A Reinforcement Learning Approach for Interdomain Routing with Link Prices

PETER VRANCX and PASQUALE GURZI, Vrije Universiteit Brussel
ABDEL RODRIGUEZ, Vrije Universiteit Brussel and Universidad Central Marta Abreu de las Villas
KRIS STEENHAUT and ANN NOWÉ, Vrije Universiteit Brussel

In today's Internet, the commercial aspects of routing are gaining importance. Current technology allows Internet Service Providers (ISPs) to renegotiate contracts online to maximize profits. Changing link prices will influence interdomain routing policies that are now driven by monetary aspects as well as global resource and performance optimization. In this article, we consider an interdomain routing game in which the ISP's action is to set the price for its transit links. Assuming a cheapest path routing scheme, the optimal action is the price setting that yields the highest utility (i.e., profit) and depends both on the network load and the actions of other ISPs. We adapt a continuous and a discrete action learning automaton (LA) to operate in this framework as a tool that can be used by ISP operators to learn optimal price setting. In our model, agents representing different ISPs learn only on the basis of local information and do not need any central coordination or sensitive information exchange. Simulation results show that a single ISP employing LAs is able to learn the optimal price in a stationary environment. By introducing a selective exploration rule, LAs are also able to operate in nonstationary environments. When two ISPs employ LAs, we show that they converge to stable and fair equilibrium strategies.

Categories and Subject Descriptors: C.2.3 [**Computer-Communication Networks**]: Network Operations; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Interdomain routing, link pricing, learning automata

ACM Reference Format:
Peter Vrancx, Pasquale Gurzi, Abdel Rodriguez, Kris Steenhaut, and Ann Nowé. 2015. A reinforcement learning approach for interdomain routing with link prices. ACM Trans. Autonom. Adapt. Syst. 10, 1, Article 5 (March 2015), 26 pages.
DOI: http://dx.doi.org/10.1145/2719648

1. INTRODUCTION

The current Internet is a highly commercial medium where business interests of Internet service providers (ISPs) influence interdomain routing policies and create a fertile

P. Gurzi and P. Vrancx have equal contribution.
Some elements of this research appeared in a preliminary version in *Proceedings of the 18th IEEE Workshop on Local and Metropolitan Networks (LANMAN '11)*.
This work is funded by the National Fund for Scientific Research (FWO–Flanders) under grant FWO G.0291.09N: QoS support in heterogeneous wireless sensor networks using distributed reinforcement learning techniques.
Authors' addresses: P. Vrancx, P. Gurzi, A. Rodríguez, and A. Nowé, AI-Lab, Computer Science Department, Vrije Universiteit Brussel, Pleinlaan 2, B1050 Brussels, Belgium; emails: peter.vrancx@vub.ac.be, pasquale.gurzi@alcatel-lucent.com, abdel.rodriguez@reduc.edu.cu, ann.nowe@vub.ac.be; K. Steenhaut, Electronics and Informatics Department, Vrije Universiteit Brussel, Pleinlaan 2, B1050 Brussels, Belgium; email: kris.steenhaut@vub.ac.be.

DOI: http://dx.doi.org/10.1145/2719648

ground for transit market growth. An ISP operator has the opportunity to increase its revenue by charging external domains for the traffic transiting on its links.

Conventional fixed pricing strategies are simple to implement but may be inefficient, as ISPs can lose profit because they incur high variable costs when routing external traffic. On the other hand, customer networks may have incentives to direct peering with an Internet exchange point (IXP) if they find it cheaper to reach an IXP by purchasing a private link [Labovitz et al. 2010; Valancius and Lumezanu 2011]. To retain profit, ISPs are interested in improving their business models by using adaptive pricing strategies. To simplify the model, we can assume that when several route alternatives are available for interdomain connections, routing algorithms choose the most cost efficient one among those that meet their requirements. This way, there is healthy competition between domain operators to get as much interdomain traffic as they can route while increasing their profit.

The transit market model described earlier can be treated as a multiagent Markov decision process (MMDP) with continuous action spaces. In fact, each ISP, represented by a different agent, selects an action by setting the price for using its links. The resulting set of actions of all ISPs determines the transition behavior of the MMDP whose state is defined by the amount of traffic crossing the links. Recently, a partially decentralized reinforcement learning approach for finite MMDP was presented in Tilak and Mukhopadhyay [2011]. However, a major difference in our approach is that ISPs are not allowed to communicate and explicitly coordinate their actions to form cartels. Moreover, agents must be able to deal with the nonstationary environment when the optimal price setting varies according to other ISPs' strategies and to the network load.

In this article, we enable ISPs to use independent learning automata (LAs) to dynamically learn the best action (link prices) to play. More specifically, we adapt a finite action learning automata (FALA) and a continuous action reinforcement learning automata (CARLA) to operate in such environment. To avoid the need for central coordination or sensitive information exchange among domains, we propose a reinforcement signal that depends only on the ISP's utility. This article builds on previous work by the authors, which investigated the possible use of CARLA to learn link prices for fixed network conditions [Gurzi et al. 2011]. To deal with the nonstationary environments, we also present a particular exploration rule that improves the agent's ability to react to the changing network conditions.

Through simulation results, we show the advantage of using an LA and compare the two proposed algorithms in this setting. In particular, we evaluate three scenarios:

—*One ISP learns in a stationary environment*: In this case, we want to discover if an ISP is able to find the optimal price for fixed load conditions when competitors use a fixed pricing strategy. Our analysis shows that both LAs are able to find the optimal price and maximize the ISP's profit.
—*One ISP learns in a nonstationary environment*: We check the ability of the LAs to adapt their pricing strategy when the network condition changes. In this case, we also show the effectiveness of our exploration rule.
—*Two ISPs learn*: When two ISPs learn simultaneously, we again have a nonstationary environment since the price setting of one ISP influences the strategy of the other one. In this case, we show through simulation how the two ISPs are able to converge to a stable solution that represents an equilibrium strategy for the considered game.

In the following, the interdomain routing model with link prices is discussed in Section 3, whereas the pricing model is presented in Section 4. In Section 5, we introduce the two LAs considered in this work, the reward functions, and the exploration rule. In Section 6, we present a test network topology and show through simulation the advantage of using LAs. Finally, in Section 8, we draw the conclusions.

2. RELATED WORK

Loja et al. [2005] study the dependency of the domain income on virtual link (VL) prices assuming cheapest-path routing. The paper examines some basic interdomain topologies and determines the Nash equilibria and Pareto efficiency when the action set is finite. However, it does not consider internal costs or provide any learning algorithm capable of dynamically choosing link prices.

Recently, Shrimali et al. [2010] used the Nash bargaining concept to model an interdomain ISP peering problem. Under this scheme, ISPs use dual decomposition to jointly optimize a social cost function referred to as Nash product. The paper analyzes the case where two ISPs have to send each other traffic flows and want to maximize their utilities. The utilities are assumed to depend on the network load and be related to some measure of the network performance that may be different for the two ISPs. The authors show how to find the optimal flow distribution maximizing the Nash product of the utility functions. However, this scheme only considers the peering relationship between two ISPs and is not suited for a transit market (i.e., ISPs compete to attract external traffic flow not destined to nodes in their own network) where coordination between ISPs is limited.

Valancius and Lumezanu [2011] analyze the effects of tiered pricing and develop a model that captures demands and costs in the transit market. The main idea of the paper is to predict how much traffic (and hence ISP profit) would change in response to different pricing strategies. In this work, we take a converse approach. We show how to learn the optimal pricing strategy by taking the ISP profit as an input. In this sense, the work that is perhaps the closest to ours, in its goal, is described in Barth and Echabbi [2008]. The authors present a distributed learning algorithm for optimizing transit prices in an interdomain routing framework. However, there are important differences with our work. First, they consider a discrete and finite action set; second, they make different assumptions on the interdomain model by not taking into account the price of interdomain links; and third, the price model proposed does not take into account internal costs of the routing service. However, it is interesting to notice that the convergence of the learning algorithm to Nash equilibrium can be proven (but not ensured), under certain conditions, when considering a finite action set.

On the other hand, when considering games with a continuous action space, the recent literature on evolutionary dynamics shows that generalizing results from games with a finite action space is not always straightforward and sometimes not even possible [Cressman 2005; Oechssler and Riedel 2002]. An important part of the theory for games with a continuous action space consists of connecting static properties that a strategy can have, such as evolutionary stability, the existence of a uniform invasion barrier, and local superiority to the notion of asymptotic stability in the replicator dynamics. Veelen and Spreij [2008] investigate these properties and show the most important relationships. Further examples of learning approaches with multiple agents can be found in Barrett et al. [2014], Panait and Luke [2005], and Buşoniu et al. [2010].

3. INTERDOMAIN MODEL

The Internet consists of a large number of interconnected domains often identified as autonomous systems controlled by economic entities called *Internet service providers* (ISPs). For scalability reasons, and to protect sensitive information, the ISPs' physical topologies are only represented at an intradomain level, whereas an aggregated version of the domain is advertised to external domains.

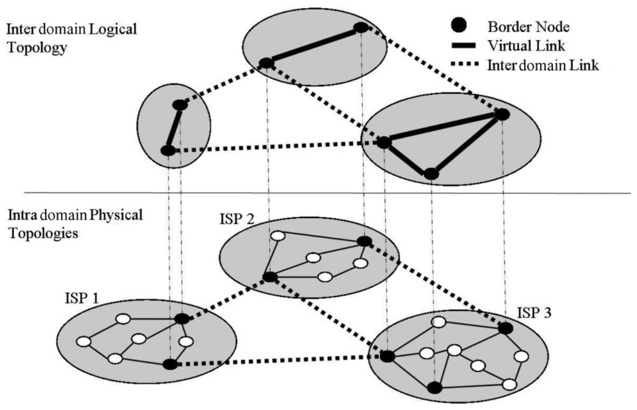

Inter domain Logical Topology
Border Node
Virtual Link
Inter domain Link
Intra domain Physical Topologies
ISP 2
ISP 1
ISP 3

Fig. 1. Interdomain network model.

Karaoglu and Yuksel [2010] define a contract routing model where a single domain is represented by a virtual topology consisting of border routers connected by virtual contract links. This is a significantly different approach from the existing routing architecture that abstracts each ISP as a single node in the interdomain topology, a choice mainly motivated by the need to hide internal network information. In the following, we consider a similar model represented in Figure 1. The VLs connecting the ISPs' border routers summarize the connectivity status and are periodically advertised to external domains. Interdomain routing is thus performed hierarchically by first searching a path in the interdomain virtual topology and then replacing each VL with an intradomain path.

