop selecting such links, it is possible to see that there are no data points for travel time and speed in Figure 3 regarding QLCO. One also notices that the average network travel time from QLCO (green curve in Figure 2(a)) is slightly higher than the one observed in QLTT. This happens due to the fact that, by avoiding several links within the network, the agents in QLCO have less available actions, and thus experience an increase in travel time.

By employing an approach that considers both objectives at the same time, aPQL makes the agents reach a compromise that seems to yield a better distribution of vehicles throughout the network: they mostly (but not entirely) avoid the links in which the CO emission is penalized

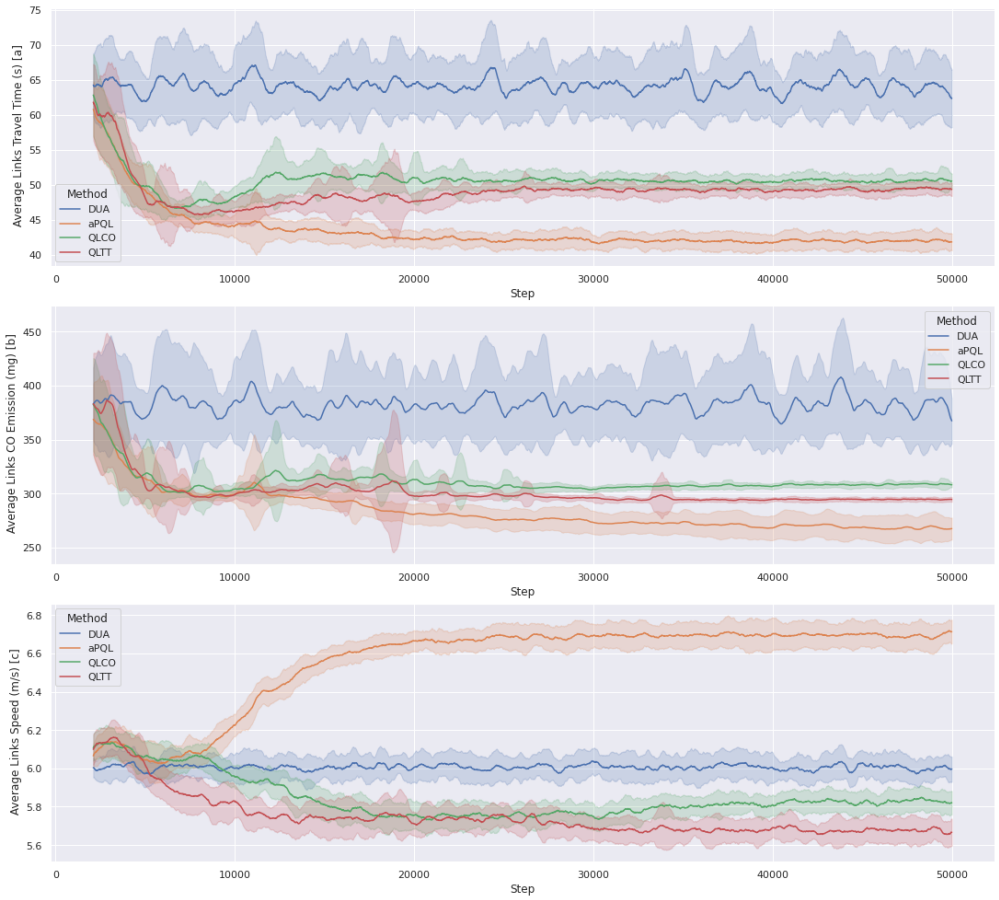

Figure 2: Full network: travel time, CO emission and speed.

(observed in the orange curves in Figure 3), using other parts of the network, taking advantage of the fact that those links are less occupied. This leads to a considerable increase in performance in relation to the other methods.

Lastly, it is worth noting that the values observed in travel time in the present paper are considerably lower than what was seen in our previous works, when the same grid networks was used ([13, 14]). This is due to the fact that here travel times are per link, while previously we have shown travel times over entire routes.

