Kastius, Alexander; Schlosser, Rainer

Dynamic pricing under competition using reinforcement learning

Journal of Revenue and Pricing Management

Provided in Cooperation with:
Springer Nature

Suggested Citation: Kastius, Alexander; Schlosser, Rainer (2021) : Dynamic pricing under competition using reinforcement learning, Journal of Revenue and Pricing Management, ISSN 1477-657X, Palgrave Macmillan UK, London, Vol. 21, Iss. 1, pp. 50-63, https://doi.org/10.1057/s41272-021-00285-3

Journal of Revenue and Pricing Management (2022) 21:50–63
https://doi.org/10.1057/s41272-021-00285-3

RESEARCH ARTICLE

Dynamic pricing under competition using reinforcement learning

Alexander Kastius[1] · **Rainer Schlosser**[1]

Received: 7 October 2020 / Accepted: 18 January 2021 / Published online: 27 February 2021

Abstract
Dynamic pricing is considered a possibility to gain an advantage over competitors in modern online markets. The past advancements in Reinforcement Learning (RL) provided more capable algorithms that can be used to solve pricing problems. In this paper, we study the performance of Deep Q-Networks (DQN) and Soft Actor Critic (SAC) in different market models. We consider tractable duopoly settings, where optimal solutions derived by dynamic programming techniques can be used for verification, as well as oligopoly settings, which are usually intractable due to the curse of dimensionality. We find that both algorithms provide reasonable results, while SAC performs better than DQN. Moreover, we show that under certain conditions, RL algorithms can be forced into collusion by their competitors without direct communication.

Keywords Dynamic pricing · Competition · Reinforcement learning · E-commerce · Price collusion

Introduction

In modern-day online trading on large platforms using the correct price is crucial. If your goods' prices are way off the competition, customers might go for cheaper competitors or ones that offer a better service or a similar product. Many traders nowadays can make use of dynamic pricing algorithms, that automatically update their price according to the competitors' current offers. Those price updates might occur at a high frequency. While it is possible to use simple pricing strategies and tune their parameters manually, this raises the question of whether it is possible to provide an automated solution for such problems.

We analyze two examples of such markets focusing on durable and replenishable goods: (i) duopolies, that require the agent to compete with a single competitor and (ii) oligopolies with multiple active competitors. Duopoly markets offer the advantage, that optimal solutions can still be computed via dynamic programming (DP), cf., e.g., Schlosser and Richly (2019), which provides an opportunity to compare and verify the results of reinforcement learning (RL).

RL offers algorithms, that are focused around solving games by maximizing a reward stream offered by the game. They are problem independent and require only hyperparameter tuning to be set up. They have been used for various problems in the past, for example, video games and robotics. A few examples of them being applied to other pricing problems are available as well, see Kephart and Tesauro (2000), Kim et al. (2016), and Rana and Oliveira (2014). Deep Q-Networks (DQN, Mnih et al. 2015) is a commonly used example of classical RL algorithms. It estimates the expected reward of pursuing a certain action and searches for the action with the highest expected reward. Other approaches, like Soft Actor Critic (SAC), see Haarnoja et al. (2018), rely on directly tuning a parametrized strategy and use value estimations only to guide updates of the policy.

In this paper, we analyze how far RL algorithms can be used to overcome the limitations of dynamic programming approaches to solve dynamic pricing problems in competitive settings. Our contributions are:

- We compute self-adaptive pricing strategies using DQN and SAC algorithms.
- We compare their performance compared to optimal strategies in tractable duopoly settings derived by dynamic programming techniques.
- We study RL strategies regarding their tendencies in a duopoly to form a cartel.

✉ Rainer Schlosser
rainer.schlosser@hpi.de

Alexander Kastius
alexander.kastius@hpi.de

[1] Hasso Plattner Institute, University of Potsdam, Potsdam, Germany

- We show that RL strategies can be successfully applied in oligopoly scenarios.

The "Related work" section discusses related work. The "RL algorithms" section gives an overview of RL algorithms. In the "Performance in duopoly environments" section, we study RL strategies in different duopoly settings against deterministic and randomized strategies. In the "Price collusion in a duopoly" section, we investigate the formation of price collusion. The "Competition between different self-adaptive learning strategies" section shows experiments, where two RL systems directly compete against each other. The "Application in oligopoly competition" section considers oligopoly settings. The "Conclusion" section concludes the paper.

Related work

The domain of dynamic pricing is large. While RL is well known in the community, publications around it are scattered and broad, as observed by den Boer (2015). Many examples are available where RL has been applied to different market models.

Kutschinski et al. (2003) present a multi-agent market model, in which the seller has to determine both the offer price and the number of goods produced in a setting with another competitor that dynamically makes the same decision. They used Q-learning to implement a pricing bot in that scenario. They observe that Q-learning does find near-optimal policies, though being slower than specialized alternatives. They use Q-learning only with discount factors being set to zero, making it unsuitable for problems where long-term effects do matter.

Kephart and Tesauro (2000) use Q-learning in multi-agent scenarios with a demand model in which the reward is a function that mostly relies on the price rank. They analyze a scenario with two agents using tabular Q-learning to learn pricing policies in a duopoly setup. In such a setup, there are several equilibria, in which very similar pricing policies are used by both agents and stay unaltered. Q-learning does converge toward such equilibria. At some points, pseudo-solutions are found, in which no exact equilibrium is achieved. Instead, both agents deviate around the equilibrium slightly. Kephart et al. show that in setups with two agents and a very simplistic demand model, such equilibria are found reliably.

Könönen (2006) presents a similar evaluation of Q-learning that uses function approximation instead of the tabular approaches and introduces policy gradient methods to the same challenge. His methodology is loosely based on Kephart and Tesauro (2000). He concludes that the convergence toward equilibria can be achieved consistently, with adjustments of the original approach. His method still requires an algorithm where a single parameter is stored for every game state and price choice combination, which becomes intractable for larger or continuous state-spaces, an issue that is overcome with the methods used in this work.

While the previous publications used early Q-learning methods to solve multi-agent problems, as we do it, many other evaluations rely on applying similar methods to very specialized single-agent environments. Gosavi et al. (2002) apply reinforcement learning to revenue management in airline pricing systems. Their simulation includes real-world problems, for example, overbooking and different classes of customers. They use temporal difference learning and neural networks in conjunction to provide a well-performing pricing model for airlines that outperforms industry-standard algorithms at that point in time. Gosavi developed a new RL algorithm aiming at the optimization of the average achieved reward per episode, which was successfully applied to yield management for the airline industry as well, cf. Gosavi (2004).

