Computer Science and Information Systems 21(1):335–362 https://doi.org/10.2298/CSIS221210071A

Knowledge Transfer in Multi-Objective Multi-Agent Reinforcement Learning via Generalized Policy Improvement

Vicente N. de Almeida, Lucas N. Alegre, and Ana L. C. Bazzan

Instituto de Informática, Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre, RS, Brazil
{vnalmeida,lnalegre,bazzan}@inf.ufrgs.br

Abstract. Even though many real-world problems are inherently distributed and multi-objective, most of the reinforcement learning (RL) literature deals with single agents and single objectives. While some of these problems can be solved using a single-agent single-objective RL solution (e.g., by specifying preferences over objectives), there are robustness issues, as well the fact that preferences may change over time, or it might not even be possible to set such preferences. Therefore, a need arises for a way to train multiple agents for any given preference distribution over the objectives. This work thus proposes a multi-objective multi-agent reinforcement learning (MOMARL) method in which agents build a shared set of policies during training, in a decentralized way, and then combine these policies using a generalization of policy improvement and policy evaluation (fundamental operations of RL algorithms) to generate effective behaviors for any possible preference distribution, without requiring any additional training. This method is applied to two different application scenarios: a multi-agent extension of a domain commonly used in the related literature, and traffic signal control, which is more complex, inherently distributed and multi-objective (the flow of both vehicles and pedestrians are considered). Results show that the approach is able to effectively and efficiently generate behaviors for the agents, given any preference over the objectives.

Keywords: reinforcement learning, multi-agent systems, multi-objective decision making, generalized policy improvement, traffic signal control.

1. Introduction

Reinforcement Learning (RL) [33] deals with agents that learn by acting in an environment and receiving rewards (numerical signals) that guide them toward selecting better actions. In recent years, RL has been successfully applied to complex tasks, such as healthcare [40], robotics [20], game playing [32, 24] and combinatorial optimization [23].

While multi-agent RL (MARL) is a natural framework to model problems that are inherently distributed, such formulation adds many challenges to RL. For instance, because multiple agents interact in a shared environment, their actions are usually highly coupled. This makes learning harder as agents try to adapt to other agents that are also learning. Besides, several convergence guarantees no longer hold when agents learn in a decentralized manner [9]. However, in many real-world problems, where the control is decentralized, it is not always possible, feasible or desirable to avoid a MARL formulation.

Besides considering multiple agents, RL has also been extended to deal with multiple objectives in the multi-objective RL (MORL) literature [17]. This is of fundamental practical importance, since many real-world problems are inherently multi-objective (e.g. in traffic signal control we may need to trade-off the waiting time of the vehicles and the pedestrians). There are two main ways to formulate a MORL problem: using separate reward functions for each objective, or using a single scalar reward that combines the agent's preferences with the values received for each objective, thus reducing a MORL problem into a single-objective RL problem. While the literature argues for both these formulations, in [29] many scenarios are presented to illustrate the need for having methods that deal with separate reward functions to solve MORL problems. For instance, the user may not know a priori what is the more appropriate trade-off among the objectives, or the underlying user preferences may change over time. To deal with such settings, the goal is to construct a set of policies such that agents can perform well, given *any* preferences or ways of combining the objectives. In this paper, we make the assumption that the preferences over objectives are expressed as linear combinations of the reward functions.

Recently, Alegre et al. [3] introduced a method that tackles this problem and enables an RL agent to transfer knowledge from policies specialized to different objectives. Their method incrementally construct a set of policies that can later be combined to generate optimal policies for any preference among objectives. However, their proposed method only addresses single-agent scenarios. To fill this gap, we extend the method presented in [3] in order to consider multiple agents. Our approach builds a shared set of policies in a decentralized way during training, and then combine these policies to create new policies specialized to different preferences over the objectives. The proposed method is applied to two different environments: a multi-agent extension of the Four-Room environment (a domain commonly used in the related literature), and a complex, inherently distributed and multi-objective problem (traffic signal control, where objectives relate to the waiting time of both vehicles and pedestrians).

In summary, the present method works as follows. During the training phase, agents are iteratively given different preferences over their objectives, and learn policies based on these preferences. These policies are shared among all agents, which have the same observation and action spaces. During the execution phase (after the training is over), by making use of generalized policy evaluation (GPE) and generalized policy improvement (GPI), which are generalizations of two key operations of RL algorithms (policy evaluation and policy improvement), the agents are able to construct policies for any given preference over objectives. Therefore, this work proposes a transfer learning (TL) method for multi-objective multi-agent reinforcement learning (MOMARL). This helps fill a gap within the literature, since the vast majority of the RL literature focuses on single-agents with single-objectives, and only a handful of works address MOMARL settings.

The main contributions of this work can be summed up as follows:

- This is the first work (to the authors' best knowledge) that leverages generalized policy improvement for settings with multiple agents and objectives.
- We empirically evaluate our method in a multi-objective multi-agent traffic signal control environment with traffic controllers optimizing for vehicles' and pedestrians' objectives, which has rarely been addressed in the traffic signal control literature.

The reader can find a discussion on the main underlying concepts behind this work and a review of the related literature in Section 2 and in Section 3, respectively. In Sec-

tion 4, the proposed method is presented and explained in details. Section 5 presents the experimental settings and discusses the results. Finally, Section 6 concludes this work.

2. Theoretical Background

This section presents underlying concepts on RL, MARL, and MORL, as well as on specific concepts such as Successor Features (SFs), Generalized Policy Evaluation (GPE) and Generalized Policy Improvement (GPI).

2.1. Reinforcement Learning

An RL problem can be formulated as a Markov Decision Process (MDP), which is a mathematical model for sequential decision making. An MDP can be formally defined as a tuple of the form $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a state transition function mapping state transitions caused by actions to probabilities, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a reward function, and $\gamma \in [0, 1)$ is the discount factor. At each time step t, an agent observes a state S_t, selects and takes an action A_t, receives a reward R_{t+1} and transitions to state S_{t+1} according to the state transition function. A policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ defines how the agent behaves in the environment by mapping states in $\mathcal{S}$ to actions in $\mathcal{A}$.

As aforementioned, agents aim to maximize the reward received. However, since RL deals with sequential decision making, at time step t, the best decision may not necessarily be to select an action that will probably lead to the greatest next reward. As the real objective of an RL agent is to maximize the expected discounted sum of all rewards received by the agent, a discount factor γ is used to discount the impact of rewards received in later time steps, thus determining the present value of a future reward.

The state-value function of a state s under a policy π, which is denoted by $V^\pi(s)$, is the expected value of all returns received by an agent that starts in state s and follows the policy π. This function is given by Eq. 1.

$$V^\pi(s) = \mathbb{E}^\pi \left[\sum_{i=0}^{\infty} \gamma^i R_{t+i+1} \middle| S_t = s \right]. \tag{1}$$

Similarly, the action-value function for a policy π, which is the expected return of taking action a in state s and then following the policy π, is denoted by $Q^\pi(s, a)$ and can be defined by the Bellman equation in Eq. 2.

$$Q^\pi(s, a) = \mathbb{E}^\pi \left[R_{t+1} + \gamma Q^\pi(S_{t+1}, \pi(S_{t+1})) \middle| S_t = s, A_t = a \right]. \tag{2}$$

A policy π is better than or equal to a policy π' if and only if the state-value function of any state s under policy π is greater than or equal to the state-value function for the same state under policy π'. This relation, shown in Eq. 3, induces a complete ordering over policies.

$$\pi \succeq \pi' \iff \forall s, V^\pi(s) \geq V^{\pi'}(s). \tag{3}$$

If a policy is better than or equal to all others, it is an optimal policy, denoted by π^*. It is important to note that there is always at least one optimal policy. Therefore, solving

a problem in RL means finding an optimal policy (or at least a policy that sufficiently approximates an optimal policy).

All optimal policies share the same state-value function (which is the optimal state-value function, denoted V^*) and the same action-value function (which is the optimal action-value function, denoted Q^*). Importantly, if the optimal action-value function is known, an optimal policy π^* can be defined as in Eq. 4.

$$\pi^*(s) \in \arg\max_{a \in \mathcal{A}} Q^*(s, a). \tag{4}$$

An important subset of RL algorithms are based on Dynamic Programming (DP) [7]. These methods convert Bellman equations into update rules [33], and make use of mathematical properties of MDPs to decrease the complexity of searching for optimal policies. RL methods based on DP rely on two fundamental operations: policy evaluation and policy improvement. Policy evaluation is the computation of the value function of a policy π, and policy improvement refers to finding a policy π' that is better than π.

2.2. Multi-Agent Reinforcement Learning

In order to model multi-agent decision making problems, MDPs are extended to Multi-Agent Systems (MAS) as Stochastic Games (SG) [30], which can be formally defined as a tuple $\langle n, \mathcal{S}, \mathcal{A}_{1..n}, \mathcal{T}, \mathcal{R}_{1..n}, \gamma \rangle$, where n is the number of agents, $\mathcal{S}$ is the state space, $\mathcal{A}_i$ is the set of actions of agent i ($\mathcal{A} = \mathcal{A}_1 \times ... \times \mathcal{A}_n$ is the joint action space), $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a state transition function, $\mathcal{R}_i$ is the reward function of agent i, and $\gamma \in [0, 1)$ is the discount factor.

Differently than single-agent RL, in MARL each reward received by an agent depends not only on its own actions, but on the actions of other agents. Thus, as previously mentioned, several convergence guarantees that are true in single-agent RL no longer hold.

