Constructivist Approach to State Space Adaptation in Reinforcement Learning

Conference Paper · June 2019

DOI: 10.1109/SASO.2019.00016

3 authors:

Maxime Guériau
Trinity College Dublin
20 PUBLICATIONS 70 CITATIONS
SEE PROFILE

Nicolás Cardozo
Los Andes University (Colombia)
52 PUBLICATIONS 256 CITATIONS
SEE PROFILE

Ivana Dusparic
Trinity College Dublin
48 PUBLICATIONS 262 CITATIONS
SEE PROFILE

Constructivist Approach to State Space Adaptation in Reinforcement Learning

Maxime Guériau
School of Computer Science and Statistics
Trinity College Dublin, Ireland
maxime.gueriau@scss.tcd.ie

Nicolás Cardozo
Systems and Computing Engineering Department
Universidad de los Andes
n.cardozo@uniandes.edu.co

Ivana Dusparic
School of Computer Science and Statistics
Trinity College Dublin, Ireland
ivana.dusparic@scss.tcd.ie

***Abstract*—Reinforcement Learning (RL) is increasingly used to achieve adaptive behaviours in Internet of Things systems relying on large amounts of sensor data. To address the need for self-adaptation in such environments, techniques for detecting environment changes and re-learning behaviours appropriate to those changes have been proposed. However, with the heterogeneity of sensor inputs, the problem of self-adaptation permeates one level deeper; in order for the learnt behaviour to adapt, the underlying environment representation needs to adapt first. The granularity of the RL state space might need to be adapted to learn more efficiently, or to match the new granularity of input data. This paper proposes an implementation of Constructivist RL (Con-RL), enabling RL to learn and continuously adapt its state space representations. We propose a Multi-Layer Growing Neural Gas (ML-GNG) technique, as an extension of the GNG clustering algorithm, to autonomously learn suitable state spaces based on sensor data and learnt actions at runtime. We also create and continuously update a repository of state spaces, selecting the most appropriate one to use at each time step. We evaluate Con-RL in two scenarios: the canonical RL mountain car single-agent scenario, and a large-scale multi-agent car and ride-sharing scenario. We demonstrate its ability to adapt to new sensor inputs, to increase the speed of learning through state space optimization, and to maintain stable long-term performance.**

***Keywords*-dynamic adaptation; reinforcement learning**

I. Introduction and Motivation

Reinforcement Learning (RL) is increasingly applied in large-scale cyber-physical systems to enable their runtime adaptation, *i.e.,* to learn the appropriate actions to take in response to interactions with the environment. Behaviour in such systems is not predefined, but learnt at run time, enabling the self-adaptation to situations that were unexpected or unknown at design time. Such applications include, for example, learning optimal traffic light settings in response to real-time traffic conditions [1], learning optimal taxi positioning in response to customer demand [2], or learning optimal EV charging times in response to availability of renewable energy [3]. However, conditions in such environments change and RL processes designed and deployed at the start of the system operation will become unsuitable and will need to further self-adapt during the system execution. Different techniques are being developed to enable RL to operate in such non-stationary environments, by, for example, continuously monitoring and predicting environment conditions. If an environment deviation is observed from the expected conditions, the learning process can be re-triggered, or previously learnt and stored knowledge can be reused [4].

However, with the increasing reliance on both mobile and static Internet of Things (IoT) devices to provide environment information, an additional source of non-stationarity arises as sensor information may come from multiple sensors with different spatial and temporal availability, or change in granularity over time. We postulate that adaptation in the self-adaptive systems which utilize RL needs to permeate one level deeper; instead of only learning how to adapt the behaviour, the learning process itself needs to be adapted to address sensor-induced changes. The representation of the environment on which the learning process is based, *i.e.,* RL state space, needs to be self-adaptive as well, in order to allow seamless adaptation to a wider range of new environment conditions. For example, consider an application receiving location information from a user's mobile phone. At certain times or locations, the precise GPS location is not available and the RL process needs to switch back and forth between fine-grained GPS information and coarser location information based on cell tower triangulation. In addition, even when fine-grained sensor information is available, learning based on the state space with the finest granularity may not be efficient, and individual states need to be aggregated into coarser-granularity states.

Therefore, throughout its lifetime, an RL agent is required to dynamically switch between different environment representations as well as learn new ones, even if its goal or other environment conditions do not change. Figure 1 illustrates an example of such required self-adaptation. An agent starts off using state space S1 provided at design time. Over time, if sensors with finer granularity become available, the agent needs to learn and use new state space representations S2 and S3, based on new sensor data. However, if sensor S3 uses a very fine granularity, resulting in a very large state space, the learning process would become too slow. To accelerate learning, the agent needs to learn how to simplify state space S3 into a coarser granularity state space S4.

Granularity of the state space needs to be tailored to specific environment conditions. Finer-grained state space representation may lead to a better action after an extensive learning

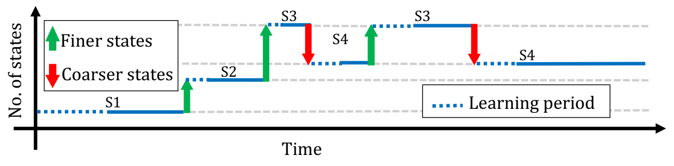


Fig. 1: Dynamic state space adaptation required in RL

period, but the one with coarser-grained states space will converge faster, leading to better actions early on.

Enabling this process to happen seamlessly throughout agent's lifetime requires an agent to autonomously learn state space representations based on sensor information, optimize/simplify them, and switch between the available representations dynamically. These processes of state space specification, discretization and aggregation are currently separate and often manual, while, we argue that, in order to enable self-adaptation of real world sensor-based RL systems, they need be interleaved and continuous.

Effectively, the adaptation process needs to closer mimic the continuous construction and adaptation of knowledge as achieved by humans. If the sensor input matches the existing state space representation, information learnt based on it should be incorporated into existing learning processes, while if a new sensor is available, new state spaces need to be learnt and adapted for new knowledge to be acquired. In human learning, an analogy to such an approach can be drawn from the constructivist theory [5], which proposes that humans build internal frameworks of knowledge, and acquire new knowledge either through assimilation (incorporating new knowledge into their existing framework) or accommodation (re-framing internal representations to the newly acquired external knowledge). Inspired by the theory of constructivism, this paper proposes Constructivist Reinforcement Learning (Con-RL), an algorithm for autonomous and continuous learning of state space representations and their run-time adaptation, as a step towards fully self-adaptive RL. Con-RL is aimed at RL-based self-adaptive systems regardless of the underlying RL algorithm they use, as its state space generation and adaptation mechanism is independent of the specific algorithm learnt to use the adaptive behaviours.