Vengerov (2008) formulates the problem of selling computation time in a cloud service as a partially observable Markov decision process, where the information about possible competitors is not fully accessible to the agent. He shows that a policy gradient algorithm can successfully handle partially observable decision processes. It is observed that algorithms like this tend to converge toward equilibria in multi-agent environments. His approach relies on non-deterministic policy gradient algorithms, a concept used by us for further analysis.

Rana and Oliveira (2014) suggest using Q-learning to solve pricing problems with multiple products with interdependent demand. Q-learning seems to be a promising approach in those cases where the user has no explicit knowledge about the exact relationship between multiple products and must take such factors into account. It is suggested to use more sophisticated approaches that rely on function approximation, such as deep Q-learning, to tackle such situations in the future.

While reinforcement learning provides a general utility tool for decision problems, specialized pricing models are still developed to utilize knowledge about customer and market behavior. Ye et al. developed such a model for the pricing of vacation homes at AirBnB that relies on a complex regression model that predicts sales probabilities and resulting profits under different price choices, a concept that is similar in structure to value estimation as it is performed in Q-learning; see Ye et al. (2018).

Kim et al. (2016) study Q-learning in energy markets. In such, the price choice is crucial for the service provider to control demand and maximize his profit. Their approach is to search for structurally similar states and use that information to update the Q-estimation at different

points. As their approach is market model specific, the generality of reinforcement learning is given up. Aside from pricing, reinforcement learning has proven itself in other challenges related to operations management, for example, supply chain management, as shown by Giannoccaro and Pontrandolfo (2002).

In contrast to our work, some studies analyze the reaction to customer behavior by RL based agents, for example, the study provided by Ghasemkhani and Yang (2018). They analyzed reinforcement learning in an environment in which demand management is necessary, but demand management based on all customers' data might violate the customer's privacy. They conclude that adaptive pricing algorithms can achieve similar without the explicit exchange of customer information.

Maestre et al. (2019) display an experiment where fairness is included in the measured reward. They consider a pricing model fair if it allocates the available goods equally between different groups of customers. It was shown, that it is possible to motivate an RL based agent to include fairness in his learning process.

The outcome of the "Price collusion in a duopoly" section displays that collusion between a self-learning agent and a manually configured one can be enforced by the competitor. This is consistent with other experiments in that area, which proposed similar effects for multiple self-learning agents; cf. Calvano et al. (2019).

In a recent publication, Bondoux et al. provide an extensive study on the possible use of reinforcement learning for airline revenue management; see Bondoux et al. (2020). They show that Deep Q-Networks provides a feasible algorithm for such systems. They apply DQN for simulations with competition. They conclude, that exploration costs have to be considered when applying RL, as increased forced exploration shortens the learning curve at the cost of higher losses during that period. Their simulation hides the competitor behavior and the current market state from the revenue management system, which prevents the direct reaction to competing market participants.

Many of those examples rely on relatively simple algorithms in the domain of RL, most notably Q-learning. Q-learning has seen many improvements in the past years, cf. artificial neural networks, experience replay, and several extensions focusing on overcoming known shortcomings of Q-learning, see Lillicrap et al. (2016), Schaul et al. (2016), Van Hasselt et al. (2016), Wang et al. (2016), Hessel et al. (2018). Many of those stay unmentioned and unused in the previous publications on dynamic pricing.

Policy gradient algorithms have seen further development as well. While the soft reinforcement learning algorithms do incorporate entropy as regularizing factor, other algorithms without that mechanism are available as well. For example, A3C might serve as an alternative to SAC, cf. Haarnoja et al. (2018); Mnih et al. (2016). Especially multi-task oriented algorithms might serve as an addition to our approach, as aggregation over different markets and products would increase the availability of training data. As every one of those markets might show different market dynamics, a well-performing algorithm needs to take this into account. Examples of such systems are IMPALA (Espeholt et al. 2018) and Distral (Teh et al. 2017), which will be evaluated in future work.

Background

Our goal is to evaluate the performance of two examples of RL algorithms on dynamic pricing problems. Two example algorithms from two different categories have been selected. The first algorithm is Deep Q-Networks (DQN), an implementation of Q-learning that uses artificial neural networks to estimate the value of an action in a given state of the environment and then selects the next action based on the value estimations. The second algorithm is Soft Actor Critic (SAC), which is a recent iteration in the group of policy gradient algorithms. It is based on two components, the actor and the critic. The actor represents a function that maps states to actions that will be pursued. The critic provides an estimator if the action chosen by the actor was chosen well, this estimation is used to adjust the actor. The next two subsections contain a deeper introduction to both algorithms.

Both algorithms require an environment that consists of a state-space S, an action space A, and a state transition function. A state transition occurs after the action choice of the agent and provides the agent with information about his reward $r \in R$. For all following experiments, we set $R = \mathbb{R}$. Those parameters allow the agent to collect data in the form of (s_t, a_t, r_t, s_{t+1}). The state transition probability $p_s(s_{t+1}|a, s)$ has to fulfill the Markov property. A similar function provides a probability distribution for the rewards, given s and a, $p_r(r|s_t, a_t)$. The transitions can be deterministic, which is the case for many of the experiments in the next sections. The agent performs an action in this environment based on a policy π. The policy can be either deterministic, $\pi(s; \phi)$, and return the action a given s or non-deterministic, $\pi_\phi(a|s)$, and return the probability of performing a given s.

In both cases, the policy has a set of parameters, ϕ, which can be adjusted to change the assignment of actions to states. The agent maximizes the long-term reward (G_t), i.e., the sum of rewards r_t (discounted by the factor γ), starting from a given state at time t. Thus, the goal is to find a policy π, that maximizes the expectation of

$$G_t = \sum_{k \geq 0} \gamma^k \cdot r_{k+t}.$$

Deep Q-networks (DQN)

In Q-learning, the goal is to provide an estimation of the long-term value of an action and follow the estimations by performing the action, that is maximizing this value. The expected long-term reward of an action a given the state s is the Q-value, cf. Watkins and Dayan (1992), i.e.,

$$Q_{s,a} = \mathbb{E}[G_t | s_t = s, a_t = a].$$

Artificial neural networks do provide a capable method of training an estimator for the Q-value. In that case, the function $Q(s, a)$ is represented by a neural network. The internal structure of Q can be chosen by the engineer. Usually, the neural network outputs the Q-values for all available actions at once, instead of using a as parameter, all definitions change accordingly. In all later examples, $Q(s, a; \phi)$ is a deep non-linear function, that relies on a chained application of linear operations like $Wx + b$ and the following application of a non-linear function like *relu*, see Mnih et al. (2015).