Not only do other agent's actions influence the reward received by each agent, but in fact agents may have opposing objectives, and the action of one agent might negatively impact the rewards received by another. Generally, the nature of a multi-agent task is classified as being either cooperative, competitive or mixed [9]. In MARL, agents can be trained in a centralized manner using, for instance, a shared global reward. However, this approach does not scale due to the curse of dimensionality, as well as because agents have difficulty understanding and distinguishing their own contribution to the larger system [26]. Another option is full decentralization, that is, agents learn and act individually, in a fully decentralized manner. A third option, is a middle ground between centralized and fully decentralized, in which agents learn and act in a decentralized manner, but employ knowledge transfer strategies in order to accelerate learning [31]. In this work, we follow this approach by allowing agents to share their learned policies, which are further combined via generalized policy improvement (Section 4.2).

2.3. Multi-Objective Reinforcement Learning

One common approach to enable agents to deal with multiple objectives is to combine all of the objectives into a single scalar reward function. Unfortunately, this rather simple solution has many drawbacks, as for instance it reduces drastically the explainability of

the model (as it will be hard to distinguish which particular objectives prompted a specific behavior), it is unable to handle many different types of preferences that might be required by a specific problem, and will require that the agents be retrained if preferences among objectives change [17].

With these drawbacks in mind, it becomes clear why, in order to deal with multiple objectives, an explicitly multi-objective approach is preferable over a scalarized reward approach (which is equivalent to a single-objective approach). Hence, instead of using a scalar reward, a vector-valued reward function will be used.

To formally describe a multi-objective decision making problem, MDPs are extended to the multiple objective setting as a Multi-Objective Markov Decision Process (MOMDP). An MOMDP only differs from an MDP in the reward function, which becomes a vector-valued function $\boldsymbol{\mathcal{R}} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}^d$ that specifies the immediate reward for each of the d objectives considered.

Since rewards are now vector-valued, state-value functions and action-value functions are now vector-valued as well, that is, $\boldsymbol{V}^{\pi} \in \mathbb{R}^d$ and $\boldsymbol{Q}^{\pi} \in \mathbb{R}^d$. For problems with single objectives, a policy π is either better, equal or worse than policy π', as value functions completely order all policies. This is not necessarily the case in multi-objective decision making problems, as value functions under policy π, when compared to value functions under policy π', might have a greater value for some objectives, but a smaller value for others. Because there is no complete ordering over policies, in MOMDPs value functions only offer a partial ordering over the policy space. Hence, in this setting, there can exist several possibly optimal value vectors $\boldsymbol{V}$ and $\boldsymbol{Q}$. In order to deal with this, a few sets of possibly optimal policies and value vectors need to be defined.

Before the sets aforementioned are defined, the utility function (or scalarization function) $u : \mathbb{R}^d \rightarrow \mathbb{R}$ needs to be introduced. This function, shown in Eq. 5, maps a multi-objective state-value vector under a policy π to a scalar value. This function commonly takes the form of a linear combination.

$$V_u^{\pi}(s) = u(\boldsymbol{V}^{\pi}(s)). \tag{5}$$

According to a MOMDP problem taxonomy proposed by [29], the three major factors that classify the nature of an MOMDP problem are: (i) whether the goal is to find one or multiple policies; (ii) whether the utility function is a linear function or, more generally, any monotonically increasing function; (iii) whether stochastic policies (instead of only deterministic policies) are allowed. For the sake of the present discussion, more definitions will be given regarding factor (ii).

In the more general case, where the utility function is simply monotonically increasing, a set of viable policies commonly used in the literature is the Pareto front PF(Π). A policy π Pareto-dominates another policy π' if its value is equal or greater for every objective, and strictly greater for at least one objective, according to Eq. 6. The Pareto front set is defined in Eq. 7. It is important to note that the Pareto front set is not necessarily the undominated set, as the Pareto front may be larger than the undominated set, depending on the utility function.

$$\boldsymbol{V}^{\pi} \succ_P \boldsymbol{V}^{\pi'} \iff \forall i, V_i^{\pi} \geq V_i^{\pi'} \wedge \exists i, V_i^{\pi} > V_i^{\pi'}. \tag{6}$$

$$PF(\Pi) = \{\pi \in \Pi \mid (\nexists \pi' \in \Pi)\, [\boldsymbol{V}^{\pi'} \succ_P \boldsymbol{V}^{\pi}]\}. \tag{7}$$

A linear utility function, shown in Eq. 8, is the inner product of a value vector $\boldsymbol{V}^{\pi}$ and a vector of weights $\boldsymbol{w}$ which adheres to the simplex constraints (that is, $\forall i \;\; w_i \geq 0$ and $\sum_i w_i = 1$) [28].

$$V^{\pi}_{\boldsymbol{w}} = \boldsymbol{w} \cdot \boldsymbol{V}^{\pi}. \tag{8}$$

If the utility function is linear, the undominated set of policies is a convex hull CH(Π), defined in Eq. 9. Since this convex hull is an undominated set, it may contain policies in excess. Thus, a coverage set for the convex hull can be defined. A set CCS(Π) is a *convex coverage set* [17] if it is a subset of CH(Π) and if for every $\boldsymbol{w}$ it contains a policy whose linearly scalarized value is maximal, as defined in Eq. 10.

$$CH(\Pi) = \{\pi \in \Pi \mid \exists \boldsymbol{w}, \forall \pi' \in \Pi \; : \boldsymbol{w} \cdot \boldsymbol{V}^{\pi} \geq \boldsymbol{w} \cdot \boldsymbol{V}^{\pi'}\}. \tag{9}$$

$$CCS(\Pi) \subseteq CH(\Pi) \wedge \left(\forall \boldsymbol{w}, \exists \pi \in CCS(\Pi), \forall \pi' \in \Pi : \boldsymbol{w} \cdot \boldsymbol{V}^{\pi} \geq \boldsymbol{w} \cdot \boldsymbol{V}^{\pi'}\right). \tag{10}$$

All of the sets defined in this Section are of extreme importance to multi-objective problems, because MORL algorithms seek to compute the policies contained in these sets in order to solve multi-objective decision-making problems. In this paper, we consider the linear utility case, and thus we tackle the problem of constructing an approximate CCS. In this case, we are inherently unable to identify solutions that lie on the concave region of the Pareto front [35].

2.4. Successor Features and Generalized Policy Improvement

In [5], it is argued that many complex problems tackled by RL can be broken down into multiple tasks, encoded by different reward functions. A reward function $\mathcal{R}_{\boldsymbol{w}}$ for a task is defined as in Eq. 11, where $\boldsymbol{\phi}(s, a, s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}^d$ is an arbitrary function representing the features of the environment and $\boldsymbol{w} \in \mathbb{R}^d$ is a vector of weights.

$$\mathcal{R}_{\boldsymbol{w}}(s, a, s') = \boldsymbol{w} \cdot \boldsymbol{\phi}(s, a, s'). \tag{11}$$

With this reward definition, given a policy π, the action-value function shown in Eq. 2 can be rewritten as in Eq. 12, where $\boldsymbol{\psi}^{\pi}(s, a)$ are successor features (SFs) [4]. It is important to note that SFs satisfy a Bellman equation, so they can be computed using conventional RL algorithms.

$$\begin{aligned} Q^{\pi}_{\boldsymbol{w}}(s, a) &= \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k \boldsymbol{w} \cdot \boldsymbol{\phi}_{t+k} \middle| S_t = s, A_t = a, \pi\right] \\ &= \boldsymbol{w} \cdot \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k \boldsymbol{\phi}_{t+k} \middle| S_t = s, A_t = a, \pi\right] \\ &= \boldsymbol{w} \cdot \boldsymbol{\psi}^{\pi}(s, a). \end{aligned} \tag{12}$$

The definitions given above, combined with a generalization of two fundamental operations of DP methods for RL (policy evaluation and policy improvement, mentioned in Section 2.1), provide a framework for knowledge transfer among tasks.

Suppose a RL agent, using an arbitrary RL method, has, through training, learned policies for k tasks (given by k different instances of $\boldsymbol{w}$). The agent thus knows a set of policies $\Pi = \{\pi_i\}_{i=1}^{k}$ and a set of corresponding SFs $\Psi = \{\boldsymbol{\psi}^{\pi_i}\}_{i=1}^{k}$.

Using Eq. 12, it is possible to compute the value-function of a policy π on the k tasks by simply computing a dot-product between the SFs, $\boldsymbol{\psi}^{\pi}$, and each weight vector $\boldsymbol{w}$. This is called generalized policy evaluation (GPE). Also, given a new task, generalized policy improvement (GPI) refers to finding a policy that is better than all of the policies in Π. Note that GPE and GPI generalize policy evaluation and policy improvement to multiple policies and tasks, so if $k = 1$, GPE and GPI are equivalent to the standard DP operations.

Clearly, there is a strong similarity between the SF framework and MORL problems encoded by separate reward functions. In fact, in [3] it is shown that the transfer problem addressed by SFs is equivalent to the multi-objective optimization in MORL. That work demonstrates this by showing that it is possible to map any SF problem to a MORL problem by converting each dimension of $\boldsymbol{\phi}$ into an objective. That is, an MOMDP is created with $\boldsymbol{\mathcal{R}}(s, a, s') = \boldsymbol{\phi}(s, a, s')$. Importantly, in this case the definitions of SFs and multi-objective action-value function become equivalent, that is, $\boldsymbol{Q}^{\pi} = \boldsymbol{\psi}^{\pi}$.

Since there exists this equivalence between SF problems and MORL problems, it is possible to apply GPI to MORL. Furthermore, [3] also shows that, if a convex coverage set is learned, performing GPI on this set enables an RL agent to learn an optimal policy for any weight vector $\boldsymbol{w}$.

3. Related Work

This section presents a review of the related literature. Section 3.1 and Section 3.2 address relevant works that have tackled SFs and GPI in MORL and MARL settings, respectively, and Section 3.3 discusses applications of RL for traffic signal control.