To model a free transit market in an interdomain framework, we make the following assumptions:

—Routing an external flow on a certain link has a cost for the owner ISP.
—ISPs choose the VL prices to generate income.
—Interdomain links are shared among domains, their prices are equal to the routing costs, and thus they do not generate income.
—The price of the path is the sum of the prices of all traversed links.
—Customers prefer the cheapest path.

According to this model, ISPs can leverage the link price to increase their income, but on the other hand, when several route alternatives are available, customers choose the most cost-efficient one. This way, there is a competition between ISPs who choose VL prices to get as much interdomain traffic as they can route as long as they can increase their utilities. We should also note that the transit market is influenced by the network load. In an overloaded network, customers do not have many alternatives for routing the paths, and ISPs are able to increase their price. On the other hand, a low load network allows for more competition and price reductions.

4. PRICING MODEL

Although we are aware of the difficulties of modeling prices and costs in an interdomain network, we present a model that incorporates the most important characteristics of the transit market. Recent work presented in Valancius and Lumezanu [2011] provides an interesting and insightful overview on the most used pricing strategies in the wholesale transit market. The authors outline how most ISPs sell connectivity at a blended rate, usually expressed per time and per used capacity (e.g., $\$/Mbps/Day$). However, since the ISP's internal costs are highly variable, the less costly flows in the blended-rate bundle must subsidize the more expensive flows to realize profit. An ISP who wants to innovate should offer different rates according to the network cost variations. Intuitively, a pricing mechanism that charges individual flows is more efficient than one that charges flows in bulk, as customers pay only for the traffic they send, and ISPs can price those flows according to their impact on the internal cost.

In the next sections, we see how to model the ISP's utility and costs by considering the contribution due to each single flow transiting the domain.

4.1. Utility Model

The utility (or profit) of an ISP is identified as the difference between the total income and the internal costs of the routing service. We refer to these routing costs as traffic-dependent costs (TDCs) to stress the dependency on the network load. An ISP is willing to route external traffic on its VLs only if the monetary revenue covers at least the TDC. ISPs can increase link prices to increase their utilities, but if the total cost is too high, the traffic will be routed on cheaper paths (i.e., via other ISPs). A flow utilizing the link pays the current price at the time the connection is established. This means that further price changes do not affect the traffic flows that have already been established.

Evidently, there is a trade-off between the choice of the price and the amount of transit traffic received. The optimal cost also depends on the TDC and consequently on the resource usage in the network.

The income of an ISP is defined by the amount of interdomain traffic sent to any of its VLs multiplied by the price paid when it entered the domain. Let $\phi_l(t)$ be the number of flows crossing link l and $x(t)$ be the price of the links at time t. The total ISP income at time t is given by the following equation:

$$R(t) = \sum_{l \in \mathrm{ISP}} \sum_{n_l=1}^{\phi_l(t)} x(T_{n_l}), \tag{1}$$

where T_{n_l} is the arrival time of the nth flow crossing link l. The utility of an ISP is the difference between the income and the TDC cost used to cover the service provisioning cost.

$$U(t) = R(t) - \mathrm{TDC}(t) \tag{2}$$

4.2. Cost Model

To define the TDC of a VL, we assume that the internal cost depends on the links' load. Intuitively, the cost for providing a service increases with the traffic load because the work load at switching points increases, and it becomes more difficult to provide the same quality of service. To model the TDC, we can calculate the marginal cost $M(n_l)$ imputed to the nth flow crossing link l in the domain. The TDC is the sum of all marginal costs of the flow crossing the domain.

$$\mathrm{TDC} = \sum_{l \in \mathrm{ISP}} \sum_{n_l=1}^{\phi_l(t)} M(n_l) \tag{3}$$

We propose using a marginal cost that increases with the load. The marginal cost of the first flow entering the domain is lower than the cost associated with the last flow. Let $C_{max,l}$ be the maximum capacity of the link and $C_{us,l}(T_{n_l})$ be the used capacity at time instant $t = T_{n_l}$ when the nth flow enters the link l

$$M(n_l) = \frac{1 + C_{us,l}(T_{n_l})}{1 + C_{max,l}}. \tag{4}$$

The marginal cost is normalized to the maximum capacity $C_{max,l}$ and varies in the interval $(1 + C_{max,l})^{-1} \leq M(n_l) < 1$, because when $C_{us,l} = C_{max,l}$, the link cannot accept any other traffic flow and $M(n_l) = 1$. The TDC fluctuates according to the network load and resource demand. Since interdomain links also have a routing cost but do not belong to any ISP, the price of interdomain links has been set to be equal to their TDCs. According to this model, an ISP is willing to rent out its links only if $U(t) > 0$, and thus $\sum_{l \in \text{ISP}} \sum_{n_l=1}^{\phi_l(t)} (x(T_{n_l}) - M(n_l)) > 0$.

5. LEARNING THE LINK PRICES

Considering that the objective of any ISP is to maximize its utility over time, we propose two different learning schemes able to learn the optimal price using only the ISP's internal information. Two main classes of LAs exist. FALA use a finite action set where each action is associated with a probability. Continuous-action learning automata (CALA instead use a continuous action set, and the probability of selecting an action is replaced with a probability density function (PDF) over the action set. We implement the *Pursuit* estimator algorithm for the FALA class and the *CARLA* algorithm for the CALA class. The two LAs are described in Sections 5.1 and 5.2, respectively. The reward functions used for both type of algorithms are discussed in Section 5.3. A selective exploration rule introduced to improve the performance of the two LAs is presented in Section 5.4.

5.1. Pursuit Algorithm

Estimator algorithms are a class of learning algorithms where the update of the action probabilities depends on the past history of action and rewards. These algorithms make use of estimates of characteristics of the random environment in which FALA operate. The idea is to use these estimates, computed online, to improve the performance in terms of speed and accuracy. Here we describe Pursuit, one of the most important estimator algorithms, which was first introduced in Thathachar and Sastry [1986]. Let $\mathcal{A} = \{a_1, a_2, \ldots, a_r\}$ be the action set of the automaton, and let

$$\mathbf{p}(k) = \{p_1(k), p_2(k), \ldots, p_r(k)\}$$

be the action probability vector at instant k. Since $p_i(k)$ is the probability with which action a_i is chosen at instant k, we have that $p_i(k) \in [0, 1]$ and $\sum_{i=1}^{r} p_i(k) = 1$.

The Pursuit algorithm estimates the expected rewards associated with every action. Let d_i be the mean value of the reward probability distribution associated with action a_i, and let $\hat{d}_i(k)$ be its estimation at instant k. The basic approach of the Pursuit algorithm is the so-called certainty-equivalence principle. Let $\hat{d}_M(k)$ be the highest estimated reward at instant k, and let $\mathbf{e}_l$ an r-dimensional vector whose lth component is 1 and all others are 0. If we consider the estimates to be the true value, then we should set $\mathbf{p}(k+1) = \mathbf{e}_M(k)$. However, since the estimates could be erroneous, the algorithm moves $\mathbf{p}(k)$ toward $\mathbf{e}_M(k)$ by a small amount determined by the learning parameter. Thus, the action probability pursues the estimated optimal vector. To obtain the estimates, the algorithm maintains two more vectors. The vector $(\eta_1(k), \ldots, \eta_r(k))$ stores the number of times that actions $a_1, \ldots, a_r$ are chosen up to instant k, whereas $(Z_1(k), \ldots, Z_r(k))$ gives the the total reinforcement obtained for each action up to time k. Let $a(k) = a_l$ be the action selected at time k, and let $\beta(k)$ be the resulting feedback. Then $Z_i(k)$, $\eta_i(k)$

and the estimates $\hat{d}_i(k)$ are updated as follows:

$$\begin{aligned} Z_l(k) &= Z_l(k-1) + \beta(k) \\ Z_j(k) &= Z_j(k-1)\ \forall j \neq l \\ \eta_l(k) &= \eta_l(k-1) + 1 \\ \eta_j(k) &= \eta_j(k-1)\ \forall j \neq l \\ \hat{d}_i(k) &= \frac{Z_i(k)}{\eta_i(k)}\ \forall i = 1, \ldots, r. \end{aligned} \tag{6}$$

As shown in Equation (6), $\hat{d}_i(k)$ gives the average reward obtained with action a_i up until instant k. After the estimation step, the action probability vector is updated as follows:

$$\mathbf{p}(k+1) = \mathbf{p}(k) + \lambda(\mathbf{e}_M(k) - \mathbf{p}(k)), \tag{7}$$

where $0 < \lambda < 1$ is again the learning rate and the index M is determined by

$$M(k) = \arg\max_i \hat{d}_i(k). \tag{8}$$

An interesting aspect of the Pursuit algorithm is that the updating of $\mathbf{p}(k)$ does not directly involve $\beta(k)$. For this reason, $\beta(k)$ does not need to be constrained in the interval [0, 1].

One particular advantage of the Pursuit algorithm is that it is generally faster than linear algorithms like L_{R-I} but retains the most important properties, such as the ϵ-optimality in stationary random environments [Thathachar and Sastry 2004].