Given a set of tuples (s_t, a_t, r_t, s_{t+1}), the goal is to optimize Q to improve the estimation of the Q-value for all given actions. The minimization goal is the squared difference between the estimated Q-value and the experienced Q-value. As the future rewards are not known, the maximum Q-value of the following state is estimated and used instead. Then ϕ is adjusted by minimizing J using an optimizer like ADAM, see Kingma and Ba (2015) and Mnih et al. (2015),

$$J(s_t, a_t, r_t, s_{t+1}; \phi) = \frac{1}{|A|}(r_t + \gamma \max_{a \in A} Q(s_{t+1}, a; \phi) - Q(s_t, a_t; \phi))^2.$$

To derive a policy from this estimation, while the agent is facing the simulation, the Q-values are computed for the current state s and all actions. The action with the highest estimated Q-value will be performed. Enforced random exploration is used to improve data collection Mnih et al. (2015) (U is a uniform distribution)

$$\pi(s) = \begin{cases} U(A), & \text{if } U([0, 1]) \leq \epsilon \\ \arg\max_{a \in A} Q(s, a; \phi), & \text{otherwise.} \end{cases}$$

The implementation used for the experiments in Sects. "Performance in duopoly environments" and "Application in oligopoly competition" include two extensions for DQN, i.e., Double Deep Q-Networks and Dueling Deep Q-Networks. Double Deep Q-Networks introduces a second network to compute the target values, it reduces the positive bias that is introduced by the maximization operator in the computation of target values, see Van Hasselt et al. (2016). Dueling Deep Q-Networks introduces a change in the neural network by computing two components of the Q-value. The last layer computes the value and the advantage instead of a single Q-value. The value is the expected reward of being in a certain state. The advantage is the expected additional reward that can be achieved by performing a certain action, cf. Wang et al. (2016). We have

$$Q_{s,a} = V_s + A_{s,a} \quad with \quad V_s = \mathbb{E}[G_t | s_t = s],$$
$$A_{s,a} = \mathbb{E}[G_t | s_t = s, a_t = a] - \mathbb{E}[G_t | s_t = s].$$

The resulting setup uses two neural networks that implement the dueling structure in their last layers. This setup is not domain-specific.

Soft actor critic (SAC)

SAC is a policy gradient algorithm, that is centered on the idea of having a parametrized policy, that can be optimized given information about its expected performance. It consists of an actor, which is the policy itself, and a critic, which provides a feedback function for the actor. The critic provides an estimation of the value of an action, its gradient regarding the action can be followed to find a better performing action. The actor is a neural network that outputs the action for a given state. The policy is then updated using the gradient on the action from the critic to adjust the parameters of the actor. Actor critic can be applied to continuous action spaces, while DQN requires the discretization of the action space. In SAC, the optimization occurs not only intending to maximize the value but also the entropy of the non-deterministic policy. The non-determinism is achieved using the output of the actor to configure the parameters of a probability distribution, cf. Haarnoja et al. (2018),

$$V(s) = \sum_{a \in A} \pi_\phi(a|s)(Q(s, a) - \alpha \log \pi_\phi(a|s)) \tag{1}$$

$$V(s) = \mathbb{E}[Q(s, a) - \alpha \log \pi_\phi(a|s)]. \tag{2}$$

This allows a policy to be optimal even without maximum Q-values as long as it keeps high entropy. This approach naturally motivates exploration and allows the algorithm to focus on multiple near-optimal solutions instead of finding one of the few optimal deterministic policies, cf. Haarnoja et al. (2018). To train an estimator for Q, the loss can be measured accordingly:

$$J_{\mathrm{QL}}(r_t, a_t, s_t, s_{t+1}; \phi_q, \phi_v) = (r_t + \gamma V(s_{t+1}; \phi_v) - Q(s_t, a_t; \phi_q))^2.$$

As Q implicitly requires an estimation of the state value, it is required to have a second estimator:

$$J_{\mathrm{VL}}(r_t, s_t, s_{t+1}; \phi_v) = (r_t + \gamma V_\phi(s_{t+1}; \phi_v) - V_\phi(s_t; \phi_v))^2.$$

Both networks can be optimized using the same algorithms that have been used in DQN. To further improve the

stability of the results, two different estimators are kept for the Q-value, the minimum output of both is used whenever a Q-value is required, cf. Haarnoja et al. (2018). To optimize the actor, the loss can be represented as the Kullback–Leibler divergence (cf. D_{KL}) between the current policy and the optimal policy under the current Q-value estimations, cf. Haarnoja et al. (2018), where $Z(s_t)$ is a normalizing constant within each state, i.e.,

$$J(\phi) = \mathbb{E}\left[D_{\mathrm{KL}}\left(\pi_\phi \| e^{Q(s_t,a_t)}/Z(s_t)\right)\right].$$

To minimize J, the gradient can be computed as follows, using f which is a reparametrized version of the policy, cf. Haarnoja et al. (2018):

$$\begin{aligned}\nabla J(\phi) =& \nabla_\phi \alpha \log(\pi_\phi(a_t|s_t)) + (\nabla_a \alpha \log(\pi_\phi(a_t|s_t)) \\ & - \nabla_a Q(s_t, a_t))\nabla_\phi f_\phi(\epsilon_t, s_t).\end{aligned}$$

The importance of the entropy in the soft value function is reflected by the parameter α in (1) and has to be chosen carefully. High values increase stability but decrease overall performance. It is possible to determine α automatically, see Haarnoja et al. (2018) for more information.

Performance in duopoly environments

In a duopoly market, the agent competes with a single competitor. The agent has to understand the customer behavior as well as the competitor's reaction.

Experimental setup

All model inputs to *reproduce* the calculation of all allocations are available online, see Source Code (2020). In our model, time is split into periods of fixed length, which start with the agent's price update. After the first half of the episode, the competitor reacts to the agent's action and updates his price (cp. the setup used in Schlosser and Richly (2019)).