3.1. SFs and GPI in MORL

SFs and GPI are presented in [4] as a generalization of the concept of successor representation [11] and policy improvement, respectively. This provides a framework for reusing knowledge across RL tasks, which is further elaborated in [5].

By formally demonstrating the equivalence between the SF framework and MORL, [3] is able to make use of GPI within the context of MORL, thus proposing a method that is able to combine previously learned policies in order to compute an optimal policy for any preference of objectives. However, this work only deals with single-agent scenarios.

Leveraging GPI within MORL is achieved by first using SFOLS, a SFs based extension of Optimistic Linear Support (OLS) [28] to assign different weights to the RL agent, so that it iteratively computes a CCS (Eq. 10). Then, once a CCS is learned, applying GPI to this set enables the agent to generate an optimal policy for any possible preference of objectives.

Most of the works discussed in this section use as one of their experimental settings the Four-Room domain, which is an environment made up of four rooms (separated by walls) in which a single agent moves around and collects different types of objects, which relate to different objectives. Because this environment is such a common benchmark in the literature, this work extends the Four-Room environment to multiple agents, and uses this domain as one of the experimental settings.

3.2. SFs and GPI in MARL

Besides the works discussed in Section 3.1, only a handful of works have mentioned SFs in the context of MORL, such as [17] (which briefly states that SFs are a subclass of multi-objective decision making problems), and [1] (which also briefly mentions that SFs and MORL are analogous). However, there have been a few works that have extended the concepts of SFs and GPI to multi-agent scenarios.

Decomposing a problem using a multi-agent solution generally introduces a source of non-stationarity. One approach that has been used to tackle this is the centralized training with decentralized execution (CTDE) framework, in which the agents learn their policies together, but execute them independently.

This CTDE method is used in the universal value exploration (UNeVEn) algorithm, presented in [16], which extends universal successor features (USFs) [8] to multi-agent universal successor features (MAUSFs), and combines this with GPI to improve the joint exploration of agents during training. Furthermore, CTDE is also used in [21], which presents an approach that makes use of SFs and GPI, and enables knowledge transfer among tasks in MARL settings.

Another relevant work that tackles SFs using CTDE is [19]. They use SFs to isolate each agent's impact on the performance of the entire system, which allows the method to create individual utility functions for each agent tailored to stabilize their training.

It is important to note that, since the works addressing SFs and GPI in MARL settings do so using the CTDE framework, they all suffer from scalability issues, for obvious reasons. The more agents in the environment, the more difficult it is to train these agents in a centralized manner. The method proposed in this work has no such scalability issues, as both execution and training occur in a decentralized fashion.

3.3. RL for Traffic Signal Control

Traffic signal controllers seek to determine a split of green times among various phases at an intersection. A phase is defined as a group of non-conflicting movements (e.g., flow in two opposite traffic directions) that can have a green light at the same time without conflict. There are many different ways that traffic controllers can go about making decisions, and [27] provides an overview of several of them.

The most basic form of traffic control is based on fixed times, where the split of green times among the phases is computed using historical data on traffic flow, if available. However, this approach is unable to adapt to changes in traffic demand, which may lead to an increase in waiting times. To mitigate this issue, an adaptive approach, such as RL, can be used to determine the split of green times using some measure of performance (for instance, accumulated waiting time).

The RL literature for traffic signal control is very extensive and diverse, and a detailed overview of different techniques is beyond the scope of this work. Thus, the reader is directed to surveys such as [6, 38, 36, 25].

Besides reducing the waiting time of vehicles, traffic signal controllers can also have different objectives, such as reducing queue lengths [12] and reducing the environmental impact of traffic by minimzing fuel consumption [18]. However, it must be emphasized that the literature on MORL for traffic signal control is small, and there are very few MORL works for signal control that address pedestrians, such as [13] and [39]. This is another literature gap that this work helps to fill.

4. Policy Transfer in MOMARL using GPI

This section describes the proposed method in detail. However, before the method is explained, some preliminary conditions need to be set forth. First, all agents must be homogeneous (that is, have the same sensors, possible actions, and objectives). We also assume that their utility functions are represented as an inner product between the objectives and the weights (Eq. 8). Finally, each agent can have its own individual weights $\boldsymbol{w}$.

During the training phase, the method uses a multi-agent extension of the SFOLS algorithm [3] to distributively solve scalarized versions of the multi-objective problem, by assigning different weights to each agent. Each agent computes a policy that is stored in a set of policies shared by all agents, until they have computed a convex coverage set. This process is explained in Section 4.1.

Once the agents have decentrally computed a CCS, the training phase is over. Then, during the execution phase, each agent is able to compute an effective policy for any possible instance of $\boldsymbol{w}$, by applying GPI [3] to the shared set. This is explained in Section 4.2. A pseudocode for the algorithm is shown in Algorithm 1.

4.1. Decentrally computing a shared set of policies

During training, the agents seek to distributively compute a set of policies, Π, in order to approximate a CCS. To do so, each agent $\boldsymbol{a}$ iteratively receives a different weight vector, $\boldsymbol{w_a}$, and learns a policy, $\pi_{\boldsymbol{a}}$, specialized to the multi-objective task $\boldsymbol{w_a}$. Every time an agent receives a weight vector, it solves the task given the scalar reward function $\mathcal{R}_{\boldsymbol{w}}(s, a, s') = \boldsymbol{w} \cdot \boldsymbol{\mathcal{R}}(s, a, s')$ using a multi-objective Q-learning algorithm. The policies learned by the agents (and their corresponding SFs) are stored in a shared set of policies, Π. This process is repeated until Π corresponds to a CCS. The key, then, is to know which weights to assign to the agents, and when to stop (that is, how to know when Π is a CCS). This is determined by using the OLS algorithm, which is an extension of Cheng's linear support algorithm [10].

Let Π be a set of policies shared among all agents and P a priority queue of weight vectors. At each iteration of the algorithm, the weights in P with greater priority are popped off the queue and assigned to the agents. If there are more agents than weights in P, then randomly sampled weights are assigned to the remaining agents. After each iteration, new weights are added, until Π corresponds to a CCS. If, at the end of an iteration, P is empty, that means that Π is a CCS, and the algorithm ends.

The first step is to add to P all weights in the extremum of the weight simplex (that is, all weights that have one component equal to 1 and all others equal to zero), and assign to these weights an infinite priority to ensure that they are the first weights to be trained on. After this initialization of the priority queue, the algorithm's main loop begins. The weights in P are popped off according to their priorities and assigned to the agents, which then proceed to learn policies for the scalarized version of their multi-objective task (using their respective weights to scalarize the task).

Once each agent $\boldsymbol{a}$ finishes learning its respective policy $\pi_{\boldsymbol{a}}$ (and corresponding SF $\boldsymbol{\psi}^{\pi_{\boldsymbol{a}}}$), the algorithm evaluates the multi-objective value, $\boldsymbol{V}^{\pi_{\boldsymbol{a}}}$, of each learned policy.

[1] Then, each policy (and SF) is added to the shared set Π, if it corresponds to a still unknown value vector (lines 20–22 of Algorithm 1).

Lastly, the algorithm determines the *corner weights*, which, as per Theorem 1, are the instances of $\boldsymbol{w}$ that can provide the maximal improvement to the set of policies. Intuitively, corner weights are the instances of $\boldsymbol{w}$ whose optimal scalarized value functions are farthest from the current known best value vector for that $\boldsymbol{w}$.

Theorem 1. *[10] Let $\mathcal{W}$ be the set of all possible reward vectors, let CCS be a convex coverage set, and let Π be a set of deterministic policies. The maximum value of*

$$\max_{\boldsymbol{w}\in\mathcal{W},\pi\in CCS} \min_{\pi'\in\Pi} \boldsymbol{w}\cdot\boldsymbol{V}^{\pi} - \boldsymbol{w}\cdot\boldsymbol{V}^{\pi'} \tag{13}$$

is at one of the corner weights of

$$V^{\mathrm{CB}}_{\boldsymbol{w}} = \max_{\pi\in\Pi} \boldsymbol{w}\cdot\boldsymbol{V}^{\pi} \tag{14}$$

Let $V^{\mathrm{CB}}_{\boldsymbol{w}}$, given by Eq. 14, be the current best value function. To determine the *corner weights*, the algorithm exploits the fact that $V^{\mathrm{CB}}_{\boldsymbol{w}}$ is a piecewise linear and convex (PWLC) [28, 10] function. Thus, the corner weights are the points in which $V^{\mathrm{CB}}_{\boldsymbol{w}}$ changes slope.

Each corner weight $\boldsymbol{w}_c$ is inserted into P with priority $\Delta(\boldsymbol{w}_c)$ (line 27), which is an optimistic estimate of the greatest possible improvement caused by learning a policy specialized to $\boldsymbol{w}_c$. This priority is given by Eq. 15, where $\overline{V}^{*}_{\boldsymbol{w}}$ is an optimistic upper-bound for the optimal value function $V^{*}_{\boldsymbol{w}}$.

$$\Delta(\boldsymbol{w}_c) = \overline{V}^{*}_{\boldsymbol{w}} - V^{\mathrm{CB}}_{\boldsymbol{w}} \tag{15}$$

After inserting the corner weights in P, the algorithm's main loop (pop off weights with greatest priority from P, assign them to the agents, learn policies, add corner weights to P) is repeated, until P is empty, which indicates that there are no more corner weights, and therefore Π forms a CCS.

Let us propose an example to better illustrate the method. For the sake of simplicity, suppose a single agent (extending this example to multiple agents is trivial, as the only difference would be that more than one policy would be learned at each iteration) with three actions (A, B and C) and two objectives. Suppose there are 5 states: s_0, s_1, s_2, s_3 and s_4. s_0 is the initial state and s_4 is the terminal state.