5. Conclusion and Future Work

Using RL to support agents making decisions that consider more than one objective poses challenges. We formulate the problem of multiple agents learning to travel from A to B in a traffic network as an RL task in which: (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 stochastic game, where states are intermediary locations in the

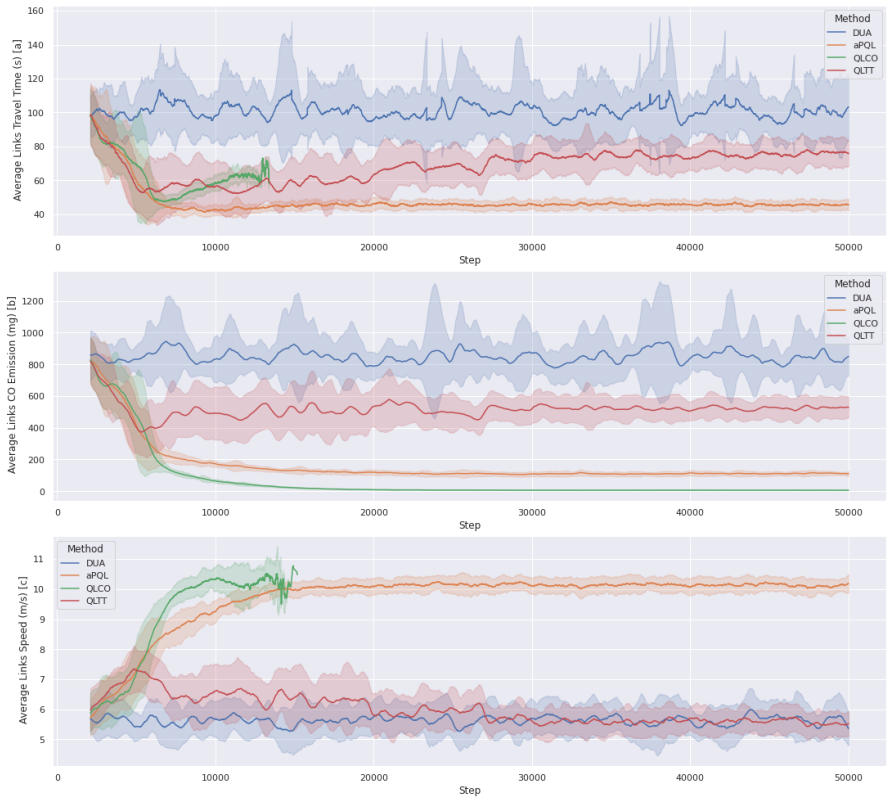

Figure 3: Horizontal links: travel time, CO emission and speed.

networks (such as intersections), in which decisions are made about which link to take.

To solve this task, we propose an adaptation in an existing algorithm, PQL [19]. This algorithm deals with more than one objective, but it was originally formulated for single-agent learning tasks, in which the environment is deterministic. In route choice neither of these apply. Thus our adaptation addresses them. We have applied the proposed method to a scenario in which hundreds of agents compete for a scarce resource (link usage), and have to learn how to travel from their origins to their destinations, aiming at minimizing their travel times, as well as the CO vehicles emit. Results show that the adapted algorithm is able to find routes that are more efficient then when either only travel time or only emissions are considered as rewards.

Future work is planned in the direction of considering communication between vehicles and infraestructure as in [14], given that this work was the basis of the present one; thus a revisiting of that case (now considering various objectives) is an open direction. In this regard, we also plan to include more objectives, such as other types of emissions.

Acknowledgments

This work is partially supported by FAPESP and MCTI/CGI (grants number 2020/05165-1 and 2021/05093-3). Ana Bazzan is partially supported by CNPq under grant number 304932/2021-3,

and by the German Federal Ministry of Education and Research (BMBF), Käte Hamburger Kolleg Cultures des Forschens/ Cultures of Research.

References

[1] A. Raith, J. Y. Wang, M. Ehrgott, S. A. Mitchell, Solving multi-objective traffic assignment, Annals of Operations Research 222 (2014) 483–516.

[2] J. Y. Wang, M. Ehrgott, Modelling route choice behaviour in a tolled road network with a time surplus maximisation bi-objective user equilibrium model, Transportation Research Part B: Methodological 57 (2013) 342–360. doi:10.1016/j.trb.2013.05.011.

[3] J. Y. T. Wang, A. Raith, M. Ehrgott, Tolling analysis with bi-objective traffic assignment, in: M. Ehrgott, B. Naujoks, T. J. Stewart, J. Wallenius (Eds.), Multiple Criteria Decision Making for Sustainable Energy and Transportation Systems, 2010, pp. 117–129.

[4] C. A. Huanca-Anquise, Multi-objective reinforcement learning methods for action selection: dealing with multiple objectives and non-stationarity, Master's thesis, Instituto de Informática, UFRGS, Porto Alegre, Brazil, 2021. URL: http://hdl.handle.net/10183/231836.