The behavior of the customer is modeled by a logistic model, which is based on real-world data, cf. Schlosser and Boissier (2018). It returns the probability of having a sale in a given period according to several features, including price rank and difference to the competitor price. The overall customer behavior reflects that lower price levels increase sales and price rank is the most noticeable feature. To maximize revenue, the agent has to find both an adequate price level and undercut his competitor to increase sales. An overview of all features available is given in Table 1. Note, for the price rank, we use the indicator function, cf. $1_{\{\cdot\}}$. Sales are computed for each half of the episode for both participants. An example of the expected

Table 1 Features (for logistic demand; Schlosser and Richly (2019)) to calibrate demand probabilities; p is the own price, o is the price of the competitor

Name	Computation	β
Constant	1	−3.82
Price rank	$1 + 0.5 \cdot 1_{\{p=o\}} + 1_{\{p>o\}}$	−0.56
Price difference	$p - o$	−0.01
Average market price	$(o + p)/2$	−0.03

The amount of realized sales is computed vice versa for both participants

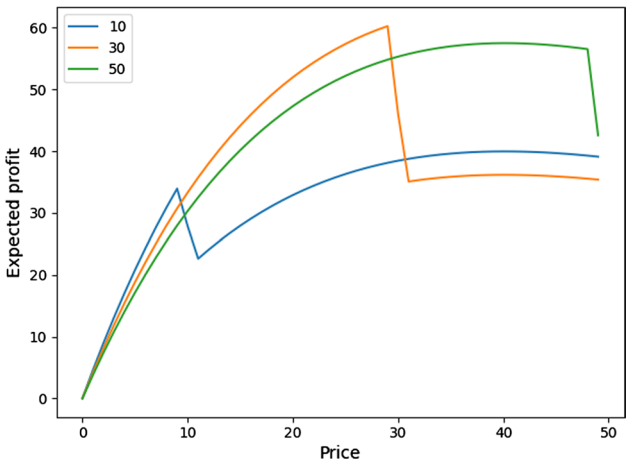

Fig. 1 Expected profit in one episode for different own prices at a fixed competitor price, cf. 10, 30, 50

revenue in this model in different market situations is displayed in Fig. 1.

The state-space in this game consists of only a single feature, the current competitor price. The action space consists of all possible prices. The price range for both competitors is one to 50. For DQN, this price range is discretized in steps of size 1. SAC does not require any discretization of the action space.

To maximize his reward, the agent has to react accordingly to the competitor's strategy. In many setups, the competitor uses a simple strategy that was manually tuned by humans. The agent has to correctly estimate the competitor's reaction to maximize his reward. He has to set a price that is forcing the competitor toward an unprofitable reaction and allows a sustainable high price level in the market. This necessity distinguishes the following experiments from other evaluations of RL on dynamic pricing, cf., e.g., Bondoux et al. (2020).

Two groups of manually tuned competitor strategies are available, deterministic and non-deterministic. The second group was introduced to provide an additional challenge for competitors in the market, as the behavior of a trader becomes less predictable. Both algorithms have been

evaluated in the same simulation against different competitors to analyze strengths and weaknesses.

In duopoly cases, one can compute *optimal* reaction strategies using *full information* based on dynamic programming, e.g., described in Schlosser and Richly (2019), Lemma 3.1, cf. *DP solution*. Note, this becomes impossible for larger state-spaces, but the duopoly provides a possibility to verify the performance of self-learned strategies. The strategies of the agent are evaluated by comparing their expected long-term reward with the expected long-term reward of the respective optimal strategy.

All experiments on the duopoly have been performed 10 times with a length of 500,000 episodes. The setup for DQN uses a neural network with three hidden layers with an output vector with 128 elements each. The last two layers compute advantage and value independently. It outputs the Q-values for all actions in one vector. SAC uses three chained layers with 128 nodes for the critic and the actor. Both algorithms do incorporate experience replay to reuse older data.

Deterministic competitors

We consider three common types of deterministic strategies: fixed price (Sect. 3.2.1), undercutting (Sect. 3.2.2), and two-bound strategies (Sects. 3.2.3–3.2.4).

Fixed price

Such a situation occurs, if the competitor sticks to a fixed price. In that case, the most advantageous strategy is to slightly undercut the competitor at all times, as long as the competitor's price is at a level that allows sustainable profit.

In this case, DQN can outperform SAC. No convergence can be observed in the strategies of SAC. SAC consistently converges toward the upper end of the action space at the competitor price, which diminishes the strategy value. We assume that this behavior is caused by the entropy term of the optimization. If the Q-values become very low, the motivation to follow the Q-value might be superimposed by the motivation to increase entropy. If this effect is not stopped, no convergence toward a well-performing strategy occurs.

DQN on the other hand, finds well-performing strategies, see Fig. 2. While having a high peak performance, repeated deviation from the optimal strategy causes a huge span in the performance of DQN at any point in time, which renders it unreliable.

Unlimited two-bound (undercutting)

A common deterministic strategy undercuts the agent by a fixed difference if possible, for example by one-price unit. If the agent would follow a similar strategy, this would lead to a tie, after one reaches his minimum price level. This setup is the specialization of a two-bound enemy, called unlimited two-bound. A two-bound strategy $\pi_o(p)$ has a lower limit p_l, which is never undercut and an upper limit p_u, which is used if the competitor undercuts p_l:

$$\pi_o(p) = \begin{cases} p-1, & \text{if } p-1 \geq p_l \\ p_u, & \text{otherwise.} \end{cases} \tag{3}$$

The optimal strategy in the given environment against an unlimited two-bound uses two-bound on its own, with limits specific to the demand function. To stay within a price range that achieves decent profitability, the agent should avoid reducing the price level to values below 24, see Fig. 3. This happens because the competitor can diminish the maximum achievable revenue by choosing a low price, see Fig. 1.