For this simple example, at each time step, the agent always transitions to the same states, regardless of the action selected. Choosing different actions in each state only changes the reward received. The MOMDP for this example is shown in Figure 1. A run of the OLS algorithm for the example is shown graphically in Figure 2.

Since this example has only two objectives, and the weights adhere to the simplex constraints, only one of the dimensions of $\boldsymbol{w}$ is sufficient to express all possible instances of weights ($w_0 = 1 - w_1$).

First, the method inserts into P the weight vectors [1, 0] and [0, 1], the weights in the extrema of the weight simplex, with infinite priority. $\boldsymbol{w}$ = [1, 0] is popped off P, and the best policy for this weight is computed. This is shown in Figure 2a.

[1] Regular policy evaluation for single-objective RL can be used to determine the value of a policy for each of its objectives.

Algorithm 1: Multi-Agent SFOLS (MA-SFOLS)

$\Pi \leftarrow \{\}$; $\mathcal{V} \leftarrow \{\}$; $W \leftarrow \{\}$; $P \leftarrow \{\}$;
foreach *weight* $\boldsymbol{w}_e$ *in extremum of weight simplex* **do**
    Insert ($\boldsymbol{w}_e$, ∞) into P;
end
while P *is not empty* **do**
    foreach *agent* $\boldsymbol{a}$ **do**
        if P *is not empty* **then**
            $\boldsymbol{w_a} \leftarrow$ pop weight with highest priority from P;
        else
            $\boldsymbol{w_a} \leftarrow$ random weight;
        end
        Insert $\boldsymbol{w_a}$ into W;
        Assign $\boldsymbol{w_a}$ to agent $\boldsymbol{a}$;
    end
    Wait until agents learn policies for their current weights;
    foreach *agent* $\boldsymbol{a}$ **do**
        $\pi_{\boldsymbol{a}}, \psi_{\boldsymbol{a}} \leftarrow$ last policy and SF computed by agent $\boldsymbol{a}$;
        $\boldsymbol{V}^{\pi_{\boldsymbol{a}}} \leftarrow$ value vector computed by agent $\boldsymbol{a}$;
        $\boldsymbol{w_a} \leftarrow$ current weight of agent $\boldsymbol{a}$;
        if $\boldsymbol{V}^{\pi_{\boldsymbol{a}}} \notin \mathcal{V}$ **then**
            Insert $\boldsymbol{V}^{\pi_{\boldsymbol{a}}}$ into $\mathcal{V}$;
            Insert $(\pi_{\boldsymbol{a}}, \psi_{\boldsymbol{a}})$ into Π;
            Remove obsolete corner weights from P;
            $W_c \leftarrow$ getCornerWeights($\boldsymbol{V}^{\pi_{\boldsymbol{a}}}, \boldsymbol{w_a}, \mathcal{V}$);
            foreach $\boldsymbol{w} \in W_c$ **do**
                $\Delta(\boldsymbol{w}) \leftarrow$ getImprovementEstimate($\boldsymbol{w}, \mathcal{V}, W$);
                Insert ($\boldsymbol{w}$, $\Delta(\boldsymbol{w})$) into P;
            end
        end
    end
end
return Π, Ψ

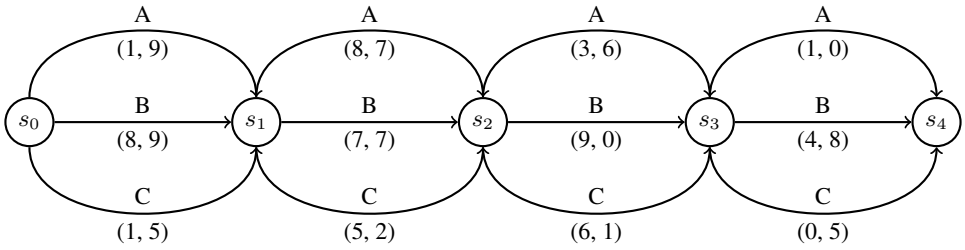


Fig. 1. Example: a simple MOMDP

Then, $\boldsymbol{w}$ = [0, 1] is popped off P, and the best policy for this weight is computed. The corner weight $\boldsymbol{w}$ = [0.32, 0.68] is found, with greatest possible improvement of $\Delta(\boldsymbol{w})$. This corner point is inserted in P, with a priority of $\Delta(\boldsymbol{w})$. This is shown in Figure 2b.

Next, $\boldsymbol{w}$ = [0.32, 0.68] is popped off P, and the best policy for this weight is computed. The corner weight $\boldsymbol{w}$ = [0.5, 0.5] is found, with greatest possible improvement of $\Delta(\boldsymbol{w})$. This corner point is inserted in P, with a priority of $\Delta(\boldsymbol{w})$. This is shown in Figure 2c.

Finally, $\boldsymbol{w}$ = [0.5, 0.5] is popped off P, and the best policy for this weight is computed. Notice that the best policy for this weight is the same as the best policy for $\boldsymbol{w}$ = [0.32, 0.68], so both curves overlap. No new corner point is found, and P is empty. The policies computed form a CCS, and the procedure ends. This is shown in Figure 2d.

4.2. Computing behaviors for new preferences using GPI

In Section 4.1, it is shown how the method makes use of a multi-agent extension of the OLS algorithm to decentrally compute a shared set of policies, and their respective successor features, which corresponds to a CCS. This section shows how the agents can create new policies for any possible weights using the policies in this shared set, thus transferring knowledge acquired from the source tasks (scalarized versions of the multi-objective problem using the weights in P) to the target tasks (scalarized versions of the multi-objective problem using new weights given to the agents during execution).

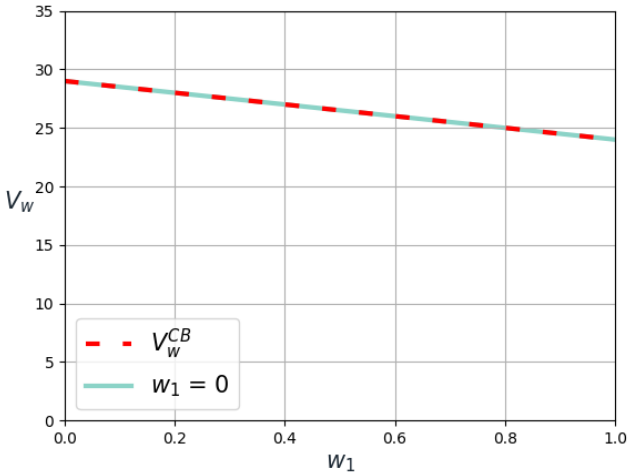
It is important to note that agents reuse knowledge acquired from policies learned by themselves and by others. This constitutes a type of multi-agent TL method, henceforth referred to as policy transfer (since previous policies are used to create new policies, both the source and target of the transfer are policies, thus policy transfer).

Policy improvement and GPI are only mentioned briefly in Section 2.1 and Section 2.4, respectively. Since they constitute a fundamental part of the proposed method, composing the core of the TL process, these operations are explained in more detail here, for the sake of a better order of presentation of information.

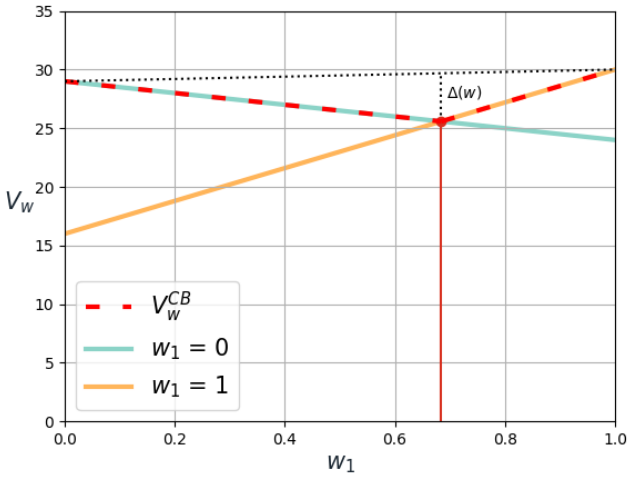
The *policy improvement theorem*, given by Theorem 2, is an extremely important result for RL. It shows that, for any state s, by selecting an action a in which $Q^{\pi}(s,a) > Q^{\pi}(s,\pi(s))$, a better policy emerges. This means that, by acting greedily with respect to the value function, policies can be improved upon.