5.2. CARLA Algorithm

In the preceding discussion, we assume that the action set (i.e., the set of possible price points) is discrete. In some cases, we may not want to restrict ourselves to a discrete set of prices but instead search for the optimal price in the whole continuous range of possible prices. A natural question that can be asked then is whether it is possible to use a nonparametric PDF to represent the action probability distribution to deal with these continuous action sets. The CARLA algorithm, introduced for applications involving searching in a continuous action space in a random environment [Howell et al. 1997], provides an answer to this question. The action set of CARLA is a continuous interval $\mathcal{A} = [x_{min}, x_{max}]$. Every time the automaton chooses an action based on its current action PDF, it gets a reinforcement from the environment and, using it, updates its PDF. Let $f_k(x)$ be the PDF for choosing an action at instant k. A common initialization of $f_0(x)$ is the uniform distribution over $\mathcal{A}$. However, domain knowledge can be encoded in this initial distribution so that the exploration focuses on certain regions. Let $a(k)$ and $\beta(k) > 0$ denote the action selected and the reinforcement obtained at k. The action PDF is then updated as follows:

$$\begin{aligned} f_{k+1}(x) &= \begin{cases} \gamma\,[f_k(x) + \lambda\beta(k)H_k(x)] & \text{if } x \in \mathcal{A} \\ 0 & \text{otherwise} \end{cases} \\ H_k(x) &= e^{\left(-\frac{(x-a(k))^2}{2\sigma^2}\right)}. \end{aligned} \tag{9}$$

In the preceding equations, $H_z(\cdot)$ is a symmetric Gaussian neighborhood function that spreads the effect of the current reinforcement around the current action, λ is again the learning rate, and σ is a parameter known as the spreading rate, used to control the extension of the neighborhood around $a(k)$ that is going to be reinforced by the last observation $\beta(k)$. Since $f(\cdot)$ is a PDF, it has to be normalized after adding the reinforcement. The normalization is carried out by the factor γ, being a constant calculated at every iteration to meet the following constrain $\int_{x_{min}}^{x_{max}} f_{k+1}(x)dx = 1$, so it can be then expressed as

$$\gamma = \frac{1}{1 + \int_{x_{min}}^{x=a(k)} \lambda\beta(k)H_k(x)dx}. \tag{10}$$

As the iterations proceed, the automaton essentially adds a small Gaussian bell (weighted by the reinforcement) on top of the PDF at every selected action. Actions yielding higher reinforcements will then accumulate higher probabilities resulting in eventual convergence to the optimal actions. The action selection mechanism is the generation of actions according to the current distribution $f(\cdot)$. To do so, a uniformly distributed pseudo-random number z is generated in the range $[0, 1]$ and then the inverse of the cumulative density function is evaluated on z to generate $a(k)$:

$$\int_{x_{min}}^{x=a(k)} f_k(x)dx = z. \tag{11}$$

Through experimental results, it has been demonstrated that this updating essentially results in the action PDF eventually peaking at an x where the expected value of the reinforcement is at its maximum. The initial PDF can be chosen as being uniform over a desired range, and over many iterations, this converges to a Gaussian distribution around the best action value. Convergence of CARLA has been proven for single-agent [Rodríguez et al. 2011] and game settings [Rodriguez et al. 2012a, 2012b].

5.3. Reward Function

Since an ISP can only access its internal information, the reward function is constructed using only information on its utility $U(t)$. Both algorithms will construct the reward from a score function calculated at every learning step k. The score function J_k at the kth iteration is calculated using the sliding window mean of the utility, as stated in the following equation where W is the window size:

$$J_k = \frac{1}{W}\int_{t-W}^{t} U(t)dt. \tag{12}$$

The main idea here is to give positive rewards to actions that yielded utility increment. The difference in the score function between two consecutive iterations, $J_k - J_{k-1}$, is proportional to this increment and is negative when the utility has decreased from the previous iteration. Given the noisy behavior of the utility function, the sliding window mean is used to filter the noise and get an average-based estimate.

Using the score function defined earlier, the reward function $\beta(t)$ that we propose for the Pursuit algorithm can be expressed as follows:

$$\beta_k^{Pursuit} = \frac{1}{x_k}(J_k - J_{k-1}), \tag{13}$$

where x_k is the action (price) selected at the kth iteration. Since high prices will produce large differences in the utility function, this normalization is used to make the reward amplitude independent from the actual action value. We recall from Equations (6) through (8) that Pursuit algorithms do not use the reward to directly update the action probabilities. For this reason, the reward function is not required to be bounded in the interval $[0, 1]$. The CARLA algorithm instead operates a *reward-inaction* rule, and the discrete probability distribution is replaced with a continuous PDF. In this case, negative rewards are not allowed, as they will compromise the PDF updating process.

The reward function of CARLA can be expressed as follows:

$$\beta_k^{CARLA} = max\left[0,\ \frac{1}{x_k}(J_k - J_{k-1})\right]. \tag{14}$$

Next we describe a selective exploration rule that has been designed to improve the dynamic adaptation of the two proposed LAs.

Algorithms 1 and 2 describe the complete process of an ISP using CARLA and Pursuit, respectively.

ALGORITHM 1: CARLA Price Learning (step *k*)

```
Require:
        $f_{k-1}(x) \leftarrow$ Action Probability Density Function
        $J_{k-1} \leftarrow$ Score Function
        $x_{k-1} \leftarrow$ Price
        $\mathbf{r}_{k-1} \leftarrow$ Reward Vector
Ensure: Link Price $x_k$
 1: $J_k \leftarrow \frac{1}{W}\int_{t_k-W}^{t_k} U(t)dt$
 2: $\beta_k \leftarrow \frac{1}{x_{k-1}}(J_k - J_{k-1})$
 3: if $\beta_k > 0$ then
 4:     $f_k(x) \leftarrow \alpha\Big(f_{k-1}(x) + \beta_k \lambda e^{-\frac{(x-x_{k-1})^2}{2\sigma^2}}\Big)$
 5:     if $(\beta_k \notin CI_{99\%}) \wedge (\mu(\mathbf{r}_{k-1}) = \sigma(\mathbf{r}_{k-1}) = 0)$ then
 6:         $x_k \leftarrow$rand($\frac{1}{x_{max}-1}$) Exploration
 7:     else
 8:         $x_k \leftarrow$rand($f_k(x)$) Select an Action using the Action PDF
 9:     end if
10: else
11:     $f_k(x) = f_{k-1}(x)$
12:     $x_k \leftarrow$rand($f_k(x)$) Select an Action using the Action PDF
13: end if
14: $\mathbf{r}_k \leftarrow$pushBack($\beta_k$)
15: return $x_k$
```

5.4. Selective Exploration

One of the difficulties of using independent LAs in a multiagent environment is that the optimal action itself is not fixed in time. The optimal price, in fact, depends on the network load (which directly influences the utility function) and on the prices chosen by other ISPs. We cannot assume that the network load is fixed over time or that other ISPs always play the same link price. For these reasons, our LA must be able to track changes in the objective function and move toward different optimal prices. If the changes are confined in a small interval, the algorithm itself is able to track the new solution. However, when the changes are large, exploration must be carried out over the entire action space; otherwise, the convergence to the new solution will be very slow or may not be possible at all.

To allow the LA to dynamically adapt its action without reinitializing the automaton, we need to introduce some form of exploration. A well-known exploration policy is the ϵ-greedy strategy. When using this rule, the best action is selected with probability $1-\epsilon$, and another action is randomly selected (with uniform distribution) with probability ϵ. One advantage of this strategy is that the entire action space is explored, even when the automaton is close to the final solution, reducing the probability of being trapped

ALGORITHM 2: Pursuit Price Learning (step k)

```
Require:
        $A = (a_1, \dots, a_n) \leftarrow$ Action Set
        $\mathbf{p}(k-1) \leftarrow$ Action Probability Vector
        $\mathbf{Z}(k-1) \leftarrow$ Total Obtained Reward Vector
        $\eta(k-1) \leftarrow$ Action Occurrences Vector
        $x_{k-1} \leftarrow$ Price
        $J_{k-1} \leftarrow$ Score Function
Ensure: Link Price $x_k$
 1: $i \leftarrow j = 1, \dots, n : a_j = x_{k-1}$
 2: $J_k \leftarrow \frac{1}{W} \int_{t_k - W}^{t_k} U(t)dt$
 3: $\beta_k \leftarrow \frac{1}{x_{k-1}}(J_k - J_{k-1})$
 4: $Z_i(k) = Z_i(k-1) + \beta_k$
 5: $\eta_i(k) = \eta_i(k-1) + 1$
 6: $m \leftarrow j : \frac{Z_j(k)}{\eta_j(k)} \geq \frac{Z_w(k)}{\eta_w(k)} \forall w \neq j$
 7: $\mathbf{e}_m = (e_0, \dots, e_j, \dots, e_n) : e_j = 0 \, \forall j \neq m, \; e_j = 1$ for $j = m$
 8: $\mathbf{p(k)} = \mathbf{p(k-1)} + \lambda(\mathbf{e}_m - \mathbf{p(k-1)})$
 9: if $(\beta_k \notin CI_{99\%}) \wedge (\mu(\mathbf{r}_{k-1}) = \sigma(\mathbf{r}_{k-1}) = 0)$ then
10:     $x_k \leftarrow$rand$(1, n)$ Exploration
11: else
12:     $x_k \leftarrow$rand$(\mathbf{p(k)})$ Select an Action using the Action Probability Vector
13: end if
14: $\mathbf{r}_k \leftarrow$pushBack$(\beta_k)$
15: return $x_k$
```

in a local optimum. On the other hand, exploration slows down the convergence speed of the algorithm when it is not needed. Moreover, the parameter ϵ must be tuned according to the environment type and the learning algorithm.

Our intuition is that the exploration should be as selective as possible to avoid waste of utility and to not compromise the convergence speed. To implement a selective exploration, we propose the following strategy. Let $\mathbf{r}_k$ be a vector with the last N received rewards

$$\mathbf{r}_k = \{\beta_{k-N+1}, \dots, \beta_{k-1}, \beta_k\}, \tag{15}$$

and let $\mu(\mathbf{r}_k)$ and $\sigma(\mathbf{r}_k)$ be its mean and standard deviation. When network conditions change,an ISP will probably experience an unexpected decrease or increase of its utility. As a consequence, the received reward would probably fall far away from the mean of the last N received rewards. By defining an interval around the mean of the received rewards, we can trigger exploration only when the last reward falls outside this interval. Equation (16) represents the 99th percentile interval—that is, given the mean μ of a normal distribution, the interval in which 99% of the observations fall:

$$CI_{99\%} = \mu(\mathbf{r}_k) \pm 2.576 \cdot \sigma(\mathbf{r}_k). \tag{16}$$

The action is randomly chosen when a reward outside this range is received. Moreover, since an ISP's utility could eventually drop to zero, exploration is triggered if $\mu(\mathbf{r}_k) = \sigma(\mathbf{r}_k) = 0$.

6. SIMULATION RESULTS

Extensive simulations were carried out to investigate the performance of the proposed pricing model and learning algorithms. The test topology consists of three symmetrically connected domains. As represented in Figure 2, every domain is composed of three border nodes. This test network is used to evaluate the influence of using an LA

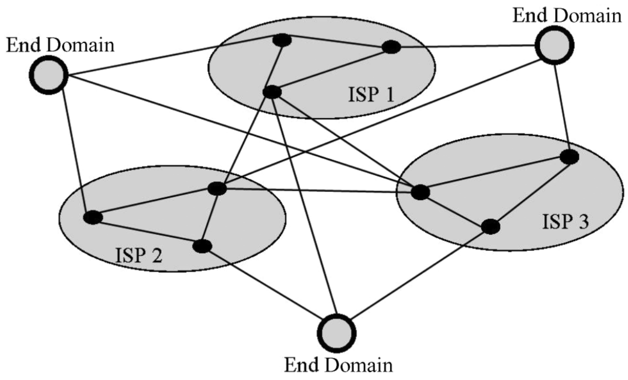


Fig. 2. Symmetric three-domain topology.