DQN showed noticeable issues in terms of stability while competing with this competitor. While finding

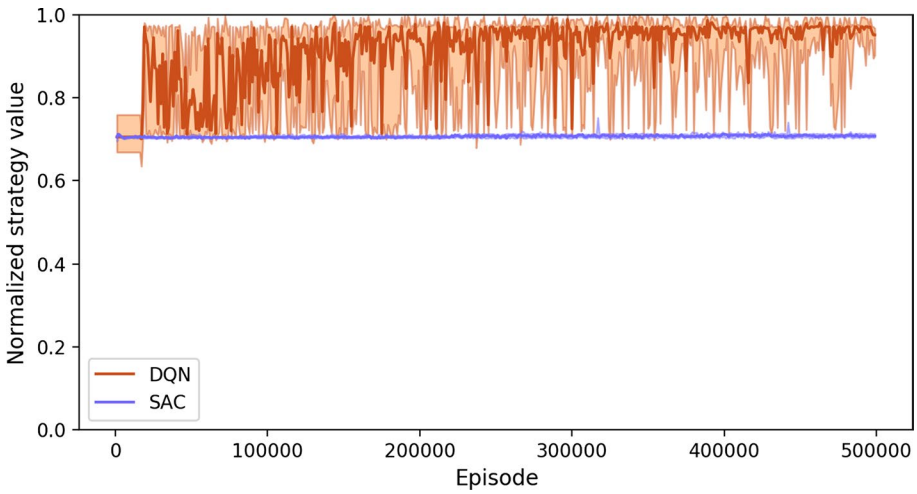

Fig. 2 Median performance with .2 and .8 quintile over 500K learning episodes of DQN vs SAC with a competitor that uses a fixed price; "Fixed price" section

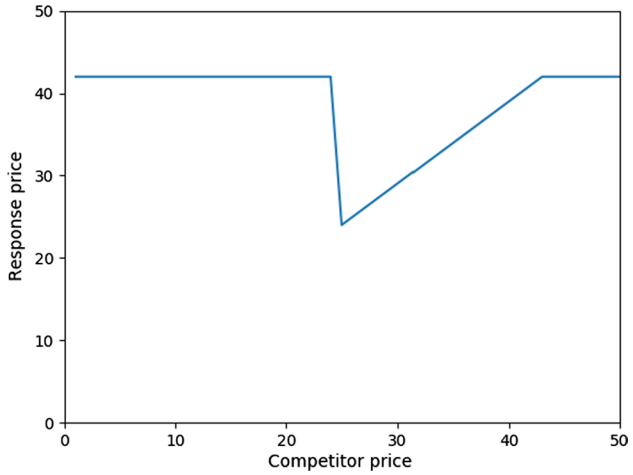

Fig. 3 Structure of an *optimal* strategy (obtained via DP) against an unlimited two-bound; "Unlimited Two-bound (Undercutting)" section

well-performing strategies, they get replaced within a few episodes, explaining the noticeable spread in performance visible in Fig. 4.

One explanation for this effect can be found in the fact that DQN struggles with correctly estimating the long-term behavior of the competitor, an analysis that will be proven in later experiments. DQN tends to stick with state-independent one-price strategies. As reacting with a one-price strategy is not sufficient in this setting, this provides an upper bound on the reward that can be achieved. A complex strategy is considered one that uses a noticeable share of the available action space if necessary. SAC is more familiar with complex strategies, as they are easily encoded within its actor. The problems of DQN might be caused by two structural challenges. The Q-values in this domain lie close together, especially because DQN was used with a high discount $\gamma := 0.999$. This makes even small errors in the estimation crucial and slightly overestimating a single action leads to the problems described above. While it handles problems that require complex strategies more efficient than DQN does, SAC has noticeable issues in problem areas where fixed price strategies are the best.

Limited two-bound

A limited two-bound, specified by (3), requires nearly the same response strategy as an unlimited one, assuming that the lower limit p_l of the two-bound is set to a value below 24. In this case, the resulting behavior will be equal to the behavior with open bounds. Accordingly, the measured performance is similar to the results from the previous experiments.

Exploitable two-bound

A two-bound strategy becomes exploitable if it is possible to undercut p_l and still achieve a decent amount of sales at a decent price. If this is done, the competitor has to stick to his upper limit, which highly increases the price difference between both competitors. This requires a fixed price strategy, as it is necessary to react with p_l at all times to avoid ending up in an undercutting cycle.

DQN outperforms SAC in such situations, as this is one of those use cases where the preference for one-price strategies of DQN becomes an advantage (Fig. 5). SAC can find well-performing strategies, but requires more training data and is unable to outperform DQN at any time. DQN tends to follow strategies, that lead to a 24/45 market state relatively fast, an example is given in Fig. 6. SAC tries to undercut the competitor in many situations while exploiting his strategy in others, an example strategy found throughout the training that illustrates this is displayed in Fig. 7.

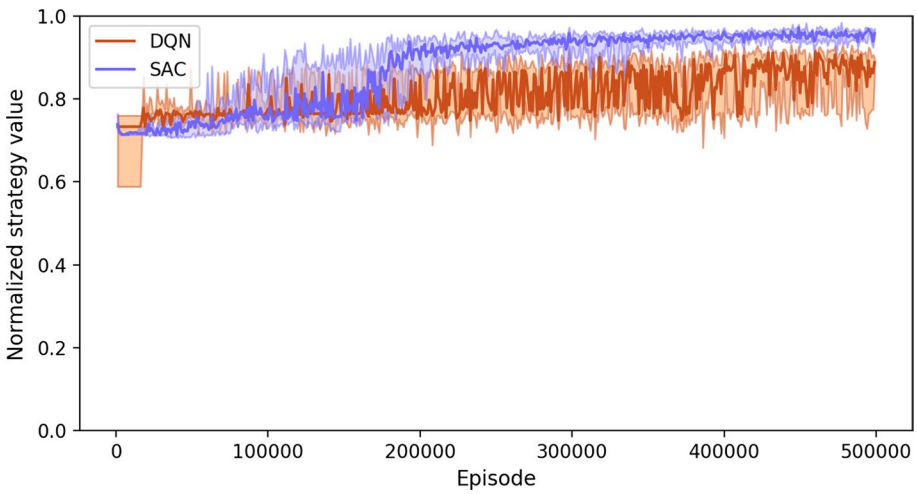

Fig. 4 Median performance with .2 and .8 quintile over 500K learning episodes of SAC and DQN (compared against optimal DP solution, with a competitor that follows a two-bound strategy as specified in (3) with $p_l = 1$, $p_u = 50$); "Unlimited Two-bound (Undercutting)" section