Theorem 2. *[33, 7] Let π and π' be two deterministic policies such that, for any possible state s,*

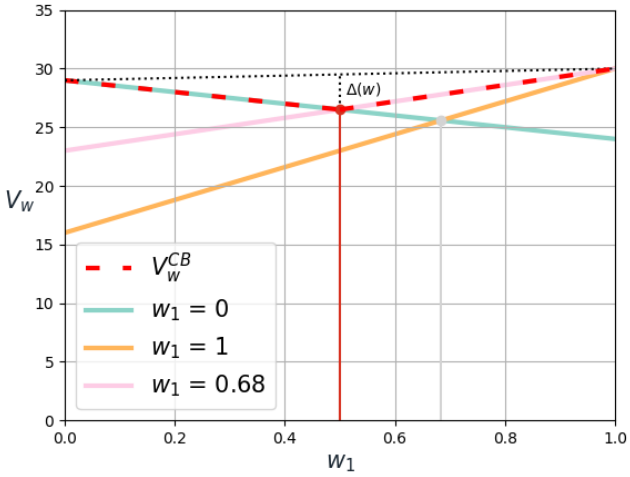
$$Q^{\pi}(s,\pi'(s)) \geq V^{\pi}(s). \tag{16}$$



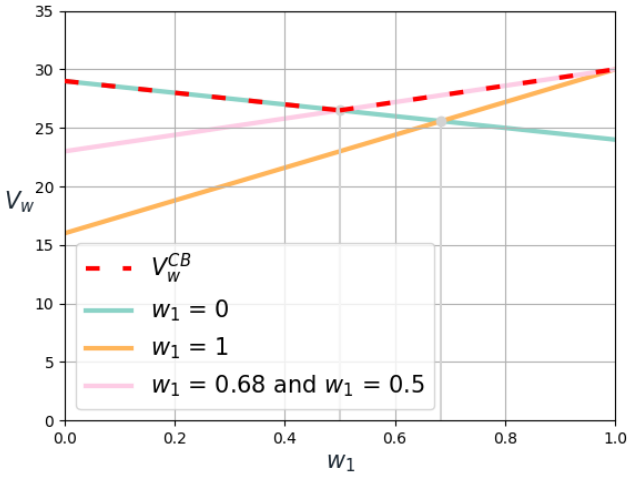
(a) Inserts weights in the extrema of the weight simplex with infinite priority. $\boldsymbol{w} = [1, 0]$ is popped off



(b) $\boldsymbol{w} = [0, 1]$ is popped off P. The corner weight $\boldsymbol{w} = [0.32, 0.68]$ is found, and is inserted in P, with a priority of $\Delta(\boldsymbol{w})$



(c) $\boldsymbol{w}$ = [0.32, 0.68] is popped off P. The corner weight $\boldsymbol{w}$ = [0.5, 0.5] is found, and is inserted in P, with a priority of $\Delta(\boldsymbol{w})$



(d) $\boldsymbol{w}$ = [0.5, 0.5] is popped off P. No new corner point is found, and P is empty. The policies computed form a CCS, and the procedure ends

Fig. 2. Computing a CCS for the example using OLS

Then, it holds that

$$\pi' \succeq \pi. \tag{17}$$

The operation of improving policies by acting greedily with respect to their value functions is called policy improvement, and is shown in Eq. 18.

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} Q^{\pi}(s, a). \tag{18}$$

Within the framework of SFs, [4] extended Theorem 2 to a set of multiple policies. This is the *generalized policy improvement theorem*, given by Theorem 3.

Theorem 3. *[4] Let Π be a set of deterministic policies and let π' be a deterministic policy such that, for any possible state s,*

$$Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s) \qquad \forall \pi \in \Pi. \tag{19}$$

Then, it holds that

$$\pi' \succeq \pi \qquad \forall \pi \in \Pi. \tag{20}$$

As aforementioned, the operation of improving a set of policies by acting greedily with respect to all of their value functions is called GPI, a generalization of policy improvement which was originally applied to the SFs framework. However, by showing the equivalence between SFs and MORL, [3] showed that it is possible to use GPI to improve policies in MORL settings. This result is key to the proposed method.

To understand how Theorem 3 can be applied to MORL, suppose the policies in Π correspond to policies learned by scalarizing a multi-objective problem using different weights. Now consider that a new weight vector is given to an agent. Because of the result in Theorem 3, this agent can create a new policy for this new weight vector by simply acting greedily with respect to the value function of all policies in Π for a scalarized version of the multi-objective problem using this new weight vector.

Now that a theoretical foundation has been laid, let us continue explaining the method. After having decentrally computed a shared set of policies Π, the agents are now ready to create new policies for any possible weight vector. Given a new weight $\boldsymbol{w}$, the agent uses Eq. 21 to generate a new policy, $\pi^{\text{GPI}}_{\boldsymbol{w}}$, best suited for the new weight vector, $\boldsymbol{w}$.

$$\pi^{\text{GPI}}_{\boldsymbol{w}}(s) \in \arg\max_{a \in \mathcal{A}} \max_{\pi \in \Pi} \boldsymbol{w} \cdot \boldsymbol{\psi}^{\pi}(s, a). \tag{21}$$

Note that, as discussed in Section 2.4, when we define the features as being the multi-objective reward function ($\boldsymbol{\mathcal{R}}(s, a, s') = \boldsymbol{\phi}(s, a, s')$), then the SFs are equivalent to the multi-objective action-value function, that is, $\boldsymbol{\psi}^{\pi}(s, a) = \boldsymbol{Q}^{\pi}(s, a)$.

For a single-agent scenario, [3] proved that the policy $\pi^{\text{GPI}}_{\boldsymbol{w}}$ is optimal for $\boldsymbol{w}$ if Π corresponds to a CCS. However, since we are interested in multi-agent scenarios, the convergence guarantees no longer hold. Nevertheless, Section 5 empirically shows that for complex MARL problems, even though $\pi^{\text{GPI}}_{\boldsymbol{w}}$ is not theoretically optimal, its performance is effective.

5. Experiments and Results

This section presents the experimental settings and results, and empirically shows that the method is both efficient at building a shared set of policies and effective at combining these policies using GPI to generate behaviors for different preferences over objectives. To evaluate the proposed method, two domains are used.

The Four-Room environment is used as one of the experimental settings in this work because it is employed in relevant related works, such as [4, 15, 3].

To show that the method also performs well for even more complex domains, and that the policies generated for new preferences over objectives are effective, a traffic signal control environment where traffic controllers optimize for both vehicles and pedestrians is also used, since this is an inherently distributed and multi-objective domain. Also, the agent's actions in this environment are highly coupled, which makes it even more challenging.

As explained in Section 4.1, the agents can use any temporal difference learning algorithm to learn policies for the weights they receive. For our experiments, in both domains, multi-objective Q-learning with experience replay [14] was used. Table 1 shows the learning parameters used. Several values for each parameter were tested, but these were the ones that induced the best performance.

Table 1. Q-learning parameters.

Parameter	Value
α	0.1
ϵ	0.05
γ	0.95
Experience replay buffer size	1000000

5.1. Four-Room

The Four-Room environment was extended to contain multiple agents. Figure 3 depicts the environment. It is a 13×13 grid composed of four separate rooms, hence its name. The black squares represent the walls separating the rooms (the agents are unable to walk through these walls). There are three different types of objects (blue squares, green triangles and red circles), which are collected by the agents once they step on their corresponding squares. Each object corresponds to a different objective (there are, therefore, three different objectives). For this multi-agent extension, a second agent has been added. The agents start in the squares indicated by the labels S_1 and S_2. Once they both reach the square indicated by the label G, the episode ends.

At each time step t, each agent observes a vector $s_t = [x, y]$ which represents the coordinates describing the agent's current position in the grid. The agents are unaware of each other, but their actions are highly coupled.

Each agent has four possible actions: up, down, right and left (which move the agent one square in the corresponding direction). Agents are unable to step on black squares (which represent walls) and unable to leave the grid.

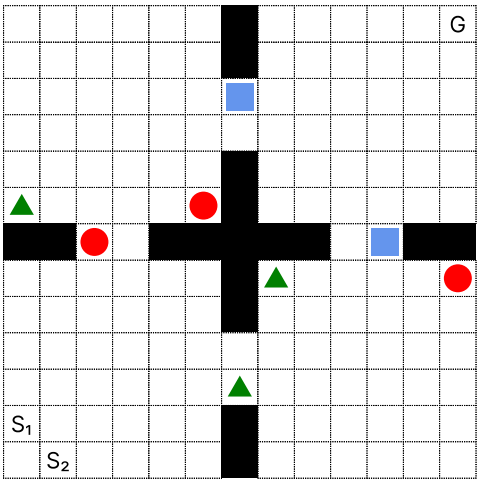


Fig. 3. Four-Room domain.

The reward $\boldsymbol{\mathcal{R}}(s, a, s') \in \mathbb{R}^3$ is a three-dimensional vector representing the type of object in the agent's current cell ([1, 0, 0] if blue square, [0, 1, 0] if green triangle, [0, 0, 1] if red circle and [0, 0, 0] if blank). Once an agent steps on a square with an object, the agent receives the respective reward, and the object disappears, so the other agent can no longer receive a non-zero reward by stepping on that square. The agents, therefore, compete against each other.

This section aims at showing that the proposed method is efficient at generating a shared set of policies that can be effectively combined via GPI. Since the method iteratively assigns weights to the agents, which learn policies to scalarized versions of the multi-objective problem at each iteration, efficiency here can be measured by the number of iterations required to build the complete set of policies. A common baseline in the literature, which is used here as well, is to use random weights at each iteration [37].

Figure 4 shows the average return of both agents for 12 random test weights, after each iteration of MA-SFOLS + GPI and Random + GPI. As mentioned, for Random + GPI each agent is assigned a random weight at each iteration (instead of the corner weights, which offer the greatest possible improvement). For both MA-SFOLS and Random, GPI is used at every iteration to combine the policies learned so far to generate behaviors for the test weights.

From the plot, it is clear that the proposed method takes considerably less iterations in order to build an approximate CCS. For this example, the shared set of policies in most runs of the MA-SFOLS was already sufficient at iteration 4, whilst Random took more than twice as many iterations (approximately 10) to reach a similar level of returns.

The results empirically show the concept explained in Theorem 1: that corner weights offer the greatest possible improvement to expand the current set of policies. In fact, according to this theorem, no other weights assigned to the agents besides the ones selected by MA-SFOLS at each iteration could have built a CCS in fewer iterations.