Table I. Learning Parameters

Parameter	Pursuit	CARLA
Action set	$[1, 9]$	$[1, 9]$
Number of actions	81	∞
Action quantization step	0.1	—
λ	0.02	0.1
σ	—	0.1
Size of reward vector $\mathbf{r}$	20	20
W	$100 * T_{ia}$	$100 * T_{ia}$

independently from the underlying intradomain physical topologies, and thus paths crossing only interdomain links are not allowed. Connection requests are generated only between end domains. Links have the capacity to accommodate a traffic flow of C_M unitary capacity. Interarrival times and duration of the flow demands are distributed according to a Poisson process, and the network load is expressed in Erlang. The sliding window parameter is chosen to get an average of 100 connection requests per domain. Since there are three domains and three generators, we set $W = 100 * T_{ia}$ for all experiments, where T_{ia} is the average interarrival time. Other parameters of the two learning algorithms are summarized in Table I. The choice of the quantization step of 0.1 for the Pursuit algorithm was taken after performing several tests. Using more actions will only increase the computation complexity, without improving the overall performance.

In the following sections,we consider three different case studies. In the first case, only one ISP learns the price while the network load and the prices of the others ISPs remain fixed. In this case, we want to test the performance of the proposed LA method in a stationary environment. In the second case, one ISP learns while the optimal price changes abruptly due to new load condition or new price settings. In this case, we want to test the two LA approaches in a nonstationary environment. Finally, we consider the case where two ISPs both learn their link prices using a combination of the two proposed LAs. We want to establish whether two learning ISPs converge to an equilibrium while still improving their utilities with regard to the nonlearning ISP.

In each of the cases under study, we evaluate the learning algorithms in two ways. We first present a qualitative analysis of the behavior of each algorithm by plotting the outcomes of multiple independent experiments. In this way, we demonstrate the algorithms' convergence to different attractor points. Additionally, we also evaluate the average performance of each algorithm. These averaged results are obtained by repeating simulations with different seeds until 5% accuracy is reached with 95% confidence—that is, the sample size for each experiment was taken so that the interval $[\bar{x} \pm 0.05 * \bar{x}]$ has a 95% probability of capturing the true mean. Here, $\bar{x}$ is the sample mean, and the 95% confidence interval was calculated assuming a normal distribution.

However, before evaluating the learning algorithms, we should answer the following question: how can we judge the solution found by the LAs if we do not know the optimal price?

The optimal price, in fact, depends on the load condition and the price settings of the involved ISPs. In the next section, we will see how to get around this problem by estimating the objective function of a learning ISP when the load and the prices of the other ISPs are fixed.

6.1. Performance Evaluation

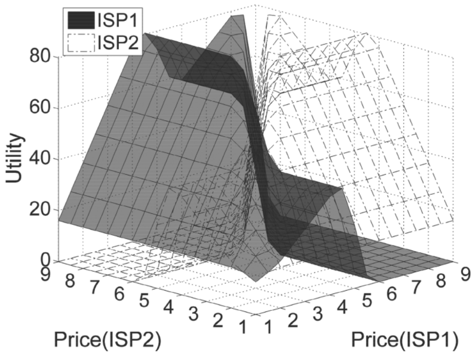
Considering that the objective function is unknown and depends on network parameters, as well as other ISPs pricing strategies, we face the difficulty of objectively evaluating the price found by the learning domains. If we knew the objective function that the automaton tries to maximize, we would be able to see if the solution actually yields the maximum of this function. To this extent, offline simulations can be carried to estimate the objective function. By fixing the network load and the price of one ISP, we can represent the utilities for different price values of the other two ISPs. Figure 3 represents utilities versus the price of ISP1 and ISP2 while ISP3's price is fixed to 5. Figure 3(a) and (b) show the utility of ISP1 and ISP2 for load values of 30 and 125 Erlang, respectively. Since the three domains are identical, the utilities of ISP1 and ISP2 are symmetric. For completeness, we also show the utility of ISP3, which has the price fixed to 5.

When one ISP learns the price and other ISPs use a fixed price, its objective function is a slice of the surfaces represented in Figure 3. Fixing the price of two ISPs (other than the network load) allows us to represent one of the possible realizations of the objective function and evaluate the solution of the LAs.

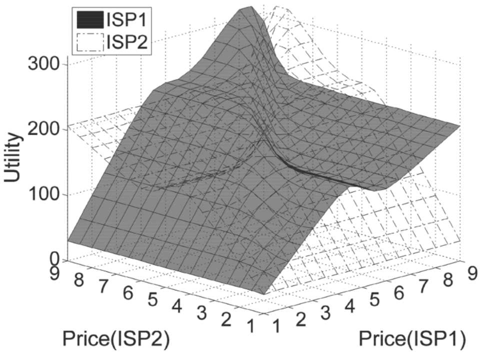
In Section 6.2, the scenario in which only one ISP learns the price is considered. We compare the learning solutions with the objective functions to see whether the ISP converges to a local or global maximum. In Section 6.3, the learning process is analyzed in a nonstationary environment, and we demonstrate the effectiveness of our selective exploration rule, whereas in Section 6.4, we consider the case of two ISPs learning simultaneously. It is important to note that the estimation of the objective functions is done only for comparison purposes and that ISPs do not perform any special offline computation to explicitly calculate them.

6.2. One ISP Learning in a Stationary Environment

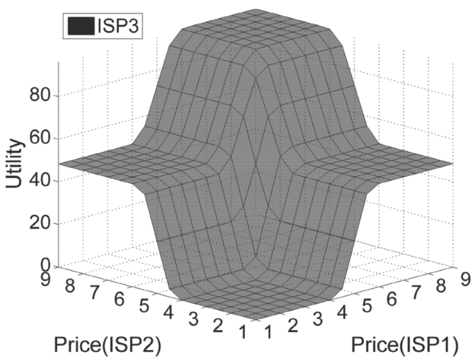
In the first experiment, we let ISP1 learn the price, assuming that other ISPs use a static strategy. To test the performance of the LAs, we compare the learning solutions with the objective function calculated for different load values and for different price settings of the other domains. Figure 4 shows results for four different load values and blocking probabilities (P_b). In all cases, ISP2 and ISP3 play a fixed price $Price(ISP2) = Price(ISP3) = 5$. In every chart, the x-axis is the action taken by ISP1, whereas the y-axis is the utility. The continuous line represents the objective function (evaluated with a 0.5 step) of the ISP1, whereas the dashed lines are the corresponding utility of the other two ISPs. The data points marked with $+$ and $\triangle$ represent solutions obtained when ISP1 learns the prices using CARLA and Pursuit, respectively. We want to demonstrate the convergence of the algorithms to different (possibly local) optima.



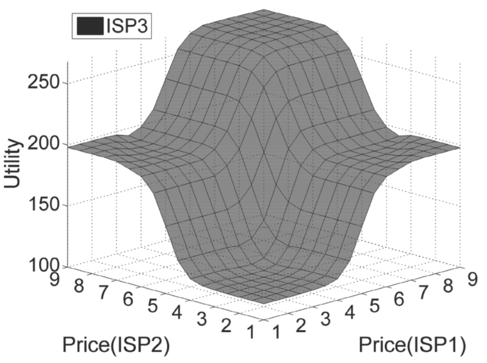
(a) ISP1, ISP2; Load=30Erl



(b) ISP1, ISP2; Load=125Erl



(c) ISP3; Load=30Erl

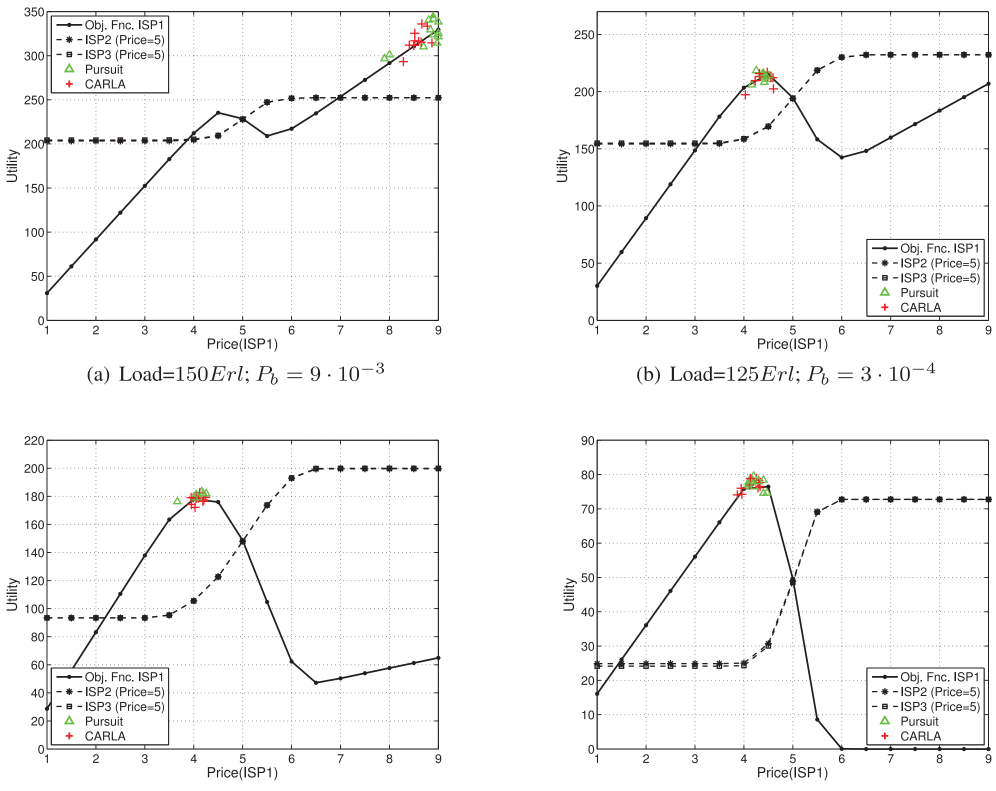


(d) ISP3; Load=125Erl

Fig. 3. Utilities when the price of ISP3 is set to 5.