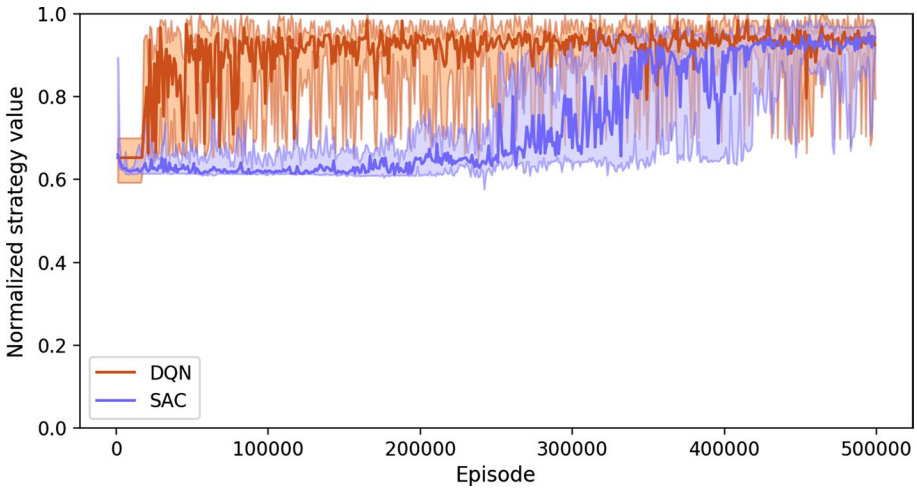

Fig. 5 Median performance with .2 and .8 quintile over 500K learning episodes of DQN vs SAC against the two-bound strategy with 25/45 limits; "Exploitable two-bound" section

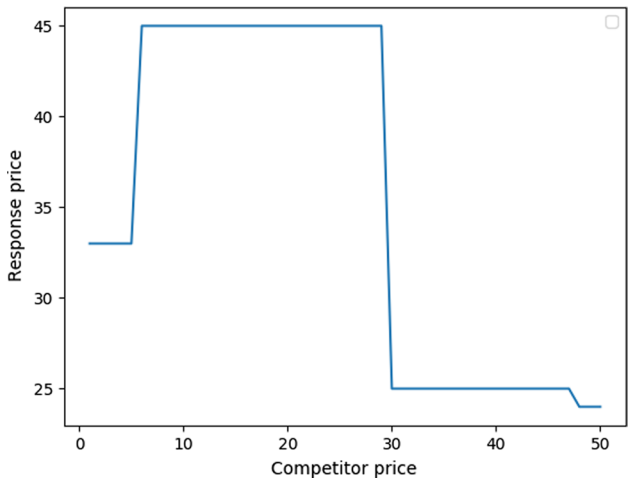

Fig. 6 A DQN example strategy that achieves 99% relative performance against the exploitable 25/45 two-bound competitor. It leads to a pattern where the reaction of the agent is either 33 or 45, which is countered by the competitor with 32 or 44. In both cases, the reaction of the agent is 25, which then leads again to a reaction of 45 by the competitor; "Exploitable two-bound" section

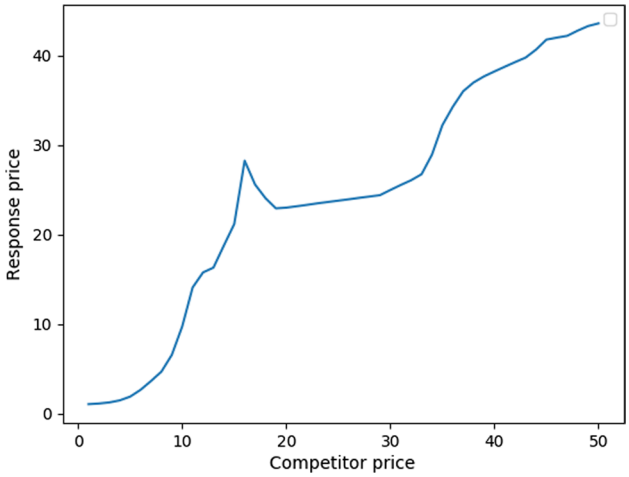

Fig. 7 A SAC strategy with 92% relative performance against the exploitable competitor with bounds 25/45. While reacting with a price around 25, it follows the undercut pattern at a price of circa 40 and above, which provides the competitor with a realistic chance of competition; "Exploitable two-bound" section

Non-deterministic competitors

All competitors introduced in the last section are deterministic. In markets where one or more market participants can be expected to use algorithms that try to anticipate the competitor's reactions, it might be advantageous to introduce non-deterministic strategies to increase the difficulty of that estimation.

Different concepts for non-deterministic strategies are available. One possibility is using a randomized strategy that does not depend on state of the market. Alternatively, the competitor might also use a mix of several deterministic strategies or add noise to a single deterministic strategy, to hide it from the agent. In the second case, depending on the noise pattern applied, it is possible to estimate the underlying structure of the competitor, given a large number of observations available. Such a situation might occur as well if the competitor is using RL on his own, especially those approaches that produce non-deterministic parametrized strategies like SAC.

Random price choice

In case of a fully randomized competitor, which is only constrained to a lower and upper limit price limit, long-term effects become irrelevant to the agent. The next state does not depend on the last one at any point. A correct reaction to the current state becomes the only required ability to perform well. Because of this effect, this setup

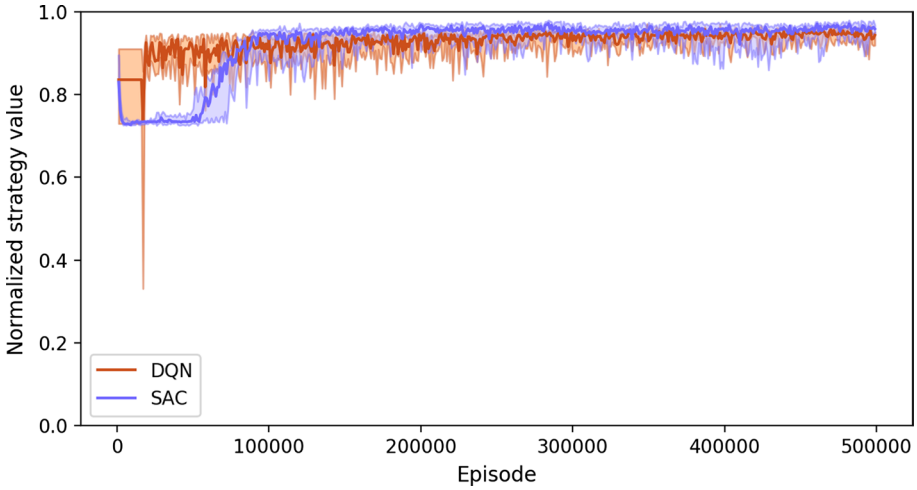

Fig. 8 Median performance with .2 and .8 quintile over 500K learning episodes of DQN vs SAC with a competitor, that chooses a random price in the price range of 10 to 50; "Random price choice" section

can display the struggle of DQN to correctly estimate long-term behavior. Figure 8 displays the relative performance of both agents. DQN achieves a high level of performance fast and does not have noticeable stability issues that could be observed against complex deterministic strategies. SAC requires more observations to achieve only slightly improved results.