The main contributions of our approach are: (1) a Multi-layer Growing Neural Gas (ML-GNG), a novel extension of Growing Neural Gas (GNG) that enables clustering by multiple criteria by using multiple layers, enabling each layer to learn the areas of the state space in which to apply a single agent action. ML-GNG learns to generalize over experienced areas of the state space to generate a smaller optimized state space, enabling the agent to learn faster. (2) a repository of state spaces and a decision making mechanism for switching between available state space representations by selecting the best representation at each time step, enabling the agent to maintain long-term stable performance.

While these two contributions do not fully achieve seamless and complete runtime state space adaptation (remaining open issues will be discussed in Section V), they are crucial for enabling addition or removal of sensors, as well as adapting to changes in sensor granularity. A new ML-GNG can be initialized at system start, every time a sensor is added, or when a granularity of a sensor changes, in order to learn the most suitable state space representation given available granularity.

We demonstrate the applicability and benefits of Con-RL using the mountain car canonical single-agent RL scenario, and a large-scale multi-agent car and ride-sharing scenario using real world data.

II. Related Work

Constructivism is a knowledge acquisition theory proposing that human learning happens as a result of the interaction of a person's internal mental representations and their external perceptions [5]. In agent learning, several constructivism-inspired applications of knowledge acquisition have been proposed, *e.g.,* in grid-world and robotics tasks [6], or in road-side traffic control [7]. The idea of combining constructivism and RL approaches to agent learning (although they contrast, with the former learning from internal adaptations and the latter from external feedback) has also been proposed, but so far only at a conceptual level [8], in order to benefit from the advantages of both approaches.

RL is an unsupervised learning technique in which an agent learns an action policy based on trial and error, by obtaining a reward from the environment for actions taken. The domain of the problem, *i.e.,* the input space, is often modelled using a finite set of states, and the goal of the learner is, at each time step, to choose the action which maximizes the long-term reward for a given state. To define the RL process, state space discretization, actions and rewards need to be specified. Manual state space specification is most commonly used and is generally accurate as it utilizes expert domain knowledge, but is not feasible in situations where a suitable state space might need to be derived from the input automatically, or adapted at run time.

Simple state space refinement at run time can be achieved using a grid-based discretization [9], where certain areas of the state space can be adapted to finer or coarser grid representations. State space aggregation techniques (*e.g.*, [10] can be used to aggregate multiple states into a single one to reduce the state space size and therefore improve the learning performance. However, in both of these approaches, the initial discretization needs to be provided manually, which require prior knowledge on the observed environment. State representations learning approaches are presented in [11], [12], but are very specific and limited to the domains in which they are applied (*e.g.,* a robot learning a physical representation of its surroundings and an agent learning states in which coordination is required).

Function approximators can be applied to discretize the input space dynamically. For instance, in [13], the use of a continuous function approximator combined to Temporal Difference (TD) errors values allows for an adaptive input space partitioning. However, the algorithm is limited to problem expressed with TD values and need to be extended to tackle non-stationary environments.

Clustering techniques have been used to enable dynamic state space generation from continuous or high dimensional input (this refers to the partition problem defined in [13]). Learning Vector Quantization, a supervised quantization technique, is used to generate labelled clusters corresponding to discrete states [14]. Growing Neural Gas (GNG) [15], an unsupervised online clustering technique, has been used as a basis for several state space partitioning techniques which underlie Q-learning [16] (a model-free RL algorithm) processes. GNG generates and updates a self-organizing network of nodes that learns the topology of the observed environment [17], and the Voronoi tessellation arising from nodes' position can be used to define an RL state space. Q-Learning Growing Neural Gas (GNG-Q) [18] is the approach most closely related to our proposal. In GNG-Q, the learning process starts with a coarse state space representation, which is gradually refined, by adding new nodes in the regions where changes in the learnt policy are observed. GNG-Q nodes are associated with Q-values propagated from the underlying learning process, which are updated at the end of a learning episode. However, as the policy has to be learnt before it is possible to observe its changes (and update Q-values), this process is slow and difficult to apply online, as well as only applicable in episodic learning environments.[1] In Temporal Difference GNG (TD-GNG) [19], another combination of GNG and RL, outdated nodes are not removed, but new nodes are added sparingly, and potentially moved towards areas where the probability of receiving positive rewards is higher. However, lack of removal of old nodes prevents GNG from capturing environment changes effectively and in a timely manner. In addition, those algorithms require the use of Q-values and hence are only suitable with an existing underlying Q-learning process.

In terms of run-time selection between multiple available state representations, the problem has only been studied in the context of multi-objective RL [20], [21], where different objectives are represented using different state spaces and are competing for control of the agent. In our scenario of dynamic state space adaptation, multiple state spaces are a result of different sensors providing input for the same goal. The decision is, therefore, not which objective takes the priority at a given time, but for which state space representation does an agent a previously more accurately learnt action.

Our proposed approach, Con-RL, is applicable to self-adaptive systems relying on a reinforcement learning process to interact with a dynamic environment. Con-RL addresses both state space learning/optimization and its run-time selection. The proposed state space learning technique, ML-GNG, uses multiple layers (where each layer represents a single action) to learn suitable actions to execute in different areas of the state space. Each layer uses a modified GNG-U [17] algorithm, an extension of GNG which provides an additional refinement technique for outdated node removal. In this way, nodes can be continuously removed and added, allowing the state space representation to follow the changes in the policy.

[1]We further discuss implications of online policy learning in Section IV where we compare the performance of our approach to GNG-Q.

III. Con-RL: Constructivist RL for Dynamic State Space Adaptation

Run-time adaptation of RL state spaces consists of two main aspects: (i) dynamically generating state spaces from the available sensor information (either at system initialization, or at run time during the system operation when a new sensor becomes available), and (ii) dynamically switching between representations either based on sensor availability or based on quality of knowledge in each available state space.

We first present the concepts and environment definitions required Con-RL to achieve self-adaptation, and then describe the algorithms for both components.

A. Dynamic State Space Learning Model

The RL system capable of dynamic state space adaptation consists of sensor-based state spaces, learnt state spaces, an algorithm that enables state space learning, and an algorithm that enables state space switching/selection. We define these below.

Sensor-based state spaces: An agent has one or more goals. Each goal has a corresponding learning process based on a discrete state space environment representation $SR_k = \{sr_1^k, \ldots, sr_m^k\}$, which is a set of states generated from the information gathered from a sensor relevant for that goal. The granularity of the sensor-based state space is derived directly from the corresponding sensor granularity, and defined as $\min\{|sr_i^k - sr_j^k|\}$, the minimum distance between any two unique sensor-based states sr_i^k, sr_j^k. The uniform distance between the states is derived from the full input range divided by the number of unique states.The sensor-based state space has the finest sensor granularity achievable.