Therefore, in this figure and those in the following sections, we plot the obtained utility after learning for each independent experiment rather than plot a single average over all experiments.

As expected, the optimal price depends on the network load as well as on the other ISPs' link prices. When the network load is high ($P_b \cong 1\%$), as represented in Figure 4(a), the optimal action is to set the maximum possible price, as the network is overloaded and domains always receive some traffic even if the price is high. As the load decreases, the objective also changes. According to the objective function shown in Figure 4(b), it is still possible to play high prices (i.e., $Price(ISP3) = 9$)), but in all cases, both CARLA and Pursuit converge to the maximum located in the region $(4, 4.5)$. Finally, in Figure 4, representing a low load scenario ($P_b < 0.0001\%$), the utility drops drastically to 0 when playing a price > 6. In this case, the maximum is located around the value 4, which, on average, has a higher utility than the price value 4.5. Charts in Figure 5 show results when ISP2 and ISP3 play different static prices for load values of 125 and 30 Erlang. In Figure 5(a) and (b), the objective function of ISP1 has been calculated by fixing the price of ISP2 to 5 and ISP3 to 3, whereas in



(a) Load=150Erl; $P_b = 9 \cdot 10^{-3}$

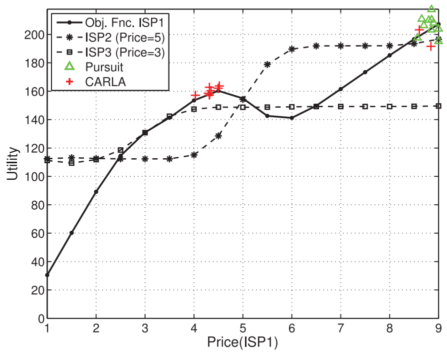
(b) Load=125Erl; $P_b = 3 \cdot 10^{-4}$

(c) Load=94Erl; $P_b \leq 10^{-5}$

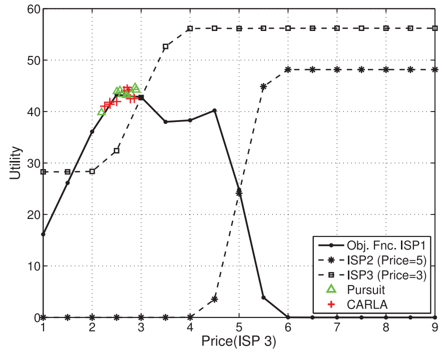
(d) Load=30Erl; $P_b \leq 10^{-6}$

Fig. 4. ISP Utilities versus Price (ISP1) when $Price(ISP2) = Price(ISP3) = 5$. Points show final utilities when ISP1 is learning. Each + or △ marks the outcome of a single independent experiment.

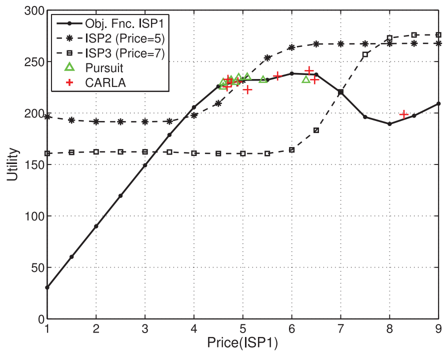
Figure 5(c) and (d), the prices of ISP2 and ISP3 have been fixed to 5 and 7, respectively. In this case, the objective functions present local maxima, whereas the global maximum, as shown in Figure 5(c) in the region [6, 6.5], does not always guarantee the highest utility among the three ISPs. In this scenario, LAs are able to find the optimal solution, but some differences emerge between CARLA and Pursuit. In the first chart (Figure 5(a)), Pursuit always converges to the global optimum, but CARLA gets trapped in the local maximum in most cases. In Figure 5(c), the maximum is located in the interval [4.5, 6.5]. In this case, the Pursuit algorithm converges to the local maximum located around the price 4.5, whereas the CARLA algorithm explores more the plateau and in some cases gets a slightly higher utility when converging to the global maximum located in the region [6, 6.5]. As we have seen, in this experiment, CARLA and Pursuit show almost the same behavior in most cases, and both algorithms are able to approach the optimal solution or at least a local optimum. However, the two automata operate in a different way, and thus it is worth analyzing the evolution of the learning process in time as represented in Figure 6. In this figure, we show typical runs for both algorithms to highlight the difference in behavior. From the evolution of the price learning process, the different action spaces used by the two algorithms are evident (continuous in CARLA, discrete in Pursuit). However, both algorithms have about the same learning speed. An important difference is that Pursuit tends to converge to a fixed price, whereas CARLA always oscillates around the



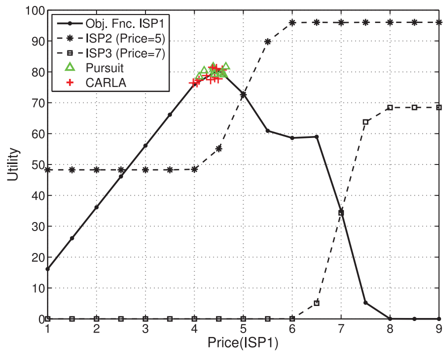
(a) Load= $125Erl$, $Price(ISP3) = 3$



(b) Load= $30Erl$, $Price(ISP3) = 3$

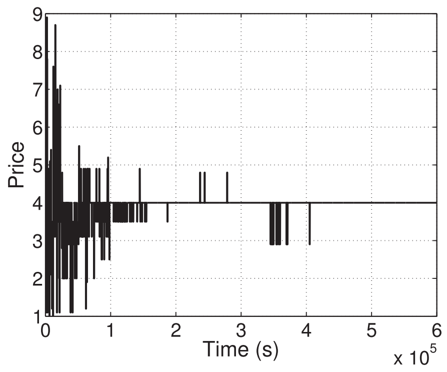


(c) Load= $125Erl$, $Price(ISP3) = 7$

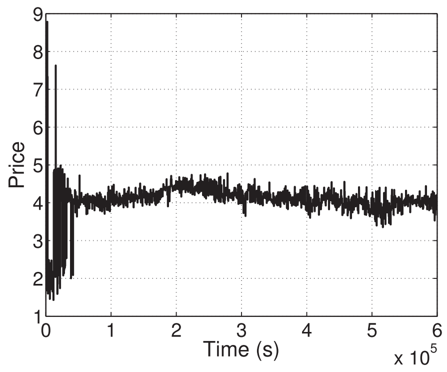


(d) Load= $30Erl$, $Price(ISP3) = 7$

Fig. 5. ISP Utilities versus Price(ISP1) when $Price(ISP2) = 5$. Points show final utilities when ISP1 is learning. Each + or △ marks the outcome of a single independent experiment.



(a) Price, Pursuit



(b) Price, CARLA

Fig. 6. Behavior of both algorithms during learning. We show the evolution of Price(ISP1) versus Time when Load = $30Erl$ and $Price(ISP2) = Price(ISP3) = 5$. Typical runs for Pursuit (a) and CARLA (b).

optimal value. This can be explained again with the different action space used by the two algorithms.

6.3. One ISP Learning in a Nonstationary Environment

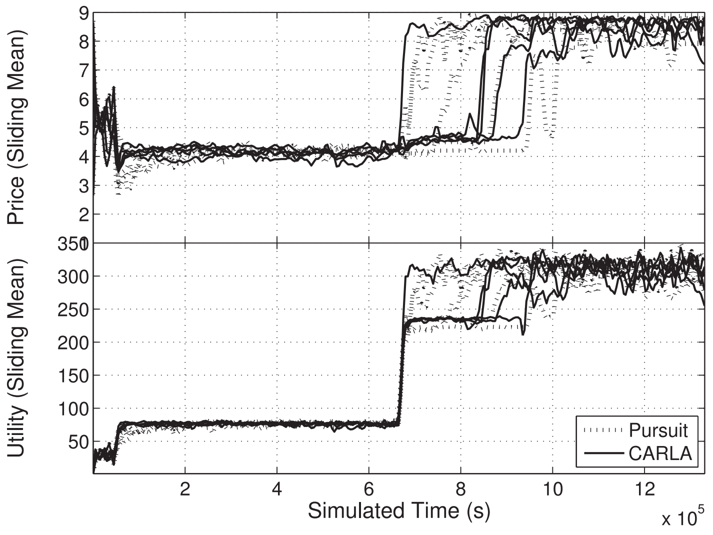
Up to this point, we have only considered scenarios where the optimal price does not change over time—that is, network load and prices of nonlearning ISPs are fixed over time. In this case, the automaton, starting from a random strategy, converges to the optimal price. However, load conditions and link prices are not constant. The network load, for example, may change on a night/day scenario or in conjunction with some popular event (e.g., elections in a country). Moreover, another ISP can always change its link prices without notifying opponent ISPs. Since the automata should avoid reinitialization as much as possible, it is important that the LAs are able to detect a change in the objective function and learn the new price without reinitializing the full learning process.

In this section, we analyze the automata behavior when the objective functions, and thus the optimal solution, change at a certain time instant. More precisely, we consider three cases: when the optimal price changes due to a network load shift from low to high load, when it changes due to a load shift from high to low load, and when it changes because one ISP sets a different link price.

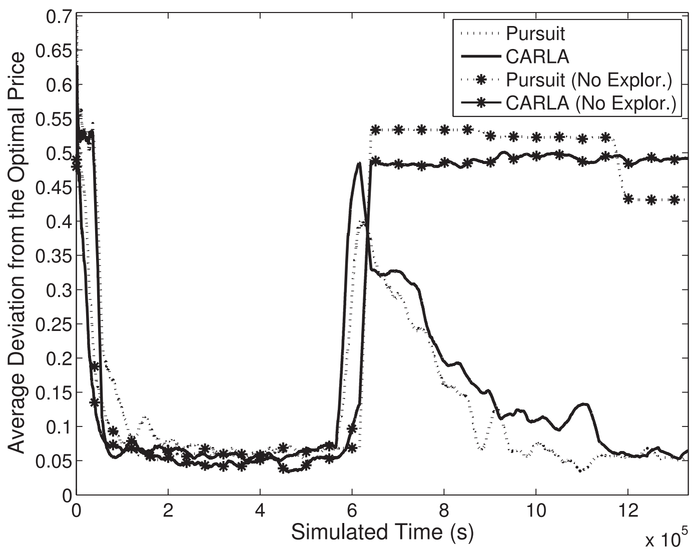
Figures 7, 8, and 9 show the price learning process and the corresponding utility for the three different scenarios when CARLA and Pursuit algorithms are used. Charts 7(b), 8(b) and 9(b) show the deviation from the optimal values normalized by the optimal price, $|x(k) - x_{opt}|/x_{opt}$, averaged over all different realizations. This value represents the error relative to the optimal value. In all cases, the objective function changes halfway through the simulation, and the LA has to learn the new price without reinitializing its action probabilities. To assess the impact of our selective exploration rule, we compare the results with and without the exploration rule.