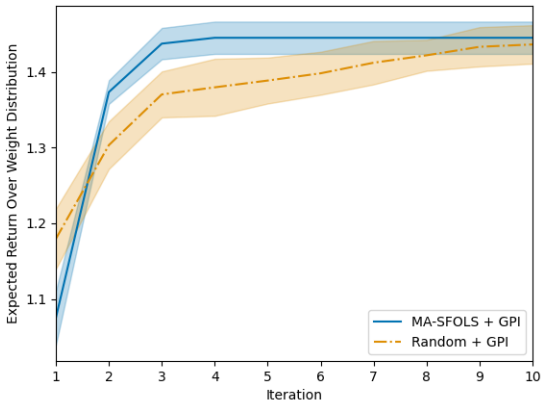


Fig. 4. Expected return of the agents over the distribution of reward weights (Four-Room).

5.2. Traffic Signal Control

The traffic scenario, shown in Figure 5, contains three intersections. Each traffic light at each intersection is controlled by an independent RL agent. The experiments were performed using SUMO-RL [2], which is based on the microscopic traffic simulator SUMO [22] (Simulation of Urban MObility). SUMO-RL provides an interface for MARL, in which each agent can receive its own observations and rewards, and provide its own action to the traffic simulation. Below we detail how the scenario and the states, actions, and rewards of the agents were defined in our experimental setting.

The scenario selected is especially interesting and non-trivial because it considers pedestrians. Figure 6 shows a zoomed view of an intersection, in which lanes, vehicles, sidewalks, pedestrians, and a crossing can be seen. Notice that there is only one crossing lane per intersection, located at the road with greater vehicular demand. This is intentional, as it ensures that a conflict exists between the vehicle and the pedestrian objectives. Were there to be a crossing on the other road, the problem would become much less complex from a multi-objective perspective, as the same actions would benefit both objectives simultaneously.

Every link has 150 m in length, two lanes and is one-way. There are four Origin-Destination (OD) pairs: $VO_1 \rightarrow VD_1$, $HO_1 \rightarrow HD_1$, $HO_2 \rightarrow HD_2$ and $HO_3 \rightarrow HD_3$. Vehicles and pedestrians are inserted in an origin node and are removed from the simulation in a destination node. Vehicles travel in the North-South (N-S) direction in vertical links, and in the West-East (W-E) direction in horizontal links. Each vertical link has a pedestrian sidewalk, and each intersection has a crossing for pedestrians. Pedestrians only go in the North-South (N-S) direction.

Traffic signals in this scenario have a minimum and maximum green time. They are referred to as *minGreenTime* and *maxGreenTime*, respectively. For the experiments, these values were 10 and 50 seconds respectively. All signals have two phases, the North-South phase, which, when green, allows pedestrians and vehicles to flow in the North-South

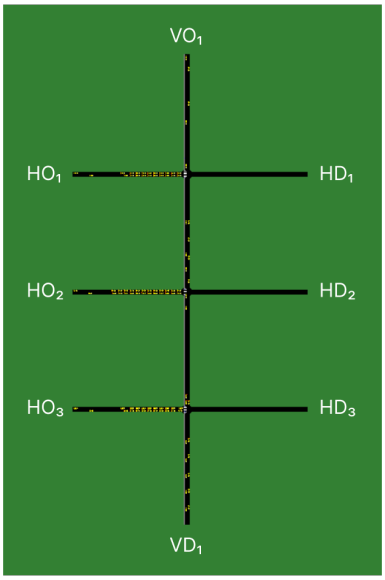


Fig. 5. Traffic signal control environment

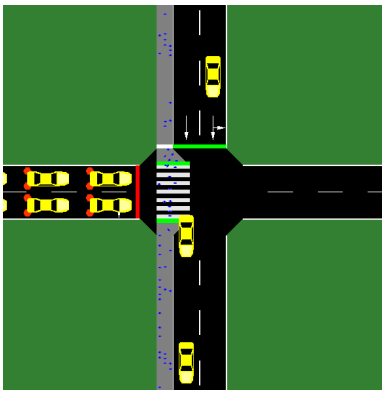


Fig. 6. Traffic intersection

direction, and the West-East phase, which, when green, allows vehicles to flow in the West-East direction.

Regarding demands, one vehicle is inserted in HO_1, HO_2 and HO_3 every 3.3 seconds, and one vehicle is inserted in VO_1 every 20 seconds. As for pedestrians, one pedestrian is inserted every second in the origin node VO_1. Each episode runs for 500 seconds. Once an episode ends, the simulation restarts.

At each time step t (which corresponds to five seconds of clock time), each agent observes a vector s_t, given by Eq. (22), which describes the current state of the respective intersection. $\rho \in \{0, 1\}$ is binary variable that indicates the current active green phase ($\rho = 0$ when the West-East phase is green, and $\rho = 1$ when the North-South and pedes-

trian phases are green). $\tau \in [0, 1]$ is the elapsed time of the current signal phase divided by *maxGreenTime* (50 seconds). L is the set of all incoming lanes (for both vehicles and pedestrians). The density $\Delta_l \in [0, 1]$ is defined as the number of vehicles or pedestrians in the incoming lane $l \in L$ divided by the total capacity of the lane. $q_l \in [0, 1]$ is defined as the number of queued vehicles or pedestrians in the incoming lane $l \in L$ divided by the total capacity of the lane. A vehicle is considered to be queued if its speed is below 0.1 m/s. A pedestrian is considered to be queued if it is stopped before a crossing waiting for its phase to turn green.

$$s_t = [\rho, \tau, \Delta_1, ..., \Delta_{|L|}, q_1, ..., q_{[L]}] \tag{22}$$

Each signal controller agent chooses a discrete action a_t at each time step t. For our scenario, since all intersections have two incoming links, there are two phases, so each agent has only two actions: *keep* and *change*. The former keeps the current green signal active, while the latter switches the current green light to another phase. The agents can only choose *keep* if the current green phase has been active for less than *maxGreenTime*, and can only choose *change* if the current green phase has been active for at least *minGreenTime*.

For this scenario, the traffic controllers have two objectives: to minimize the waiting time of vehicles, and to minimize the waiting time of pedestrians. Thus, the reward function $\boldsymbol{\mathcal{R}} \rightarrow \mathbb{R}^2$ is a two-dimensional vector, given by Eq. (23), where the first component is the change in cumulative vehicle waiting time between successive time steps, and the second component is the change in cumulative pedestrian waiting time between successive time steps.

$$\boldsymbol{\mathcal{R}}_t = [Wv_t - Wv_{t+1}, Wp_t - Wp_{t+1}] \tag{23}$$

Wv_t is the cumulative vehicle waiting time at time step t, given by Eq. (24), where V_t is the set of incoming vehicles, and $w_{v,t}$ is the total waiting time of vehicle v since it entered the incoming road until time step t.

$$Wv_t = \sum_{v \in V_t} w_{v,t} \tag{24}$$

Wp_t is the cumulative pedestrian waiting time at time step t, given by Eq. (25), where P_t is the set of incoming pedestrians, and $w_{p,t}$ is the total waiting time of pedestrian p at the crossing until time step t.

$$Wp_t = \sum_{p \in P_t} w_{p,t} \tag{25}$$

Figure 7 shows the average return of all three traffic controllers for 25 random test weights,[2] after each iteration of MA-SFOLS + GPI. The performance of a few policies generated by GPI is shown in Figure 8, Figure 9 and Figure 10. Figure 8 shows the waiting time of pedestrians for the selected policies, and it is clear that the greater the value of w_1 (the weight corresponding to the pedestrian objective), the smaller the waiting time of pedestrians (thus, greater the performance for this objective). Figure 9 shows the waiting

[2] We uniformly sample weight vectors by sampling from a d-dimensional Dirichlet distribution ($\boldsymbol{\alpha} = \mathbf{1}$), as in [1].

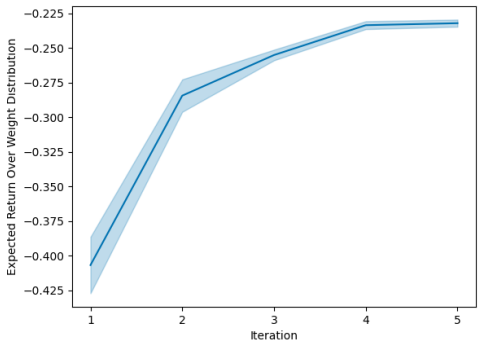


Fig. 7. Expected return of the agents over the distribution of reward weights

time of vehicles for the selected policies, and the same effect is visible (the greater the value of the weight corresponding to the vehicle objective, the better the performance for this objective). Figure 10 shows a zoomed view of Figure 9.

Note that the legends of the plots only show one dimension of the weight vector (since there are only two objectives, $w_0 = 1 - w_1$ and $w_1 = 1 - w_0$). If a plot shows waiting time of vehicles, the legend shows the weight dimension for vehicles (w_0), and if a plot shows waiting time of pedestrians, the legend shows the weight dimension for pedestrians (w_1).

It is clear from these plots how conflicting both objectives are (the higher the preference over one objective, the worse the performance is for the other objective). This makes this problem even more difficult, and also helps to serve as a motivation for multi-objective methods. Clearly, for this problem, a multi-objective method is required, as a priori scalarization of the problem would not be sufficient to address this scenario if the weights were not known in advance.

Section 5.1 showed the efficiency of the proposed method (measured by iterations required to form an approximate CCS). This section aims at showing the effectiveness of the method, that is, that the policies generated by GPI are effective for new reward weights. Thus, in the next experiments, we present an evaluation phase in which the set of policies identified during the training phase (via Algorithm 1) is shared with all agents. We compare the performance of policies generated via GPI with policies learned by Q-learning given a few selected reward weights.

Figures 11, 12, 13, 14 and 15 show a comparison between the performance (for both objectives) of policies generated by MA-SFOLS + GPI and the performance of policies learned by Q-learning via a priori scalarization for a few weights. It is clear that the policies generated by GPI have a very similar performance to the policies directly learned by Q-learning via a priori scalarization. Therefore, these results show that the method is able to appropriately leverage previous knowledge in order to create effective policies for new values of $\boldsymbol{w}$.