Learnt state spaces: Every state space SR_k corresponds to exactly one sensor used to provide information required for a particular goal $R_k^{g_i}$, with granularity G_k. We propose that every sensor $R_k^{g_i}$ can also be used to learn other state spaces of coarser granularity $G_s < G_k$. That is, the sensor-based states can be aggregated into states of coarser granularity to create a new state space in order to improve the learning process. Therefore, along with the sensor-based state spaces $SR_1, \ldots, SR_n$, every goal also haves corresponding learnt-based state spaces $SL_1, \ldots, SL_n$, where each state $sl_i^k \in SL_k$ is mapped to at least one state $sr_j^k \in SR_k$ (but generally more than one, in order to result in coarser state space). Thus the mapping function $g : SR_k \rightarrow SL_k$ is surjective.

Multi-layer Growing Neural Gas: We propose a new technique, named Multi-layer Growing Neural Gas (ML-GNG), to generate the learnt state space SL_k from a sensor-based state space SR_k. For every SR_k there is an instance of ML-GNG generating and continuously updating the corresponding SL_k. Therefore, states in each SR_k are fixed, while the states in each SL_k change over time, as the state space is dynamically adapted.

B. Con-RL Agent: State Space Generation using ML-GNG

Con-RL agents incorporate the proposed Multi-layer Growing Neural Gas (ML-GNG) technique, that enables state clustering according to multiple criteria simultaneously, where each criteria is implemented in an independent learning layer, associated to a corresponding goal g_i. We present the high-level overview of Con-RL in Algorithm 1.

Algorithm 1 Con-RL high-level overview

```
Input: Signal $s$: current observation in the n-dimensional input space; $m$: number of actions available to the agent
Output: Action $a_{j+1}$: to be executed in the environment
1:  QLearning.INITIALIZE()
2:  ML-GNG.INITIALIZE($m$)
3:  for each Signal $s$ received at timestep $t$ do
4:      $sr_j^k$ := QLearning.GETSTATE($s$)
5:      QLearning.UPDATE($sr_{j-1}^k, a_j, \alpha, \gamma, r_j$)
6:      if ACTIONCOUNTER($a_j$, $sr_{j-1}^k$) $> \theta$ then
7:          ML-GNG.GETLAYER($a_j$).UPDATE($s$)
8:      end if
9:      $a_{QLearning}$ := QLearning.GETPOLICY($sr_j^k$)
10:     $a_{ML-GNG}$ := ML-GNG.GETPOLICY($s$)
11:     $a_{j+1}$ := SELECTACTION($a_{QLearning}$,$a_{ML-GNG}$)
12:     EXECUTE($a_{j+1}$)
13: end for
```

At system initialization (or at run time, if and when new sensors become available), we sample sensor data and infer sensor granularity. That granularity determines the granularity of sensor-based state space SR_k, based on which a new learning process is instantiated. To illustrate the process, we use Q-learning [16], but any approach resulting in a mapping of actions to states can be used. In Q-learning, an agent learns to associate every possible action it can execute with the expected long-term reward of taking that particular action in a particular set of environment conditions. This is represented by a Q-value, $Q(s,a)$, which is updated as follows:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r_i + \gamma \max_{a_i'} Q(s_i', a_i')) \quad (1)$$

where r_i is the immediate reward received from the environment, s is the current environment state, a is the current action, and s_i' and a_i' respectively are the next possible environment and action for goal i. α and γ are learning parameters which assign weight to new agent's experiences and the rate at which old experiences are discounted.

For each instantiated Q-learning process, an ML-GNG is also instantiated, using m different layers, where m is the cardinality of the agent's action set. Each layer of ML-GNG, as depicted in Figure 2, is based on a single Growing Neural Gas with Utility (GNG-U) [17] clustering process and dedicated to one action a. GNG-U is an extension of the plain GNG with an additional refinement step used to identify and remove outdated nodes, helping the network to adapt to non-stationary environments.

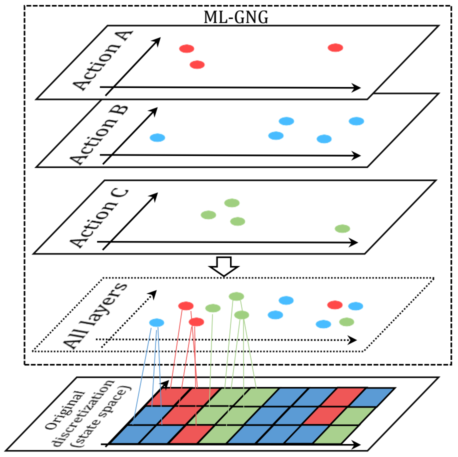


Fig. 2: Illustration of ML-GNG layers for a 2-dimensional state space and 3 actions. Nodes from all layers are projected to show how they link to the original sensor-based state space.

The time steps of Q-learning and ML-GNG are of different frequencies. Q-learning is updated after every action execution, while learning for the layer j of ML-GNG is triggered every θ number of times action a_j is executed in the same state. θ can be constant or gradually reduced as the learning converges. By initially being updated more frequently, Q-learning gains knowledge about action suitability in early visited states. Learnt behaviour is then propagated to ML-GNG, which generalizes actions to surrounding regions of the state space that have not yet sufficiently been explored in Q-learning. In this process, agents learn a simplification of the sensor-based state space and generate a coarser state space SL_k.

Each layer of ML-GNG (Line 7 in Algorithm 1) executes a modified GNG-U algorithm, presented in Algorithm 2.

The components of the ML-GNG algorithm are as follows:

- *Nodes* set N. Each node $u \in N$ represents a point (and its neighbourhood determined by ML-GNG, according to the metric distance) in the learnt state space where a particular action is suitable. The node position in the n-dimensional state space is denoted using a reference vector $\omega_u \in \mathbb{R}^n$. The *winner* node at every ML-GNG time step is defined as the node closest to the received input space signal, and the *second winner* node is the second closest. Each node u has an associated *error* e_u and *utility* v_u, which are used to inform the node addition and removal processes.
- *Edges* set E. Represents the connections between the nodes. We denote $E_u = \{l_1, \dots, l_x\}$, $E_u \subseteq E$, as the set of edges connecte to node u, where $l_1 = (u_1, u), \dots, l_x = (u_x, u)$. The nodes $u_1, \dots, u_x$ are the neighbours nodes of u. Each edge l has an associated age, $age(l)$, which represents how "outdated" an edge is.