Noised strategies

In the case of a noised strategy, a deterministic strategy is used and the chosen price is slightly adjusted by adding noise. An example configuration for this setup is a combination of the two-bound from the last experiments with the addition of values drawn from a normal distribution. π describes such a noised policy, with $\mathcal{N}(0, \sigma)$ representing a value drawn from a normal distribution and π_{orig} being the deterministic strategy of the opponent:

$$\pi(s) = \pi_{\text{orig}}(s) + \mathcal{N}(0, \sigma). \tag{4}$$

This situation offers the same general challenges to the agent that have been present against a two-bound competitor. It is expected that the agent has to collect more data to correctly estimate the underlying deterministic strategy and then achieves the same results as before. Figure 9 shows that this expectation does fully hold. In general, SAC outperforms DQN, as was the case for a normal two-bound, but the performance of DQN is more stable in general. The maximum performance achieved by DQN is slightly lower than in the deterministic use case, whereas the minimum performance is slightly increased.

In the last experiments, it was shown that there are some exceptional situations, in which Q-learning outperforms SAC. As those situations required special preconditions or competitors that do not fully follow business logic, e.g., by randomizing their price choices as a whole, it can be concluded that SAC and future iterations of policy gradient algorithms provide a well-performing basis to build pricing

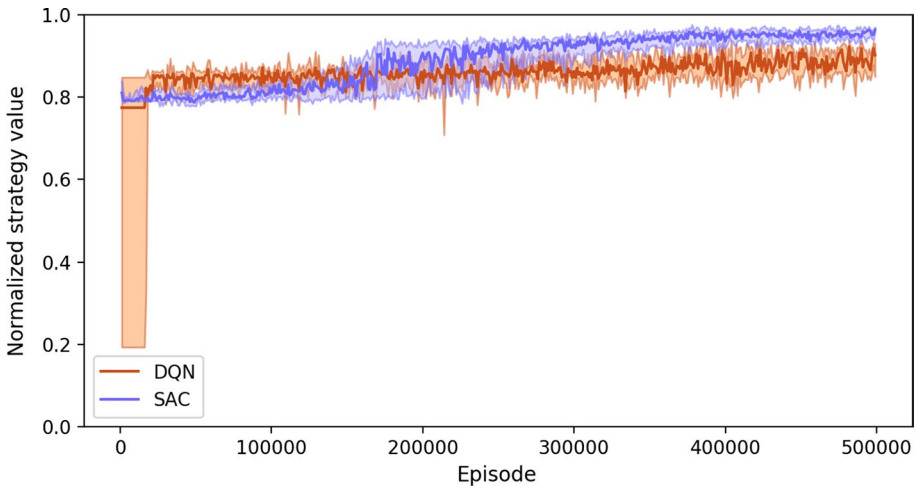

Fig. 9 Median performance with .2 and .8 quintile over 500K learning episodes of DQN vs SAC with a competitor, that applies noise; "Noised strategies" section

systems. This result should be repeated in a more complex market.

Price collusion in a duopoly

As long as the competitor in the duopoly is directly competing with the agent, he has an incentive to join the competition by undercutting. In practice, there is the possibility to make an implicit offer of cooperation between both market participants. In such a case, the competitor is configured to answer a specified cartel price, e.g., 33. In this simulation, 33 is the price that yields the highest possible expected profit if both market participants use the same price. The cooperating competitor that agrees to a cartel can increase the agents' motivation by following a dumping price two-bound strategy until the agent agrees to the cartel.

Figure 10 displays the performance of SAC and DQN in the case that the competitor does offer a cartel at 33 but behaves like an unlimited two-bound in all other cases. In such a situation, competing can slightly outperform agreeing to a cartel. SACs ability to compete efficiently then outperforms the tendency of DQN to agree to the cartel.

If the competitor is performing price dumping until the agent agrees to the cartel, this advantage is not given anymore. Following his tendency to use undercutting, SAC misses the opportunity given by the cartel. This can be observed in the results in Fig. 11, which yields better results for DQN than in most other experiments.

The cartel experiments show a noticeable effect in markets that are driven by self-learning algorithms. It proofs, that both market makers don't need to exchange any information but price reactions, to allow both of them to form a cartel and drop competition. Naturally, this leads to an increased price level for the customer. The experiments on two-bound show a similar effect, where the agent does avoid diminishing the price level in the market and offers his competitor a return to a high price level at his own cost. This mechanism is still more desirable by the customer, as the

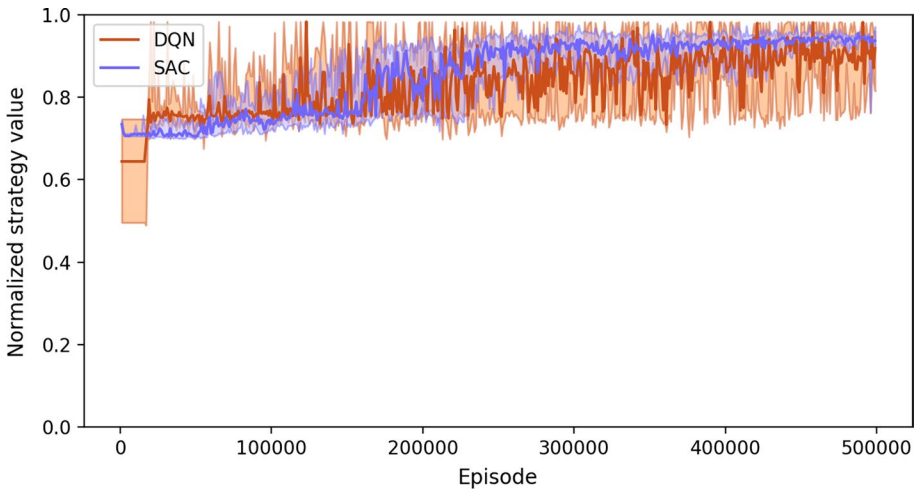

Fig. 10 Median performance with .2 and .8 quintile over 500K learning episodes of DQN vs SAC with a competitor, that offers a cartel price at 33 and undercuts by one in all other situations; "Price collusion in a duopoly" section