First of all, we can observe the importance of using our selective exploration rule when LAs operate in a nonstationary environment. As the network condition changes, the automaton implementing selective exploration starts to receive rewards outside the interval of Equation (16) and thus triggers exploration sessions, which allows it to escape from the old solution. Only in the case shown in Figure 9(b) is CARLA able to track the changes without any exploration, whereas in all other cases, the LAs are not able to operate in a nonstationary environment without using exploration.

Figures 7(a) and 8(a) show the learning process when the network load increases and decreases in the middle of the simulation time, whereas Figures 7(b) and 8(b) show the corresponding average deviation from the optimal value. In both cases, the two automata are able to track the change and converge to a new solution. Some differences emerge between the two algorithms, however. As shown in charts 7(b) and 8(b), CARLA is able to converge faster to the new solution, whereas Pursuit has more difficulties. In the first scenario, the load is changed from 30 to 150 Erlang while the two nonlearning ISPs play a price of 5. The corresponding objective function changes are shown in Figure 4(a) and (b) of Section 6.2. From the few runs represented in the chart on the left side, it is possible to see how both CARLA and Pursuit first converge to the local optimum corresponding to the value 4.5 and after some time converge close to the new optimal price 9. From the chart on the right, we can see that both algorithms have the same convergence speed, and both share the risk of staying trapped in a locally optimal price setting. On the other hand, when considering the other scenario when the load changes from 150 to 30 Erlang, a substantial difference exists between the two algorithms. CARLA, in fact, shows a faster reaction than Pursuit, and the error at the end of the simulation is, on average, less than 5% of the optimal price. Pursuit

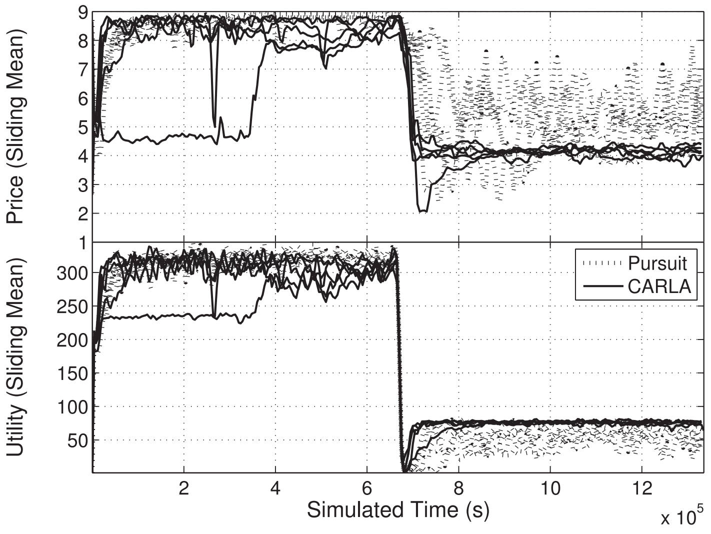


(a) Load=30 $\Rightarrow 150Erl$

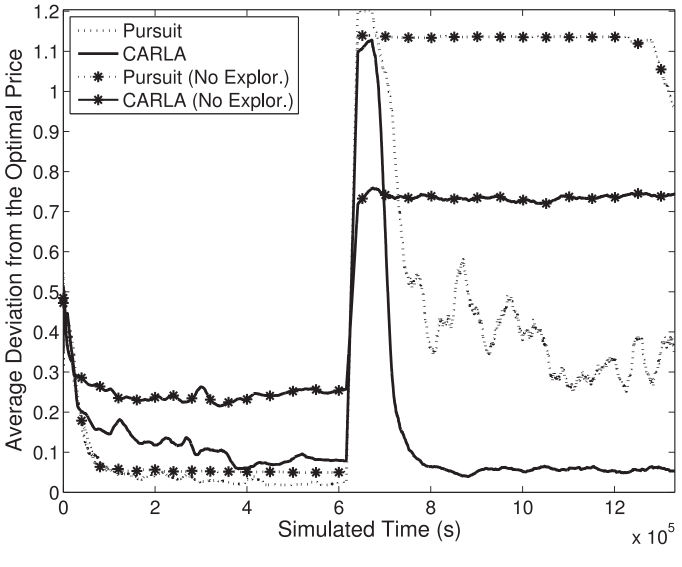


(b) Load=30 $\Rightarrow 150Erl$

Fig. 7. Chart (a) shows the price learning process (Price(ISP1) and Utility(ISP1) vs. Time) when load changes from low (30 Erlang) to high (125 Erlang). We show five independent runs for each algorithm. Chart (b) shows the deviation from the optimal value averaged over multiple runs. To calculate these averages, simulations with different seeds were run until 5% accuracy was obtained with 95% confidence. The learning process without use of the exploration rule is also shown in the bottom chart.

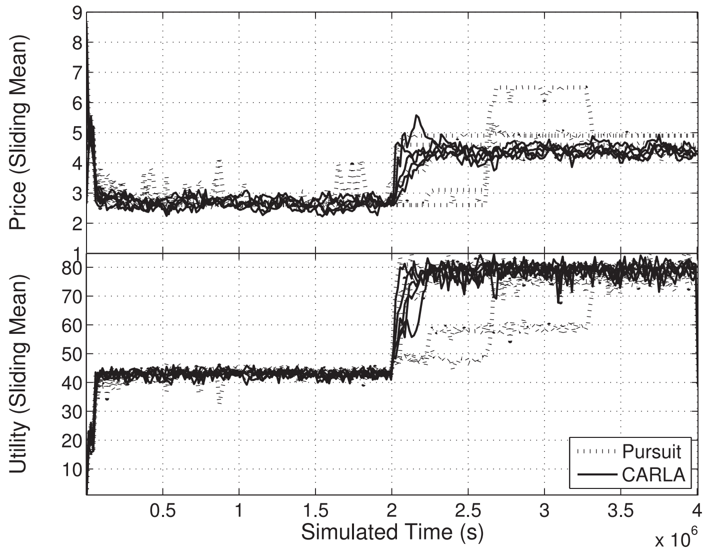


(a) Load=150 $\Rightarrow 30Erl$

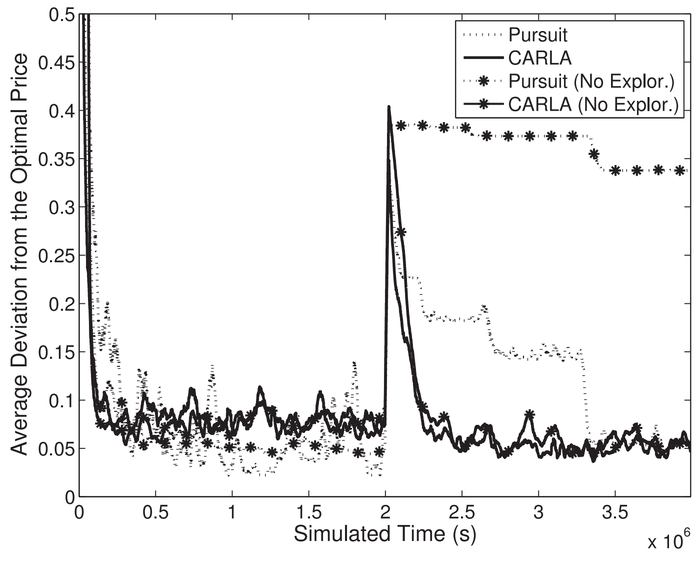


(b) Load=150 $\Rightarrow 30Erl$

Fig. 8. Chart (a) shows the price learning process (Price(ISP1) and Utility(ISP1) vs. Time) when load changes from high (125 Erlang) to low (30 Erlang). We show five independent runs for each algorithm. Chart (b) shows the deviation from the optimal value averaged over multiple runs. To calculate these averages, simulations with different seeds were run until 5% accuracy was obtained with 95% confidence. The learning process without use of the exploration rule is also shown in the bottom chart.



(a) $Price(ISP3) = 3 \Rightarrow 7$



(b) $Price(ISP3) = 3 \Rightarrow 7$

Fig. 9. Chart (a) shows the price learning process (Price(ISP1) and Utility(ISP1) vs. Time) when ISP3 changes its static price from 3 to 7. We show five independent runs for each algorithm. Chart (b) shows the deviation from the optimal value averaged over multiple runs. To calculate these averages, simulations with different seeds were run until 5% accuracy was obtained with 95% confidence. The learning process without use of the exploration rule is also shown in the bottom chart.

instead encounters more difficulties and oscillates around the optimal value showing an average error of 30% of the optimal value.

In the third scenario, the network load is fixed while one of the nonlearning ISPs changes its price from 3 to 7 and the objective function changes as shown in Figure 5(b) and (d). As well in this case, CARLA converges much faster to the new optimal price, whereas Pursuit takes significantly longer to learn the new price.

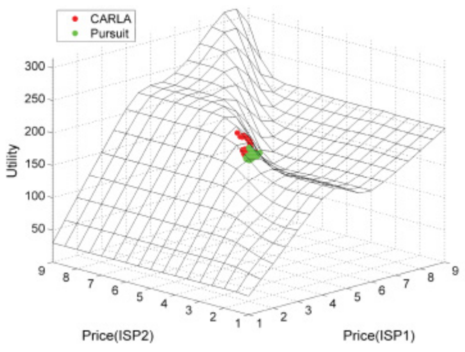
From this experiment, we can conclude two main results. First, both algorithms need to implement a selective exploration rule to be able to learn in a nonstationary environment. Second, CARLA performs better in a nonstationary environment since, on average, it learns a new solution faster than Pursuit.