The main steps of the GNG-U implementation in Algorithm 2 executed by each layer of ML-GNG are:

1) *Update* (Lines 2-9, 38-41): age, error, and utility of the winner node are updated. Errors of all nodes are

Algorithm 2 GNG-U: a single layer of ML-GNG

```
Input: State position $s$: location of the current state in sensor-
    based state space
Parameters: $\lambda$: the insertion parameter, $a_{max}$: the maximum
    edge age, $\alpha$: the error discount factor when creating a
    node, $\beta$: the general error discount factor, $k$: the utility
    factor, $\epsilon_b$: the adaptation factor for the winner node, $\epsilon_n$:
    the adaptation factor for its neighbours
1: procedure UPDATE($s$)
2:     if $|N| < 2$ then
3:         $N.add(u)$ ; $\omega_u := s$
4:     else
5:         $\{u', u''\}$ := NEARESTNODES($s$)
6:         MAP($E_{u'}$, for $l \in E_{u'}$ then $age(l) := age(l) + 1$)
7:         Update error and utility of $u'$:
8:         $e_{u'} := e_{u'} + ||\omega_{u'} - s||^2$
9:         $\nu_{u'} := \nu_{u'} + (||\omega_{u''} - s||^2 - ||\omega_{u'} - s||^2)$
    ▷ Adapt
10:        Move $u'$ towards $s$: $\omega_{u'} := \omega_{u'} + \epsilon_b(s - \omega_{u'})$
11:        for all $n$ neighbour of $u'$ do
12:            Move $n$ towards $s$: $\omega_n := \omega_n + \epsilon_n(s - \omega_n)$
13:        end for
14:        if $u'$ is neighbour $u''$ then
15:            Reset edge age: $age(u') := 0$, $age(u'') := 0$
16:        else
17:            Add an edge between $u'$ and $u''$: $E.add(u', u'')$
18:        end if
    ▷ Refine: GNG and GNG-U
19:        Remove old edges:
20:        FILTER($E$, for $l \in E$ if $age(l) < a_{max}$)
21:        Remove nodes without edges:
22:        FILTER(N, for $u \in N$ if $E_u \neq \emptyset$)
23:        $m$ := lowest utility node, $m \in N$
24:        $\overline{u}$ := highest error node, $\overline{u} \in N$
25:        if $e_{\overline{u}}$ / $\nu_m > k$ then
26:            Remove node $m$: $N := N \setminus \{m\}$
27:            Remove all edges of $m$:$E := E \setminus \{l\}, \forall l \in E_m$
28:        end if
    ▷ Add new nodes
29:        if number of input signals is a multiple of $\lambda$ then
30:            $\overline{u}$ := highest error node, $\overline{u} \in N$
31:            $f$ := neighbour of $\overline{u}$ with highest error, $f \in N$
32:            Add node $r$: $\omega_r := 0.5(\omega_{\overline{u}} + \omega_f)$; $N.add(r)$
33:            Add edges of node $r$: $E.add((r, \overline{u}), (r, f))$
34:            FILTER($E_{\overline{u}}$, for $(\overline{u}, x) \in E_{\overline{u}}$ if $x \neq f$)
35:            Update errors of $e_{\overline{u}}, e_f, e_r$:
36:            $e_{\overline{u}} := \alpha \cdot e_{\overline{u}}$ ; $e_f := \alpha \cdot e_f$ ; $e_r := e_q$
37:        end if
    ▷ Discount parameters
38:        for all Node $u \in N$ do
39:            Discount error of $u$: $e_u := \beta \cdot e_u$
40:        end for
41:    end if
42: end procedure
```

discounted.

2) *Adapt* (Lines 10-18): the winner node and all of its neighbours are moved closer towards the received signal.
3) *Refine* (Lines 19-28): the edges and nodes are removed to withdraw the old and inaccurate state representations. Edges are removed if they reach their maximum age, and nodes are removed if they are no longer connected to any edge or if a ratio of their error and utility has reached a threshold value.
4) *Add* (Lines 29-37): new nodes are added to the under-represented areas, *i.e.,* areas with the highest errors.

The output of ML-GNG is a policy (Line 10 in Algorithm 1) which gives the action to perform when the agent observes the state sr_j^k in the sensor-based space. Selection of the action to execute is done by projecting all nodes from all the GNG-U layers on the same space (as illustrated in the bottom of Figure 2). The action associated with the layer containing the closest node is selected.

The next action to be executed is then selected between the actions picked by both sensor-based and ML-GNG-generated states, using the selection process we describe in the next section.

C. Con-RL Agent: Run-time State Space Selection

An RL agent learns in the environment by observing the environment state, executing an action according to the policy learnt so far, and updating the suitability of the action based on the reward received from the environment. In Con-RL, the environment-agent interaction cycle is expanded to include state space learning using ML-GNG, a repository of state spaces containing sensor-based and learnt state spaces, and an *action selector* which picks a suitable action from all of the policies based on all available state spaces from the repository. This expanded cycle is shown in Figure 3.

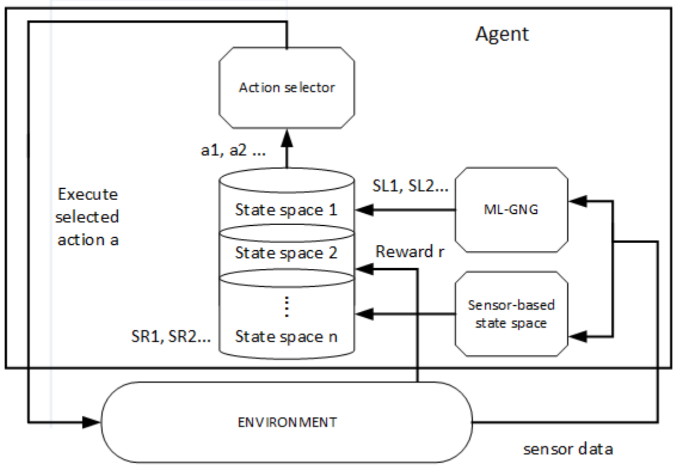


Fig. 3: Con-RL agent in the environment

The action selector implements two strategies. The simplest strategy uses a sensor-based state space in the early exploration stages until the number of nodes that ML-GNG contains reaches a threshold, from which point on ML-GNG is used. At first, during the exploration stage, the overall performance is not as accurate, but as the underlying sensor-based state space gains more experience, ML-GNG is able to generalize across it and converge quicker. In a more complex strategy, at each time step the selector considers the $confidence$ associated with actions learnt by the sensor-based state space and ML-GNG. In the sensor-based state space, $confidence$ denotes how many times the proposed action has been executed in the given state, relying on the assumption that more suitable actions are executed more frequently. In ML-GNG, $confidence$ reflects the distance between the nearest node and the received signal –that is, the closer the node, the more confidence ML-GNG will have that the suggested action is applicable to the position in the state space denoted by the received signal. If the confidence in the underlying sensor-based state is higher than the configurable parameter $threshold_{grid}$, then the action with the highest Q-value in that state is executed; if the nearest distance to the position of the current state in ML-GNG projected layer is closer than the configurable parameter $threshold_{mlgng}$, the action associated with the ML-GNG layer that the nearest point comes from is executed. If more than one state spaces exceeds the threshold, the one with the highest relative confidence (compared to this threshold) is used. If neither of the confidence values is higher than the threshold, a random action is executed giving an agent more exploration experience. More complex action selection approaches, such as those based on multi-objective action selection, or learnt by meta-learning processes, are further discussed in Section V.