[5] P. Auer, N. Cesa-Bianchi, P. Fischer, Finite-time analysis of the multiarmed bandit problem, Machine Learning 47 (2002) 235–256. doi:10.1023/A:1013689704352.

[6] C. Watkins, Learning from Delayed Rewards, Ph.D. thesis, University of Cambridge, 1989.

[7] J. d. D. Ortúzar, L. G. Willumsen, Modelling transport, 4 ed., John Wiley & Sons, Chichester, UK, 2011.

[8] R. B. Dial, A model and algorithm for multicriteria route-mode choice, Transportation Research Part B: Methodological 13 (1979) 311–316. doi:https://doi.org/10.1016/0191-2615(79)90024-9.

[9] A. L. C. Bazzan, F. Klügl, Introduction to Intelligent Systems in Traffic and Transportation, volume 7 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*, Morgan and Claypool, 2013. doi:10.2200/S00553ED1V01Y201312AIM025.

[10] G. de. O. Ramos, B. C. da Silva, A. L. C. Bazzan, Learning to minimise regret in route choice, in: S. Das, E. Durfee, K. Larson, M. Winikoff (Eds.), Proc. of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), IFAAMAS, São Paulo, 2017, pp. 846–855. URL: http://ifaamas.org/Proceedings/aamas2017/pdfs/p846.pdf.

[11] T. B. F. de Oliveira, A. L. C. Bazzan, B. C. da Silva, R. Grunitzki, 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, Rio de Janeiro, 2018, pp. 1–8. URL: https://doi.org/10.1109/IJCNN.2018.8489655.

[12] R. Grunitzki, A. L. C. Bazzan, Combining car-to-infrastructure communication and multi-agent reinforcement learning in route choice, in: A. L. C. Bazzan, F. Klügl, S. Ossowski, G. Vizzari (Eds.), Proceedings of the Ninth Workshop on Agents in Traffic and Transportation (ATT-2016), volume 1678 of *CEUR Workshop Proceedings*, CEUR-WS.org, New York, 2016. URL: http://ceur-ws.org/Vol-1678/paper12.pdf.

[13] G. D. dos. Santos, A. L. C. Bazzan, Accelerating learning of route choices with C2I: A preliminary investigation, in: Proc. of the VIII Symposium on Knowledge Discovery, Mining and Learning, SBC, 2020, pp. 41–48. doi:10.5753/kdmile.2020.11957.

[14] G. D. dos. Santos, A. L. C. Bazzan, Sharing diverse information gets driver agents to learn faster: an application in en route trip building, PeerJ Computer Science 7 (2021) e428. URL: https://doi.org/10.7717/peerj-cs.428. doi:10.7717/peerj-cs.428.

[15] G. Dytz dos Santos, A. L. C. Bazzan, A. P. Baumgardt, Using car to infrastructure communication to accelerate learning in route choice, Journal of Information and Data Management 12 (2021). URL: https://sol.sbc.org.br/journals/index.php/jidm/article/view/1935.

[16] A. L. C. Bazzan, R. Grunitzki, A multiagent reinforcement learning approach to en-route trip building, in: 2016 International Joint Conference on Neural Networks (IJCNN), 2016, pp. 5288–5295. doi:10.1109/IJCNN.2016.7727899.

[17] R. Rădulescu, P. Mannion, D. Roijers, A. Nowé, Multi-objective multi-agent decision making: a utility-based analysis and survey, Autonomous Agents and Multi-Agent Systems 34 (2020). doi:10.1007/s10458-019-09433-x.

[18] C. F. Hayes, R. Radulescu, E. Bargiacchi, J. Källström, M. Macfarlane, M. Reymond, T. Verstraeten, L. M. Zintgraf, R. Dazeley, F. Heintz, E. Howley, A. A. Irissappane, P. Mannion, A. Nowé, G. de Oliveira Ramos, M. Restelli, P. Vamplew, D. M. Roijers, A practical guide to multi-objective reinforcement learning and planning, CoRR abs/2103.09568 (2021). URL: https://arxiv.org/abs/2103.09568. arXiv:2103.09568.

[19] K. Van Moffaert, A. Nowé, Multi-objective reinforcement learning using sets of pareto dominating policies, J. Mach. Learn. Res. 15 (2014) 3483–3512.