The results in this section and in Section 5.1 empirically show that the method is both efficient and effective at building a shared set of policies that can be combined via GPI

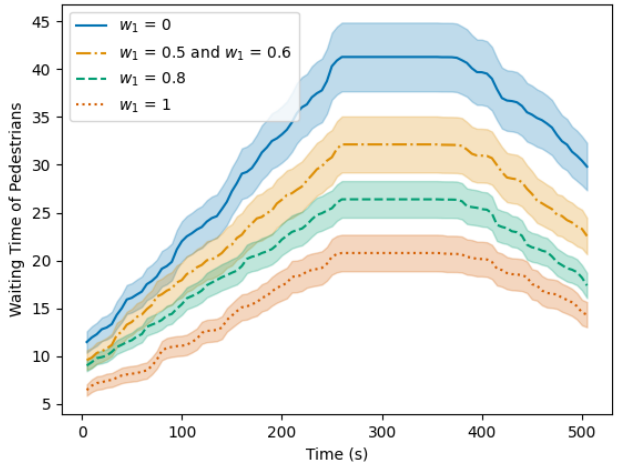


Fig. 8. Waiting time of pedestrians for a few policies generated by GPI

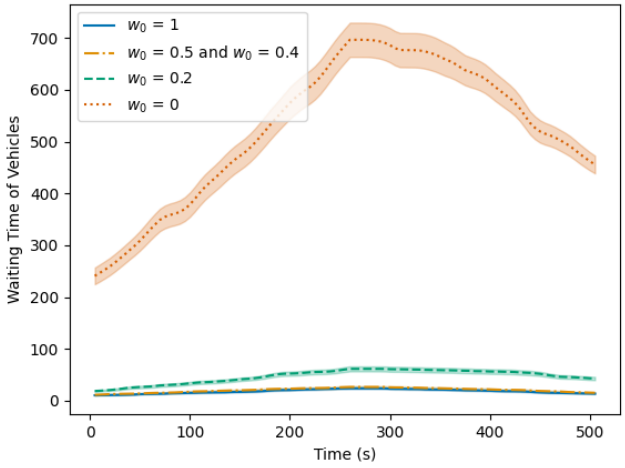


Fig. 9. Waiting time of vehicles for a few policies generated by GPI

to generate behaviors for new weights, so that the RL agents can perform well for any required preference of objectives.

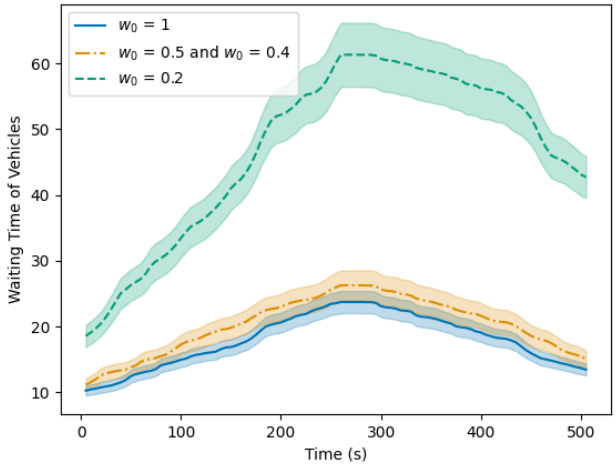


Fig. 10. Zoomed view of waiting time of vehicles

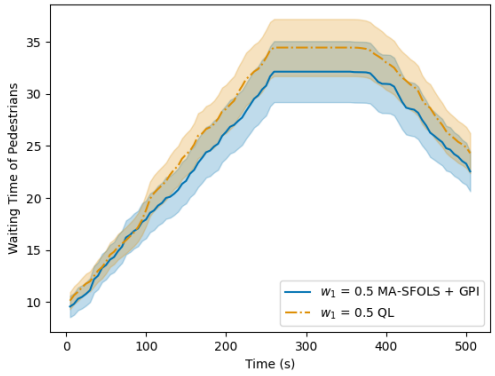


Fig. 11. MA-SFOLS + GPI vs Q-learning: $\boldsymbol{w} = [0.5, 0.5]$ (pedestrians)

6. Conclusion

This work introduced a multi-objective multi-agent transfer learning method in which homogeneous agents decentrally build a shared set of policies during training. During the execution/evaluation phase, the agents combine these policies via GPI to create new policies specialized to any new linear combinations of their objectives. Within the method, two layers of knowledge transfer can be discerned: knowledge sharing and policy transfer.

Knowledge sharing refers to the fact that the policies learned by one agent can be used by another, thus all the agents in the system share their knowledge with each other. Policy transfer refers to the fact that, by leveraging a generalization of policy improvement, the

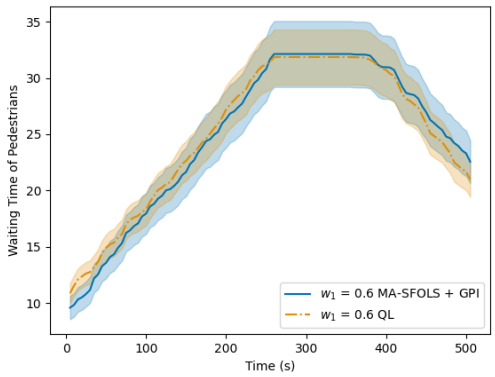


Fig. 12. MA-SFOLS + GPI vs Q-learning: $\boldsymbol{w} = [0.4, 0.6]$ (pedestrians)

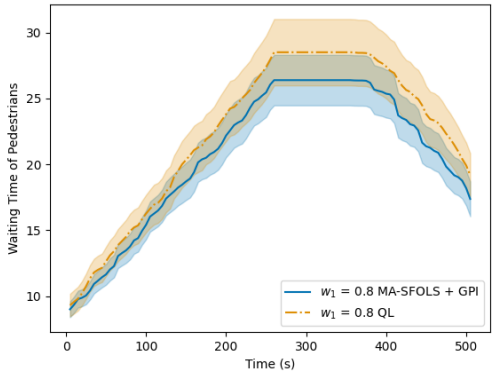


Fig. 13. MA-SFOLS + GPI vs Q-learning: $\boldsymbol{w} = [0.2, 0.8]$ (pedestrians)

proposed method also enables agents to combine policies learned by themselves and by the other agents in order to create new policies, thus, policy transfer.

The proposed method is very useful for problems that are inherently distributed (therefore, a multi-agent solution is preferable or required) and multi-objective, and scalarizing them is not an option, either because the weights for the objectives are not known during training, or because the trade-off between objectives may change over time.

A traffic signal control domain with vehicles and pedestrians was used to evaluate the method, in addition to a standard domain in the SFs literature. It must be noted that optimizing for pedestrians is rarely addressed in the RL literature for traffic signal control.

The results empirically showed that the method is both efficient and effective at building a shared set of policies that can be combined via GPI to generate behaviors for new weights, so that the RL agents can perform well for any required preference of objectives.

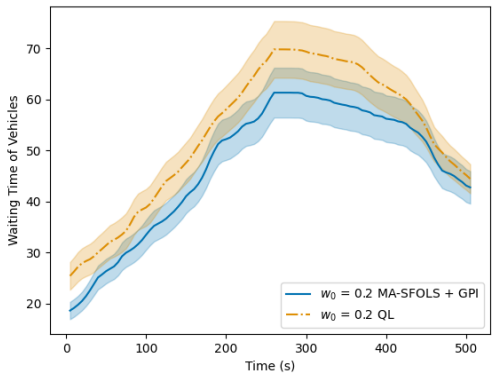


Fig. 14. MA-SFOLS + GPI vs Q-learning: $\boldsymbol{w} = [0.2, 0.8]$ (vehicles)fig:four-room-returns

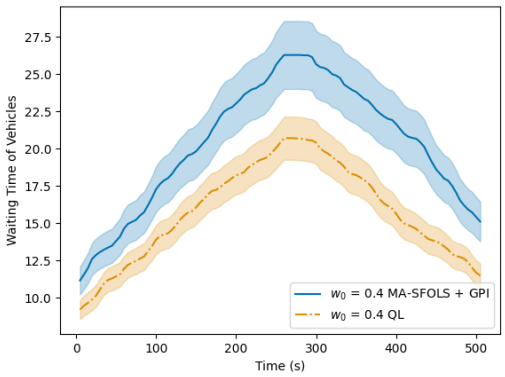


Fig. 15. MA-SFOLS + GPI vs Q-learning: $\boldsymbol{w} = [0.4, 0.6]$ (vehicles)fig:four-room

This is one of the first works to address TL in the context of MOMARL, and the first work (to the authors' best knowledge) that leverages generalized policy improvement for settings with multiple objectives and multiple agents.

In future work, we would like to address the two main limitations of our approach. First, we assumed linear utility functions, and thus our method is only able to identify policies in the CCS. How to deal with the more general case of non-linear utility functions is still an open problem in MORL [34]. Second, we considered independent learning agents that do not take into account the effects of non-stationarity caused by other agents. Extending our method in a way such that agents' policies are conditioned on the preferences of other agents is a promising research direction. In order to tackle this problem, we plan to design a method that learns an inherently multi-agent coverage set, instead of a regular CCS; that is, a coverage set that takes into account the preferences over objectives of all agents simultaneously.

Acknowledgments. Ana Bazzan is partially supported by CNPq (grant 304932/2021-3). This work was partially funded by FAPESP and MCTI/CGI (grants number 2020/05165-1) and by CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil, Finance Code 001), and also partially sponsored by the German Federal Ministry of Education and Research (BMBF), Käte Hamburger Kolleg Cultures des Forschens/ Cultures of Research. We are grateful to the anonymous reviewers inputs.