Through the combination of ML-GNG and the state space selector, Con-RL fulfils the two design criteria: (i) a coarser state space is learnt from the original sensor-based state space, enabling run-time adaptation and optimization of the state space to improve convergence time, and (ii) consistent performance is maintained by switching between the available state spaces, based on which one has higher confidence.

Both the sensor-based state space and learnt state space are stored in a repository, and their learning processes benefit from one another as the quality of a suggested action by either updates the learning processes of both. Updates from the underlying sensor-based learning process are propagated to ML-GNG early on, so ML-GNG can converge without waiting on full policies to be learnt. Only the input space signal location and action information are propagated to the ML-GNG layer; unlike the state-of-the-art, Q-values do not need to be propagated as each action is represented by a separate layer in ML-GNG. A benefit of only propagating actions is that underlying learning processes can change (by, for example, implementing a different RL algorithm, or changing the scale of rewards therefore impacting the relative scale of Q-values) without requiring any change to the implementation of ML-GNG. Maintaining separate layers per action also allows for easy removal or addition of actions, either permanently, or by limiting certain actions to certain areas of the state space. If an action is not allowed at a particular point in time, the layer associated with that action will not be projected to the final layer from which the closest action is to be selected.

IV. Evaluation

This section evaluates the Con-RL approach to dynamic generation and adaptation of RL state spaces at run time in order to show that: (i) the proposed ML-GNG technique is able to learn a suitable optimized state space representation from sensor input, based on which it converges to a stable performance quicker than the state space based on sensor-provided granularity, and (ii) in the presence of changing state space representations, the selector can adapt to use the most suitable state space, maintaining a continuous stable performance.

We evaluate the approach in two case studies. A canonical single agent mountain car scenario is first used to examine the number of episodes required to reach a stable performance and evaluate the ability of the selector to maintain continuous stable performance, without the external elements affecting the environment (*e.g.,* environment changes due to impact of other agents, as present in multi-agent scenarios). Second, a multi-agent ride-sharing scenario is simulated in order to demonstrate the applicability of our approach in large-scale scenarios based on real data with multiple agents operating in a shared environment. For each scenario we compare the performance of a sensor-based state space with the performance of the ML-GNG learnt state space, and the performance of the state space selector using multiple available representations. As an additional baseline, in the mountain car scenario, we also compare our approach with GNG-Q [18], as the most closely related work. GNG-Q is not applicable in the complex car and ride sharing scenario, as its node adaptation process is only launched at the end of an episode, *i.e.,* only applicable in episodic learning. In car and ride sharing, the problem is a continuous learning process where the state space does not contain a terminal state, so it is not clear how and when would the node adaptation be performed.

A. Mountain Car

The problem set up for this classical RL benchmark and the parameters used are taken from Sutton [22] and listed in Table I. In this scenario, an agent must learn to drive a car up a steep mountain road. The car's engine is insufficient to do this, so it must learn to first reverse up the opposite hill to gain enough potential energy to complete the task. The agent aims to minimize the number of actions required to reach the top of the mountain (goal). I.

Parameter	**Value/Range**
State: Position	$[-1.2, 0.6]$ **(goal at 0.6)**
State: Velocity	$[-0.07, 0.07]$
Actions	Left -1, Neutral 0 or Right 1
Reward	100 if at the goal, -10 otherwise.

TABLE I: Mountain Car Parameters

The environment state is a combination of the agent's velocity and position, and we assume that this information comes from two sensors capable of sensing them. For the initial

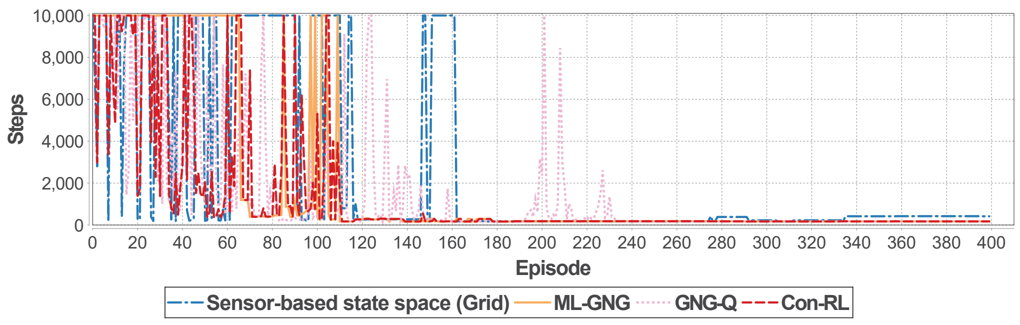


Fig. 4: Convergence of mountain car policy

sensor-based state space, we have arbitrarily divided values received from the sensors for both velocity and position in 10 equal slots, resulting in a 10x10 grid state space. Therefore, we are assuming a sensor granularity of 0.18 (*i.e.,* the full 1.8 range divided into 10 slots) for the position, and slots of 0.014 for the velocity. Hence, the sensor-based state space representation yields a grid composed of 100 states. The Con-RL approach is deployed at the start of system execution, in order to, according on this sensor-based state space, learn the most suitable coarser-grained state space representation. Please note that for simplicity of the evaluation this process is assumed to happen at the start of the execution. Con-RL is utilized in exactly the same way should new sensors for velocity and position become available at any point during system execution.

The Q-learning parameters used are: $\alpha = 0.1$, $\gamma = 0.9$, exponential decay selection policy, starting with $\epsilon = 1$ and updated according to the following formula: $\epsilon = \exp^{-Et}$ with $E = 0.015$ and where t corresponds to the current number of episodes. The ML-GNG parameters used are: $\lambda = 10$, $a_{max} = 200$, $\alpha = 0.5$, $\beta = 0.05$, $k = 1000$, $\epsilon_b = 0.5$, and $\epsilon_n = 0.1$.

GNG-Q parameters are based on the original paper and fine-tuned to optimize its performance in our case study, as follows: $\alpha = 0.1$, $\gamma = 0.95$, $\lambda = 1000$, $a_{max} = 100$, $\epsilon_b = 0.5$, and $\epsilon_n = 0.1$. The total number of nodes was capped to 20 (experimentally determined to give the best performance), as there is no specific mechanism stopping GNG-Q to grow (original authors manually stop node generation when the performance is satisfying).