6.4. Two ISPs Learning

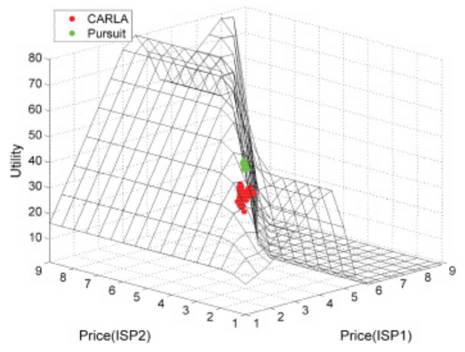
In this experiment, we let ISP1 and ISP2 learn the link prices while the price of ISP3 is fixed to 5. In this scenario, the two ISPs cannot maximize their utilities independently, but we have to focus on the stability of the solution. Considering that ISPs are competitors, fairness is not required; however, it still represents an interesting solution because when all ISPs believe that they have received a fair utility, they are most likely to behave honestly. Since we consider three equal ISPs, fairness is fulfilled in the line where the two ISPs get equal utilities. In particular, the maximum of this line corresponds to the point where $price(ISP1) = price(ISP2) = 4$ in Figure 3(a) and $price(ISP1) = price(ISP2) = 9$ in Figure 3(b). These points are Pareto optimal (PO) solutions, as there are no other solutions ensuring a higher utility to one ISP without making the other's utility worse off. However, not all points along this line are stable. In a continuous action setting, two players are in a stable equilibrium if they do not have any incentive to unilaterally change their prices in the vicinity of the equilibrium point. A Nash equilibrium can satisfy this stability criterion but may be different from the PO solution. In our case, the PO outcomes are not stable, as one ISP can get a higher utility (to the detriment of the other) by slightly changing link prices. In Figure 10, the learning solutions found by the two learning ISPs are shown. This time, the two automata oscillate around an equilibrium region; the two spots represent this region and are achieved by sampling the utility and prices reached on several runs.

Figure 10(a) and (b) represent learning solutions when both ISP1 and ISP2 use the same LA. Only ISP1 solutions are represented, as ISP2 solutions are symmetrical and similar. For high load ($125Erl$), the two ISPs converge to an equilibrium strategy around the regions delimited by the prices $price(ISP1) = price(ISP2) = 4$. In particular, when using the Pursuit algorithm, the learning dynamic shows a convergence to the point $(4, 4)$, whereas CARLA explores more the neighborhood of this region. This difference can be explained by considering that Pursuit uses an update rule, which only increases the probability of the action with the highest average reward and equally decreases the probabilities of the other actions such that the sum of probabilities is equal to 1. CARLA, instead, uses a Gaussian function to update the PDF of its continuous action set. Depending on the spreading rate parameter, the Gaussian function encourages the exploration of the neighborhood of the successfully applied action. A more evident difference is shown in the low load scenario where CARLA and Pursuit converge around two different equilibrium points. CARLA converges to the point $(2.8, 2.8)$, whereas Pursuit finds an equilibrium around the point $(3.4, 3.4)$, which corresponds to a higher utility and is also closer to the PO solution.

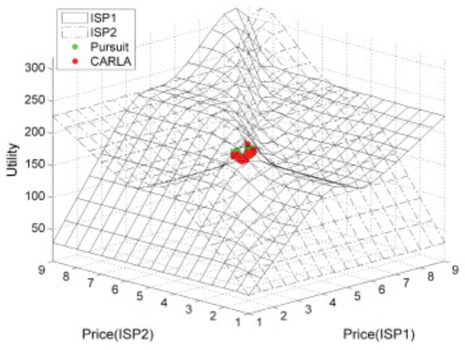
From these experiments, it is evident how the different convergence regions represent equilibria corresponding to local optima for both learners. In this context, any exploration will produce an oscillation around the equilibrium, but the ISPs do not have an incentive to move away from this region. Even if the final outcomes are not PO, they still represent safe and stable solutions and the two learning ISPs are still



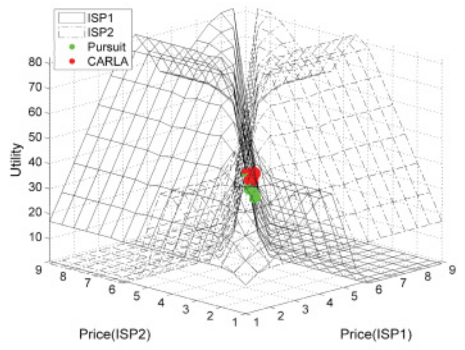
(a) ISP1, Load=125Erl



(b) ISP1, Load=30Erl



(c) Load=125Erl



(d) Load=30Erl

Fig. 10. Utilities and Learning Evolution when $Price(ISP3) = 5$. (a) and (b) show ISP1 Utility and Learning Evolution when both ISP1 and ISP2 use CARLA or both use Pursuit. (c) and (d) show ISP1 and ISP2 Utilities and Learning Evolution when ISP1 uses Pursuit and ISP2 uses CARLA. Each dot shows the outcome of a single independent experiment.

able to improve their utilities in regard to the nonlearning ISP. In Figure 11, we show the evolution of the price points during learning for two ISPs using CARLA when ISP 3 plays a fixed price ($price(ISP3) = 5$). The experiment is repeated multiple times, each time starting from a different initial price setting for the learning ISPs. It can be observed that the equilibrium point is a global attractor for the learners, and all learning trajectories move toward this point.

Since there is a significant difference between the two learning algorithms, we also analyze the case where one ISP uses CARLA and the other uses Pursuit. Figure 10(c) and (d) show the learning solution when ISP1 uses Pursuit and ISP2 uses CARLA. While in the high load scenario (Load = 125), the two domains converge to the same solution and get equal utilities; in the low load scenario, the CARLA algorithm gains a higher utility compared to Pursuit. This can be explained considering that CARLA can adapt its price faster than Pursuit. In this experiment, in fact, ISP2 (using CARLA) is able to get a higher utility by decreasing its price with regard to ISP1. The convergence region is centered around the point $(3.2, 2, 8)$. As shown in Section 6.3, CARLA has the ability to adapt its price faster than Pursuit. The result is that when the two automata compete, CARLA can improve its utility without Pursuit being able to adapt its price. To summarize the results presented earlier, Figure 12 shows the average utility of the

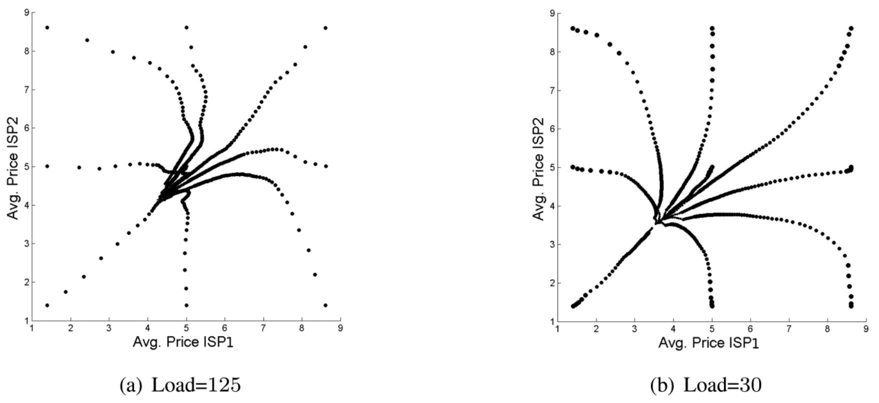


Fig. 11. Evolution of price setting during learning for two learning ISPs. Each trajectory represents an experiment starting from different initial price settings.

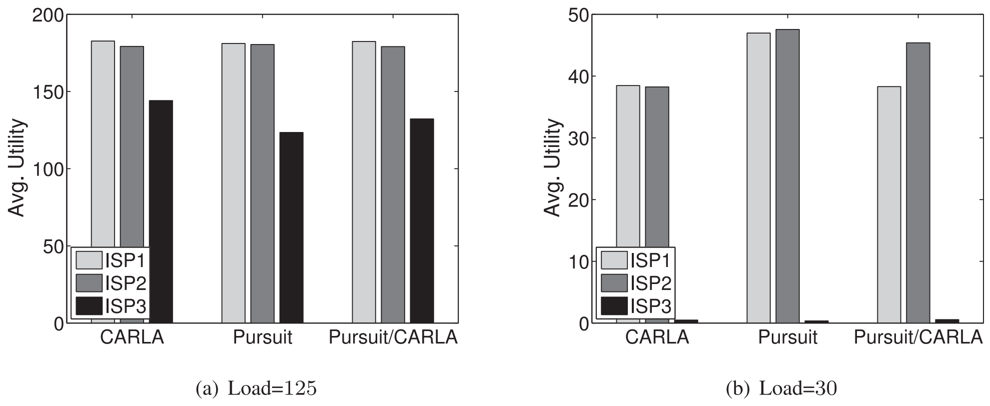


Fig. 12. Average Utilities versus Learning Automata when ISP1 and ISP2 learn. To calculate these averages, simulations with different seeds were run until 5% accuracy was obtained with 95% confidence.

three ISPs for the different cases when ISP1 and ISP2 both use CARLA or Pursuit and when ISP1 uses Pursuit and ISP2 uses CARLA. In the high load scenario, there are no evident differences between the three cases, except for the nonlearning ISP (ISP3), which improves its utility when CARLA is employed in both domains. This can be explained considering that CARLA explores more the neighborhood of the optimal price, allowing ISP3 to gain more utility. In the low load scenario, we can see how ISP1 and ISP2 improve their utilities when both employ Pursuit. On the other hand, when the two ISPs use different algorithms, the ISP using CARLA gets a higher utility thanks to its ability to adapt its price faster than the Pursuit algorithm.

6.5. Summary of Results

In this section, we give an overview of the conclusions of all simulation results. We have considered three distinct scenarios:

—*In the stationary setting*, where a single ISP attempts to optimize its link prices while other ISP prices and network conditions are fixed, we see that both algorithms give similar performance. When a single optimal price point exists, both algorithms reliably converge to this optimum. When multiple local optima exist, however, we see that the Pursuit automata are always able to converge to this point, whereas CARLA may get stuck in a local optimum. It should be noted, however, that since the Pursuit algorithm uses a discrete action set, its performance very much depends on using a suitable discretization, which may require domain knowledge. Furthermore, extremely fine discretizations will result in very large action sets, which can slow learning considerably.

—*In the second series of simulations*, we study the impact of sudden changes in environment dynamics (e.g., a sudden increase in network load). The basic learning algorithms are not able to deal with these sudden switches and continue playing a previously learned suboptimal strategy. We address this problem by introducing a selective exploration rule. This extension allows the learners to deal with nonstationary problems where sudden shifts in the optimal strategy can occur. Learners trigger a new round of exploratory actions whenever an unexpected change in utility is observed. Using this exploration rule, both algorithms are able to escape the suboptimal solution and identify the new optimum. However, we do see that the CARLA algorithm is able to find the new optimal solution considerably faster than the Pursuit algorithm.