References

1. Abels, A., Roijers, D.M., Lenaerts, T., Nowé, A., Steckelmacher, D.: Dynamic weights in multi-objective deep reinforcement learning. In: Proceedings of the 36th International Conference on Machine Learning. vol. 97, pp. 11–20. International Machine Learning Society (IMLS) (2019)
2. Alegre, L.N.: SUMO-RL. https://github.com/LucasAlegre/sumo-rl (2019)
3. Alegre, L.N., Bazzan, A.L.C., da Silva, B.C.: Optimistic linear support and successor features as a basis for optimal policy transfer. In: Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., Sabato, S. (eds.) Proceedings of the 39th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 162, pp. 394–413. PMLR (17–23 Jul 2022), https://proceedings.mlr.press/v162/alegre22a.html
4. Barreto, A., Dabney, W., Munos, R., Hunt, J.J., Schaul, T., van Hasselt, H.P., Silver, D.: Successor features for transfer in reinforcement learning. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017)
5. Barreto, A., Hou, S., Borsa, D., Silver, D., Precup, D.: Fast reinforcement learning with generalized policy updates. Proceedings of the National Academy of Sciences 117(48), 30079–30087 (2020)
6. Bazzan, A.L.C.: Opportunities for multiagent systems and multiagent reinforcement learning in traffic control. Autonomous Agents and Multiagent Systems 18(3), 342–375 (June 2009)
7. Bellman, R.E.: Dynamic Programming. Princeton University Press, Princeton (1957)
8. Borsa, D., Barreto, A., Quan, J., Mankowitz, D.J., Munos, R., Hasselt, H.V., Silver, D., Schaul, T.: Universal successor features approximators. In: Proceedings of the 7th International Conference on Learning Representations (ICLR) (2019)
9. Buşoniu, L., Babuska, R., De Schutter, B.: A comprehensive survey of multiagent reinforcement learning. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 38(2), 156–172 (2008)
10. Cheng, H.T.: Algorithms for partially observable Markov decision processes. Ph.D. thesis, University of British Columbia (1988), https://open.library.ubc.ca/collections/ubctheses/831/items/1.0098252
11. Dayan, P.: Improving generalization for temporal difference learning: The successor representation. Neural Computation 5(4), 613–624 (1993)
12. Duan, H., Li, Z., Zhang, Y.: Multiobjective reinforcement learning for traffic signal control using vehicular ad hoc network. EURASIP Journal on Advances in Signal Processing 2010 (12 2010)
13. Egea, A.C., Connaughton, C.: Assessment of reward functions in reinforcement learning for multi-modal urban traffic control under real-world limitations (2020), arXiv preprint arXiv:2010.08819
14. Fedus, W., Ramachandran, P., Agarwal, R., Bengio, Y., Larochelle, H., Rowland, M., Dabney, W.: Revisiting fundamentals of experience replay. In: Proceedings of the 37th International Conference on Machine Learning. Vienna, Austria (2020)
15. Gimelfarb, M., Barreto, A., Sanner, S., Lee, C.G.: Risk-aware transfer in reinforcement learning using successor features. In: Proceedings of the 35th Annual Conference on Advances in Neural Information Processing Systems. Online (2021)

16. Gupta, T., Mahajan, A., Peng, B., Böhmer, W., Whiteson, S.: Uneven: Universal value exploration for multi-agent reinforcement learning (2021), arXiv preprint arXiv:2010.02974
17. Hayes, C.F., Rădulescu, R., Bargiacchi, E., Källström, J., Macfarlane, M., Reymond, M., Verstraeten, T., Zintgraf, L.M., Dazeley, R., Heintz, F., Howley, E., Irissappane, A.A., Mannion, P., Nowé, A., Ramos, G., Restelli, M., Vamplew, P., Roijers, D.M.: A practical guide to multi-objective reinforcement learning and planning. Autonomous Agents and Multi-Agent Systems 36(1), 26 (Apr 2022), https://doi.org/10.1007/s10458-022-09552-y
18. Khamis, M.A., Gomaa, W.: Enhanced multiagent multi-objective reinforcement learning for urban traffic light control. In: 2012 11th International Conference on Machine Learning and Applications. vol. 1, pp. 586–591 (2012)
19. Kim, S.H., Stralen, N.V., Chowdhary, G., Tran, H.T.: Disentangling successor features for coordination in multi-agent reinforcement learning. In: International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2022. pp. 751–760 (2022)
20. Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: A survey. The International Journal of Robotics Research 32(11), 1238–1274 (2013)
21. Liu, W., Niu, D., Dong, L., Sun, C.: Efficient exploration for multi-agent reinforcement learning via transferable successor features. IEEE/CAA Journal of Automatica Sinica 9 (2022)
22. Lopez, P.A., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flötteröd, Y.P., Hilbrich, R., Lücken, L., Rummel, J., Wagner, P., Wießner, E.: Microscopic traffic simulation using SUMO. In: The 21st IEEE International Conference on Intelligent Transportation Systems (2018)
23. Mazyavkina, N., Sviridov, S., Ivanov, S., Burnaev, E.: Reinforcement learning for combinatorial optimization: A survey. Computers and Operations Research 134, 105400 (2021)
24. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature 518(7540), 529–533 (Feb 2015)
25. Noaeen, M., Naik, A., Goodman, L., Crebo, J., Abrar, T., Far, B., Abad, Z.S.H., Bazzan, A.L.C.: Reinforcement learning in urban network traffic signal control: A systematic literature review (2021), engrxiv.org/ewxrj
26. Rădulescu, R., Mannion, P., Roijers, D., Nowé, A.: Multi-objective multi-agent decision making: a utility-based analysis and survey. Autonomous Agents and Multi-Agent Systems 34 (04 2020)
27. Roess, R.P., Prassas, E.S., McShane, W.R.: Traffic Engineering. Prentice Hall, 3rd edn. (2004)
28. Roijers, D.: Multi-Objective Decision-Theoretic Planning. Ph.D. thesis, University of Amsterdam (2016)
29. Roijers, D.M., Vamplew, P., Whiteson, S., Dazeley, R.: A survey of multi-objective sequential decision-making. J. Artificial Intelligence Research 48(1), 67–113 (Oct 2013)
30. Shapley, L.S.: Stochastic games. Proceedings of the National Academy of Sciences 39(10), 1095–1100 (1953)
31. Silva, F.L.d., Costa, A.H.R.: A survey on transfer learning for multiagent reinforcement learning systems. Journal of Artificial Intelligence Research 64, 645–703 (2019)
32. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D.: Mastering chess and shogi by self-play with a general reinforcement learning algorithm (2017)
33. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. The MIT Press, second edn. (2018)
34. Vamplew, P., Foale, C., Dazeley, R.: The impact of environmental stochasticity on value-based multiobjective reinforcement learning. Neural Computing and Applications (Mar 2021)
35. Vamplew, P., Yearwood, J., Dazeley, R., Berry, A.: On the limitations of scalarisation for multi-objective reinforcement learning of pareto fronts. In: Wobcke, W., Zhang, M. (eds.) AI 2008: Advances in Artificial Intelligence. pp. 372–378. Springer, Berlin, Heidelberg (2008)
36. Wei, H., Zheng, G., Gayah, V.V., Li, Z.: A survey on traffic signal control methods (2020), http://arxiv.org/abs/1904.08117, preprint arXiv:1904.08117

37. Yang, R., Sun, X., Narasimhan, K.: A generalized algorithm for multi-objective reinforcement learning and policy adaptation. In: Wallach, H., Larochelle, H., Beygelzimer, A., d' Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32. pp. 14610–14621 (2019)
38. Yau, K.L.A., Qadir, J., Khoo, H.L., Ling, M.H., Komisarczuk, P.: A survey on reinforcement learning models and algorithms for traffic signal control. ACM Comput. Surv. 50(3) (2017)
39. Yin, B., Menendez, M.: A reinforcement learning method for traffic signal control at an isolated intersection with pedestrian flows. pp. 3123–3135 (07 2019)
40. Yu, C., Liu, J., Nemati, S., Yin, G.: Reinforcement learning in healthcare: A survey. ACM Comput. Surv. 55(1) (nov 2021)

Vicente Almeida received the B.Sc. degree (cum laude) in computer engineering from the Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 2022, where he is currently pursuing the masters degree with the Institute of Informatics. His research interests include data exploration, reinforcement learning, and visual analytics. In his masters, he is currently working on the problem of exploring large datasets by combining multiple hypothesis testing with data-informed dimensions.

Lucas N. Alegre received the B.Sc. degree (cum laude) in computer science from the Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 2021, where he is currently pursuing the Ph.D. degree with the Institute of Informatics. Part of his Ph.D. was done in the AI Lab at the Vrije Universiteit Brussel (VUB). His research interests include reinforcement learning, machine learning, artificial intelligence, and their applications to real-world problems. In particular, in his Ph.D., he is tackling the problem of how to design sample-efficient multi-task and multi-objective reinforcement learning algorithms capable of learning multiple behaviors that can be combined to solve novel problems.

Ana Bazzan is a professor of Computer Science at the Institute of Informatics at the Universidade Federal do Rio Grande do Sul (UFRGS), where she leads the Artificial Intelligence Group. Her research focuses on multiagent systems, in particular on agent-based modeling and simulation, and multiagent learning. Since 1996, she has collaborated with various researchers in the application of multiagent systems. In recent years, she has contributed to different topics regarding smart cities, focusing on transportation. In 2014, she was General Co-chair of AAMAS (the premier conference in the area of autonomous agents and multiagent systems).

Received: December 10, 2022; Accepted: September 02, 2023.