We ran 100 experiments, with each experiment running for 1000 episodes, and each episode allowing a maximum of 100000 steps to reach the goal.

Figure 4 shows the performance (number of episodes to reach the goal, capped at 10000) for the sensor-based state space (*i.e.,* grid), ML-GNG, and complete Con-RL incorporating state space selector selecting between these two representations at each time step. Performance of the baseline GNG-Q is also plotted. The grid representation reaches its first good performance very early but the learnt policy fluctuates up until episode 160. The ML-GNG representation takes slightly longer to reach the first instance of a good performance, around episode 60, but it fully converges by episode 110, 50 episodes sooner than the grid.

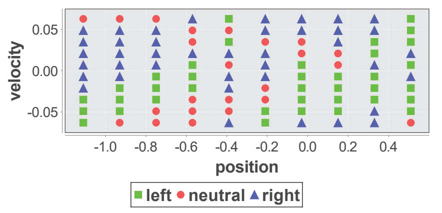
The baseline GNG-Q requires more time to converge, by episode 240, as adaption of the nodes is only launched when an episode completes. The Con-RL selector representation picks the best policy between the two representations (grid and ML-GNG) at each time step, achieving a stable performance by episode 110, without being negatively affected by the instability present in the grid-based state space up until episode 160.

Comparing the sensor-based state space (Figure 5a) and the ML-GNG learnt state space (Figure 5b), we see a significant reduction in the number of states (from 100 to 19) using ML-GNG, enabling a faster convergence as observed in Figure 4. Figure 5c shows the state space learnt in GNG-Q, with a total of 20 nodes after 1000 episodes. The three representations show clear similarities and have converged to similar policies, however, as observed in Figure 4, the baseline GNG-Q took significantly longer to reach that performance.

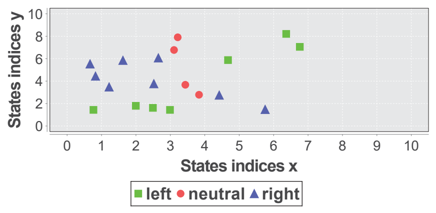
B. *Car and Ride-Sharing*

The second case study is based on a real world system: a car and ride-sharing decentralized system, such as those presented in [23]–[25]. The system consists of a fleet of shared autonomous vehicles, whose task is to serve ride sharing requests by customers. Each shared vehicle is controlled by an RL agent, which, in response to ride requests, picks up passengers and drives them to their destination, If there are no requests to serve, it can stay in its current area or relocates to zones with a higher number of requests. The car's goal is to maximize the number of requests served and its occupancy, while also minimizing riders' waiting time. We simulate 200 shared vehicles on the map of lower Manhattan and use the NYC taxi requests data [26] from 50 consecutive Tuesdays between July 2015 and June 2016. We simulate a total of 659,579 trips (1,074,690 passengers) in four periods: night-time (2am-5am), morning rush hour (7am-10am), midday (11am-2pm), and afternoon rush hour (6pm-9pm). As all the results follow a similar pattern, we only present results for the morning rush-hour (7am-10am) simulation.

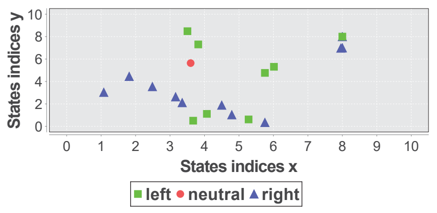
Details of each agent's initial three-dimensional state space, actions, and rewards are shown in Table II. The total size of the resulting sensor-based state space is 275 states, resulting from the 5 occupancy states $\times$ 11 own zone requests states $\times$



(a) Policy of sensor-based state space (grid)



(b) ML-GNG nodes position and actions



(c) GNG-Q nodes position and learnt actions

Fig. 5: Sensor-based (grid), ML-GNG and GNG-Q learnt state space policies (learnt actions)

Parameter	Value/Range
State: Occupancy	0,1,2,3,4 (**goal** $>$ **1**)
State: Req. in own zone	$0, 1, 2, \ldots, 10+$
State: Req. in neighb. zone	$0, 5, 10, \ldots$ 20+
Actions	pick up, rebalance, idle
Reward	100 at goal, 0 otherwise

TABLE II: Ride Sharing Parameters

5 states reflecting the sum of requests from all neighbouring zones. These parameters were fine-tuned for our previous work on ride-sharing assignment algorithms presented in [25].

The Q-learning parameters used are: $\alpha = 0.1$, $\gamma = 0.9$, and the selection policy is exponential decay, with $\epsilon = \exp^{-Et}$, $E = 0.0001$ and t is increased every environment interaction. The GNG-U parameters used are: $\lambda = 10$, $a_{max} = 200$, $\alpha = 0.5$, $\beta = 0.05$, $k = 1000$, $\epsilon_b = 0.5$ and $\epsilon_n = 0.1$.

In this case study, as it consists of 200 agents learning their state spaces independently, we do not inspect individual states as we did in the mountain car example, but focus on (a) sizes of the learnt state spaces per agent, observing any differences between them and their distribution, and (b) the impact of 200 agents implementing Con-RL on the overall large-scale system performance. As performance measures, we compare the overall percentage of requests served by the system (measure of system-wide performance), average waiting time until a request is fulfilled (measure of passenger-centric performance), and the average car occupancy per trip (measure of car-centric performance). We compare the performance of the sensor-based state space only (grid), and Con-RL, *i.e.,* the selector switching between ML-GNG generated state space and the grid. To show the progress of convergence of both approaches, we present the results after 5, 8, 10 and 15 days of training (*i.e.,* 5, 8, 10, 15 and 15 times 3 hours of the selected time period–7-10am). The results presented reflect the performance of the 200 vehicles during an additional day (3 hours), where they apply the policy learnt during the training period. The results are presented in Table III and represent the averages of 10 simulations for each experiment.

We observe that all three metrics show the same pattern: after only 5 days of training, grid serves only 53% of the requests and passengers wait more than 3 minutes on average, as, due to the size of the state space, the grid did not have sufficient time to explore all state-action combinations and learn adequate actions. Con-RL, which learns the ML-GNG state space and decides between the grid and ML-GNG at each time step, is significantly better, serving 74% of requests with a 2.807 minutes average waiting time, as it is able to generalize from limited grid experiences. The grid performance after 8, 10, and 15 days shows a consistent improvement from 72%, to 82%, and 88% respectively, while also gradually reducing the waiting time to 2.203 minutes after 15 days. Con-RL is still outperforming the grid, with respect to all metrics, after 5 and 8 days of training. However, after 10 and 15 days, as the grid has a longer period to explore the full state space, the differences in performance are not as clear cut. We therefore extract the results of those simulations in Figure 6 for further analysis.