—*In the final set of simulations*, we consider a fully dynamic, game-type scenario in which multiple ISPs attempt to learn optimal link prices at the same time. We observe that both algorithms converge to an equilibrium point between the ISP prices, irrespective of the starting conditions of the game. In scenarios where one ISP uses the CARLA algorithm while the other uses Pursuit, however, we see that the CARLA learners gain a higher utility compared to Pursuit, as they are able to adapt faster in the game scenario.

7. DISCUSSION

In this paper, we proposed the use of LAs to learn optimal link prices in interdomain routing settings with multiple ISPs. The learning problem can then be treated as an (unknown) game between the different automata in the system. Each automaton in the system individually updates its strategy based only on a performance measure that depends on the current network state. The approach was validated in a number of network simulations corresponding to different learning scenarios.

We believe that this automata game approach has great potential for the control of complex systems that are hard to solve analytically. These possible control applications are not limited to the link pricing case study presented in this article. Networks consisting of discrete LAs have been well studied, both for general control problems [Witten 1977; Wheeler Jr. and Narendra 1986; Vrancx et al. 2008] and in a variety of other settings (e.g., see Thathachar and Sastry [2004] for an overview). The continuous CARLA algorithm has been successfully applied in several control applications [Howell et al. 1997; Howell and Best 2000]. More recently, we also investigated the use of multiple CARLA, both in complex mechatronic control settings [Rodriguez et al. 2013] and for the management of water reservoir networks [Rodriguez et al. In Press].

We argue that, in general, LAs are very interesting building blocks for learning control in distributed control applications. LAs can be viewed as policy iterators that update their action probabilities based on private information only. This is a very attractive property in applications where communication is expensive. LAs are updated strictly on the basis of a local cost function and not on the basis of any knowledge regarding the system dynamics or other automata in the system—that is, they do not need to explicitly model the system or opponent strategies. As such, LA agents provide a very simple yet efficient control method for complex systems.

Moreover, LAs can also be treated analytically. In this article, we demonstrate how the different LA models converge to local attractors in the game resulting from their interactions. General convergence proofs exist for a variety of settings ranging from a single automaton model acting in a simple stationary random environment to distributed automata interacting in a complex environment [Thathachar and Sastry 2004]. In the case of the CARLA algorithm used in this work, we can expect the algorithm to converge to the unique optimum when the system exhibits linear dynamics and a linear cost function is considered. If the system dynamics or its cost function are not linear, however, multiple optima may exist. Using the standard CARLA algorithm for controlling subsystems in a nonlinear interacting system, while ignoring the presence of the other controllers, the system will successfully converge, but convergence to the global optimum cannot be guaranteed Rodríguez et al. [2011].

To deal with the complexities arising in games with multiple attractors, we recently proposed an extension to the basic CARLA learning scheme [Rodriguez et al. In Press]. More precisely, we introduced a novel method for estimating the basin of attraction around a locally superior strategy in a common interest game. This allows learners to efficiently explore the action space by avoiding the basin of attraction of previously discovered attractors. We show that sets of CARLA using this technique converge to a Nash equilibrium or even the PO Nash equilibrium. In future work, we hope to further demonstrate how this approach can achieve excellent performance in a variety of different distributed control problems.

8. CONCLUSIONS

In this article, we presented an interdomain routing framework in which ISPs compete with each other to sell resources in an interdomain transit market. Two different LA algorithms, used by ISP operators to learn the optimal price, were analyzed and compared. A finite action LA using the Pursuit algorithm was compared with a continuous action learner named CARLA. To improve the performance of both LA methods, we introduced a selective exploration rule that triggers exploration of the action space when an extraordinary low or high reward is received.

Simulation results show that an ISP can improve its utility by using an LA to learn the best pricing strategy. Experiments with different traffic loads and price combinations show the dependency of the objective function on the network state. When one ISP learns the price, it is able to find the optimal solution in all settings. When two ISPs employ the same LA, they can converge to a stable strategy while still improving their utilities.

The two algorithms exhibit a different behavior, however. When one ISP uses CARLA and another uses Pursuit, CARLA is seen to obtain a higher utility than Pursuit. On the other hand, LAs using the Pursuit algorithm converge to the optimal solution with higher precision than those using CARLA, but this will also depend on the action set discretization used with the Pursuit algorithm. The continuous action space and the Gaussian-shaped impulse, used to update the action PDF, reduce the convergence precision and cause small oscillations around the optimal solution. If we compare the two algorithms in a dynamic scenario, where the optimal solution changes at a certain time instant, CARLA definitely shows a faster response than Pursuit. Thanks to our exploration and reward transformation rules, both algorithms are able to move toward new optimal solutions. Pursuit, however, is not able to escape from the old solution as fast as CARLA. The same factors that prevent CARLA from converging with high precision to the optimal price in the stationary scenario are shown to be a strength when the LAs are used in a dynamic environment. The ability of CARLA to adapt the solution to the new network condition makes it the most suitable choice for a network operator. We conclude that a CALA is more suitable for use in these dynamic environments.

REFERENCES

Enda Barrett, Jim Duggan, and Enda Howley. 2014. A parallel framework for Bayesian reinforcement learning. *Connection Science* 26, 1, 7–23.

Dominique Barth and Loubna Echabbi. 2008. Optimal transit price negotiation: The distributed learning perspective. *Journal of Universal Computer Science* 14, 5, 745–765.

Lucian Buşoniu, Robert Babuška, and Bart De Schutter. 2010. Multi-agent reinforcement learning: An overview. In *Innovations in Multi-Agent Systems and Applications*. Studies in Computational Intelligence, Vol. 281. Springer, 183–221.

Ross Cressman. 2005. Stability of the replicator equation with continuous strategy space. *Mathematical Social Sciences* 50, 2, 127–147.

Pasquale Gurzi, Kris Steenhaut, Ann Nowé, and Peter Vrancx. 2011. Learning a pricing strategy in multi-domain DWDM networks. In *Proceedings of the 18th IEEE Workshop on Local and Metropolitan Area Networks*. IEEE, Los Alamitos, CA, 1–6.

Mark N. Howell and Matt C. Best. 2000. On-line PID tuning for engine idle-speed control using continuous action reinforcement learning automata. *Control Engineering Practice* 8, 2, 147–154.

Mark N. Howell, Geoff P. Frost, Timothy J. Gordon, and Qing H. Wu. 1997. Continuous action reinforcement learning applied to vehicle suspension control. *Mechatronics* 7, 3, 263–276.

Hansan T. Karaoglu and Murat Yuksel. 2010. Value flows: Inter-domain routing over contract links. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM'10)*. 342–347.

Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. 2010. Internet inter-domain traffic. *ACM SIGCOMM Computer Communication Review* 40, 4, 75–86.

Krisztina Loja, Jeanos Szigeti, and Tibor Cinkler. 2005. Inter-domain routing in multiprovider optical networks: Game theory and simulations. In *Proceedings of Next Generation Internet Networks*. IEEE, Los Alamitos, CA, 157–164. DOI:http://dx.doi.org/10.1109/NGI.2005.1431661

Jorg Oechssler and Frank Riedel. 2002. On the dynamic foundation of evolutionary stability in continuous models. *Journal of Economic Theory* 107, 2, 223–252.

Liviu Panait and Sean Luke. 2005. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems* 11, 3, 387–434.

Abdel Rodríguez, Ricardo Grau, and Ann Nowé. 2011. Continuous action reinforcement learning automata—performance and convergence. In *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence*. 473–478.

Abdel Rodriguez, Peter Vrancx, Ricardo Grau, and Ann Nowé. 2012a. Learning approach to coordinate exploration with limited communication in continuous action games. In *Proceedings of the AAMAS 2012 Workshop on Adaptive and Learning Agents*. 17–23.

Abdel Rodriguez, Peter Vrancx, Ricardo Grau, and Ann Nowé. 2012b. An RL approach to common-interest continuous action games. In *Proceedings of the 11th International Conference on Adaptive Agents and Multi-Agent Systems (AAMAS)*. 1401–1402.

Abdel Rodriguez, Peter Vrancx, and Ann Nowé. In Press. A reinforcement learning approach to coordinate exploration with limited communication in continuous action games. *Knowledge Engineering Review* 31, 2. Available at http://ai.vub.ac.be/ALA2012/KER.html.

Abdel Rodriguez, Peter Vrancx, Ann Nowe, and Erik Hostens. 2013. Model-free learning of wire winding control. In *Proceedings of the 9th Asian Control Conference (ASCC)*. IEEE, Los Alamitos, CA, 1–6.

Gireesh Shrimali, Aditya Akella, and Almir Mutapcic. 2010. Cooperative inter-domain traffic engineering using Nash bargaining and decomposition. *IEEE/ACM Transactions on Networking* 18, 2, 1063–6692.

Mandayam A. L. Thathachar and P. Shanthi Sastry. 2004. *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Kluwer Academic.

Mandayam A. L. Thathachar and P. Shanthi Sastry. 1986. Estimator algorithms for learning automata. In *Proceedings of the Platinum Jubilee Conference on System Signal Processing.*

Omkar Tilak and Snehasis Mukhopadhyay. 2011. Partially decentralized reinforcement learning in finite, multi-agent Markov decision process. *AI Communications* 24, 4, 293–309.

Vytautas Valancius and Cristian Lumezanu. 2011. How many tiers? Pricing in the Internet transit market. In *Proceedings of the ACM SIGCOMM 2011 Conference*. 194–205.

Matthijs Veelen and Peter Spreij. 2008. Evolution in games with a continuous action space. *Economic Theory* 39, 3, 355–376.

Peter Vrancx, Katja Verbeeck, and Ann Nowé. 2008. Decentralized learning in Markov games. *IEEE Transactions on Systems, Man, and Cybernetics, Part B* 38, 4, 976–981.

Richard Wheeler Jr. and Kumpati S. Narendra. 1986. Decentralized learning in finite Markov chains. *IEEE Transactions on Automatic Control* 31, 6, 519–526.

Ian H. Witten. 1977. An adaptive optimal controller for discrete-time Markov environments. *Information and Control* 34, 4, 286–295.

Received March 2013; revised November 2014; accepted January 2015