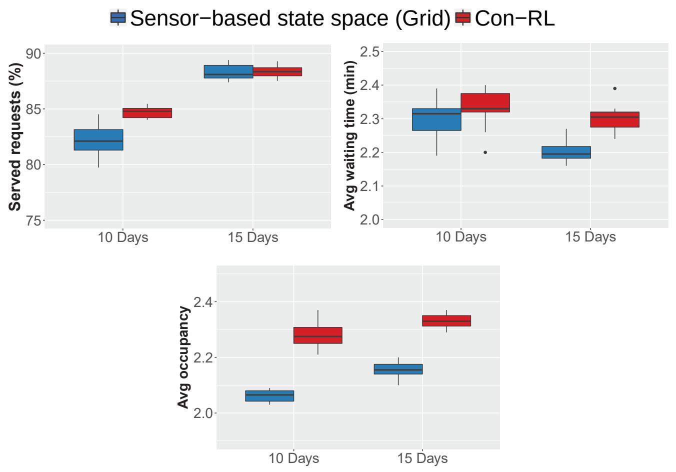


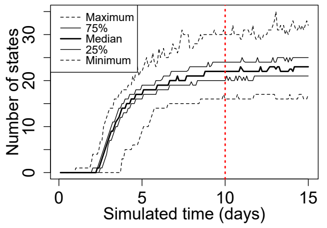
Fig. 6: Ride-sharing metrics distributions over 10 simulations

At 10 days, Con-RL performs better in terms of requests served and average occupancy, while grid achieves a shorter average waiting time. At 15 days, grid has had sufficient time to converge, and selector performs better only in terms of average occupancy, with a comparable number of requests served. The results across the 10 runs are consistent, with the standard deviation for requests served at 15 days being 0.5% for both grid and Con-RL, 0.04 for average waiting time for grid and 0.05 for Con-RL, and 0.02 for average occupancy for grid and 0.04 for Con-RL.

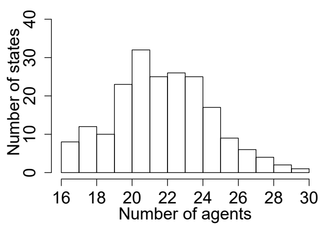
Figure 7 focuses on the size of the state spaces learnt by the agents (*i.e.,* the number of nodes in ML-GNG implemented by each agent, keeping in mind that the original sensor-

	5 days		8 days		10 days		15 days	
	Grid	Con-RL	Grid	Con-RL	Grid	Con-RL	Grid	Con-RL
Served requests (%)	52.898	73.692	71.625	80.324	82.201	84.703	88.26	88.367
Avg waiting time (min)	3.071	2.807	2.57	2.594	2.304	2.329	2.203	2.304
Avg occupancy	2.274	2.492	2.103	2.327	2.063	2.282	2.154	2.33

TABLE III: Ride-sharing metrics for sensor-based state space representation (Grid) and Con-RL



(a) Evolution of ML-GNG number of states



(b) Distribution of ML-GNG number of states after 10 days

Fig. 7: Distribution of 200 agents' state space sizes

based state space consists of 275 states). Figure 7a shows the distributions and evolution of agents' state space sizes as they were being learnt by their respective ML-GNG processes during the learning process. ML-GNG starts off with no states, but as the values from the underlying Q-learning process start propagating, we observe a steep increase in the number of states between day 2 and 5 (6-15 hours worth of learning data). Around day 8, the growth slows down and the number of states have converged around day 10.

Please note that the actual positions of these states still continue to be adjusted in order to ensure capturing any minor updates, but the number of states does not further increase, as the agents have learnt that this number has a sufficient granularity to adequately capture all the relevant environment information. Looking at the lines illustrating the minimum and maximum size of the state space, we also observe that size of the suitable state space differs per agent; we further inspect this in Figure 7b. Figure 7b shows a snapshot of the distribution of the state space sized per agent, after ML-GNG converged (*i.e.,* after 10 days). We see that the number of states per agent ranges from only 16 to 30, with a median of 22. These plots show that ML-GNG designs a specific state space adapted for each agent based on its interactions with the environment, and in combination with the improved performance results highlight the need for adaptive state space generation rather than manually predefining and hard-coding states by designers. The final performance is on a par with the original state space of size 275, but we have shown that state spaces around 1/10th of that size were able to achieve the same performance significantly quicker.

V. Conclusions and Open Issues

This paper presents Con-RL, an approach for autonomous RL state space learning and adaptation. Con-RL combines the novel ML-GNG technique for multi-layered clustering, used to learn optimized state space representations, and a run-time state space selector, used to select the most suitable state space representation to base the action decision on at each time step. We evaluated Con-RL in two scenarios: a single-agent mountain car scenario, and a multi-agent ride-sharing scenario. The results show an increase in the learning speed, based on learnt (optimized) state spaces, and a continued stable performance enabled by the state space selector.

By autonomously learning state space representations, Con-RL removes the need for manual state space specification; it is able to deduce the state space from the sensor input and reduce its size to a suitable granularity in order to optimize the learning time without sacrificing the precision of the learnt policy. In the mountain car example, we demonstrated a reduction of the state space size from 100 to approximately 20, achieving the same performance faster. ML-GNG-based state space converged to a stable performance by episode 110, while it took an additional 50 episodes for the sensor-based state space to stabilize to the same performance. Moreover, Con-RL maintained a stable performance through perturbations in both grid and ML-GNG learning processes by, at each time step, selecting an action in which it had the highest confidence, from the available multiple state space representations. Similarly, in the car and ride sharing scenario, Con-RL shows a drastic improvement in the speed of learning, serving 74% of the requests after being trained for 5 days (15 hours) worth of data. The grid-based state space required 10 additional days to reach the same level of service as ML-GNG, at which point, at day 15, they both serve 88% of the requests. Implications of this improvement are of significant importance for online systems relying on sensors to observe their environment; new sensors can be added at run time without manual interventions, and state space can be learnt and self-adapted with minimal negative effects on the run-time performance.

While we demonstrate the feasibility of Con-RL in this paper, we have also opened up numerous avenues for potential further improvements in the run-time generation and self-adaptation of state spaces. Scenarios with multiple sensors (and therefore multiple sensor-based and learnt state spaces), need to be evaluated, especially in the cases where multiple sensors are used to optimize towards multiple goals. In particular, more complex approaches to state space selection (*e.g.,* W-learning [20] or Pareto Q-learning [21]) can be used. Currently, Con-RL provides the techniques for generating a sensor-based state space and its optimization using ML-GNG, but it does not provide the capability of reusing learnt actions from previously ML-GNG-generated states in new ML-GNG processes (which could even further speed up the performance if a new sensor is added at run time). Techniques akin to transfer learning [27] could be incorporated in the approach to generate mappings between old and new sensor input granularities, in order to seed the new learning processes with existing knowledge. In this paper we evaluated Con-RL to learn state spaces based on moderately-sized starting sensor-based state spaces, *i.e.,* 2-dimensional (velocity and position) state space in the mountain car scenario of the overall size of 100, and 3-dimensional (occupancy, local requests, and neighbouring requests) state space in the car and ride sharing scenario of the overall size of 275. Further investigation is needed to evaluate the performance in state spaces with larger dimensions, and in particular the possible integration with Deep RL to optimize the number of inputs into neural networks representing the state spaces.

A significant advantage of Con-RL is that it removes the need for manual state space design, thereby removing the need for human expert knowledge and enabling online self-adaptation. However, the process could potentially be improved if it were guided by a non-expert end-user at run time. Interaction with users, by for example reward shaping mechanisms [28], could guide ML-GNG towards more significant areas of the state space sooner. A limitation of ML-GNG (as well as all GNG-based approaches) is that distances between input signals (states) need to be comparable because its state space optimization process rely on a direct comparison of the distances between the received signal and existing nodes. However, sensor data can also be obtained after it has been interpreted by a higher-level service returning more abstract values (*e.g.,* freezing, cold, warm, hot) that do not have measurable distances between them. An open issue for future investigation is whether the relationships between these inputs can be inferred or learnt from environment interactions to enable ML-GNG, and therefore Con-RL, to operate on abstract state spaces as well.

VI. Acknowledgements

We thank the anonymous reviewers for their input in earlier versions of this work. This publication is supported in part by a research grant from Science Foundation Ireland (SFI) under Grant Number 13/RC/2077 and 16/SP/3804 and by Irish Research Council through "Surpass: how shared autonomous cars will transform cities" New Horizons award.

References

[1] I. Dusparic, J. Monteil, and V. Cahill, "Towards autonomic urban traffic control with collaborative multi-policy reinforcement learning," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2016, pp. 2065–2070.

[2] M. Han, P. Senellart, S. Bressan, and H. Wu, "Routing an autonomous taxi with reinforcement learning," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 2016, pp. 2421–2424.

[3] I. Dusparic, A. Taylor, A. Marinescu, F. Golpayegani, and S. Clarke, "Residential demand response: Experimental evaluation and comparison of self-organizing techniques," *Renewable and Sustainable Energy Reviews*, vol. 80, pp. 1528–1536, 2017.

[4] A. Marinescu, I. Dusparic, and S. Clarke, "Prediction-based multi-agent reinforcement learning in inherently non-stationary environments," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 12, no. 2, p. 9, 2017.

[5] J. Piaget, *The Construction of Reality in the Child; translated by Margaret Cook*. Ballantine New York, 1954.

[6] F. Guerin, "Constructivism in AI: Prospects, progress and challenges," 2008, pp. 20–27.

[7] M. Guériau, F. Armetta, S. Hassas, R. Billot, and N. E. Faouzi, "A constructivist approach for a self-adaptive decision-making system: Application to road traffic control," in *28th IEEE International Conference on Tools with Artificial Intelligence*, 2016, pp. 670–677.

[8] R. Sutton, "Presentation: Mind and time: A view of constructivist reinforcement learning," 2008, 8th European Workshop on Reinforcement Learning.

[9] J. A. Martín H, J. de Lope, and D. Maravall, "Robust high performance reinforcement learning through weighted k-nearest neighbors," *Neurocomputing*, vol. 74, no. 8, pp. 1251 – 1259, 2011.

[10] D. Abel, D. E. Hershkowitz, and M. L. Littman, "Near optimal behavior via approximate state abstraction," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48*. JMLR. org, 2016, pp. 2915–2923.

[11] R. Jonschkowski and O. Brock, "Learning state representations with robotic priors," *Autonomous Robots*, vol. 39, no. 3, pp. 407–428, 2015.

[12] Y.-M. De Hauwere, P. Vrancx, and A. Nowé, "Learning multi-agent state space representations," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, vol. 1, 2010, pp. 715–722.

[13] K. Samejima and T. Omori, "Adaptive internal state space construction method for reinforcement learning of a real-world agent," *Neural Networks*, vol. 12, no. 7-8, pp. 1143–1155, 1999.

[14] M. Abramson, P. Pachowicz, and H. Wechsler, "Competitive reinforcement learning in continuous control tasks," in *Proceedings of the International Neural Network Conference*, 2003.

[15] B. Fritzke, "A growing neural gas network learns topologies," in *Advances in neural information processing systems*, 1995, pp. 625–632.

[16] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[17] B. Fritzke, "A self-organizing network that can follow non-stationary distributions," in *International conference on artificial neural networks*. Springer, 1997, pp. 613–618.

[18] M. Baumann and H. Kleine Büning, "State aggregation by growing neural gas for reinforcement learning in continuous state spaces," in *10th International Conference on Machine Learning and Applications and Workshops*, vol. 1, 2011, pp. 430–435.

[19] D. C. D. L. Vieira, P. J. L. Adeodato, and P. M. Goncalves, "A temporal difference gng-based approach for the state space quantization in reinforcement learning environments," in *IEEE 25th International Conference on Tools with Artificial Intelligence*, Nov 2013, pp. 561–568.

[20] M. Humphrys, "W-learning: Competition among selfish q-learners," Tech. Rep., 1995.

[21] K. Van Moffaert and A. Nowé, "Multi-objective reinforcement learning using sets of pareto dominating policies," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 3483–3512, Jan. 2014.

[22] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. MIT Press, 1998.

[23] Oisin Dolphin, "AI has proven itself at the poker table but can it prove itself in car-sharing technology?" 2017. [Online]. Available: http://goodtravelsoftware.com/blog/

[24] J. Wen, J. Zhao, and P. Jaillet, "Rebalancing shared mobility-on-demand systems: A reinforcement learning approach," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, Oct 2017, pp. 220–225.

[25] M. Guériau and I. Dusparic, "SAMoD: Shared autonomous mobility-on-demand using decentralized reinforcement learning," *21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1558–1563, 2018.

[26] NYC Taxi and Limousine Commission, "Tlc trip record data," 2018. [Online]. Available: http://www.nyc.gov

[27] A. Taylor, I. Dusparic, M. Guériau, and S. Clarke, "Parallel transfer learning in multi-agent systems: What, when and how to transfer?" in *International Joint Conference on Neural Networks (IJCNN)*, 2019, p. in press.

[28] H. B. Suay, T. Brys, M. E. Taylor, and S. Chernova, "Reward shaping by demonstration," in *Proceedings of the Multi-Disciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, 2015.