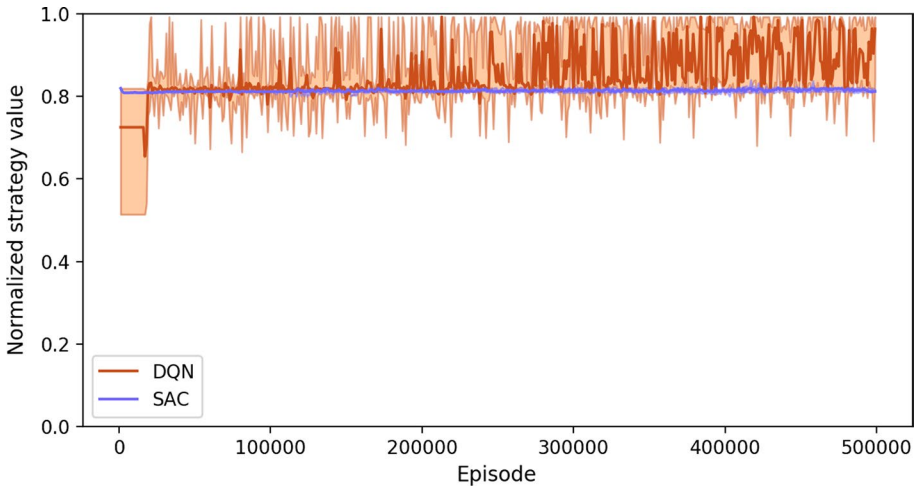

Fig. 11 Median performance with .2 and .8 quintile over 500K learning episodes of DQN vs SAC with a competitor, that offers a cartel price at 33 and undercuts by one within a range of 10 to 20. The competitor does never exceed this price range; "Price collusion in a duopoly" section

lower limit of the agent is lower than the cartel price that might have been used otherwise.

Competition between different self-adaptive learning strategies

The duopoly scenario introduces the possibility of direct competition between both algorithms. As the strategy of both agents changes over time, the Markov property is violated. To reduce this effect, training of both algorithms does not occur at the same time. Instead, training occurs in turn. Each algorithm can optimize its policy for 50,000 episodes to find a strategy against the fixed policy of his competitor, before being fixed for 50,000 episodes. Agent *a* uses SAC, agent *b* uses DQN. The experiments lasted over 1,000,000 episodes, with 10 training cycles. For evaluation, the average achieved rewards of both agents have been measured. Figure 12 displays the results.

No algorithm was able to continuously outperform his competitor, the length of the training episode allowed it for both algorithms to find a well-performing strategy relatively fast. No other reliable pattern can be observed in the earnings.

Schlosser and Richly (2019) observed repeating patterns of strategies when dynamic programming was competing with itself. The results of several executions of this setup did not yield strategies that contained human visible patterns. Agent *b* showed a strong tendency toward one or two price strategies, usually in a reasonable price range between 20 and 50. His competitor, the SAC agent, focuses on finding competing strategies against those immediately. This can be observed in Figs. 13 and 14. This figure shows that DQN can explore more. This effect is encouraged by the fact that the SAC agent does respond with more different prices then DQN does. This better response behavior provides DQN with the opportunity to collect information about more price combinations. This does not prevent DQN from sticking to

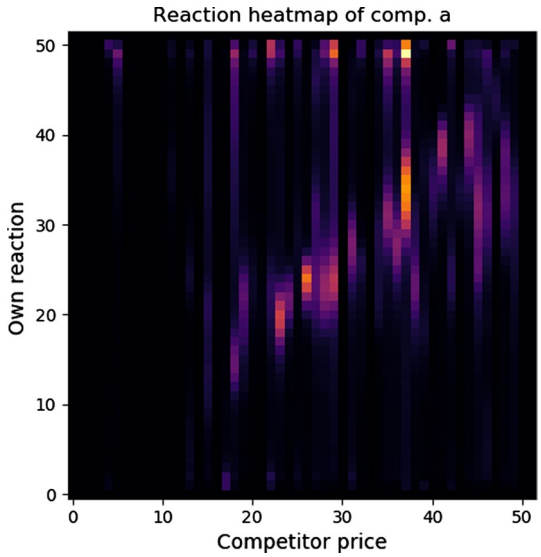

Fig. 13 Price response probabilities of SAC during a training phase against prices of DQN, "Competition between different self-adaptive learning strategies" section

fixed price strategies, which can be observed in the heatmap for the training phase of SAC. The competitor price range during the training of SAC is always limited to a single vertical price band. The exploration of SAC occurs only on this price range, which limits the number of different price combinations that can be observed. SAC finds near-optimal response strategies fast. SAC tends to explore the upper end of the price range, independent of the opponent price. This effect increases at the lower end of the price range.

Both algorithms tend to develop strategies against their counterparts relatively fast and continue to outperform their competitor. With the given setup, no equilibrium of strategies could be observed, with no cycles being available either.

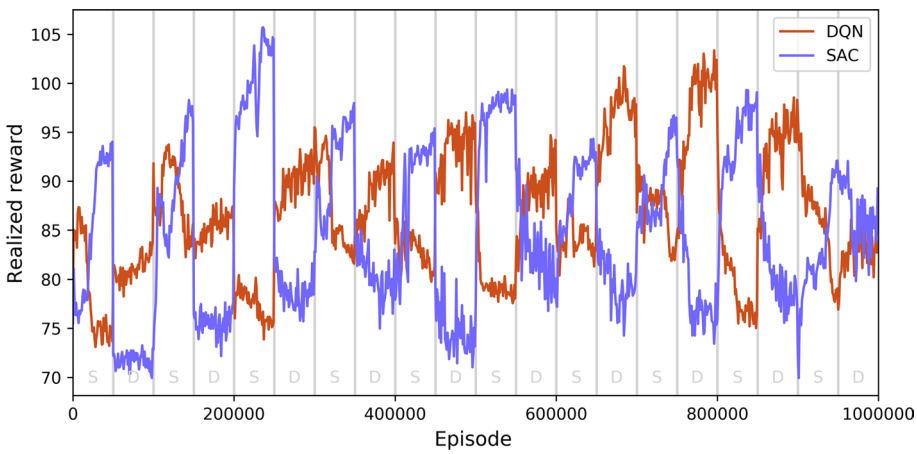

Fig. 12 Average reward measured using a moving average over 500 episodes (10 training iterations) for both agents aggregated over 10 experiments; "Competition between different self-adaptive learning strategies" section

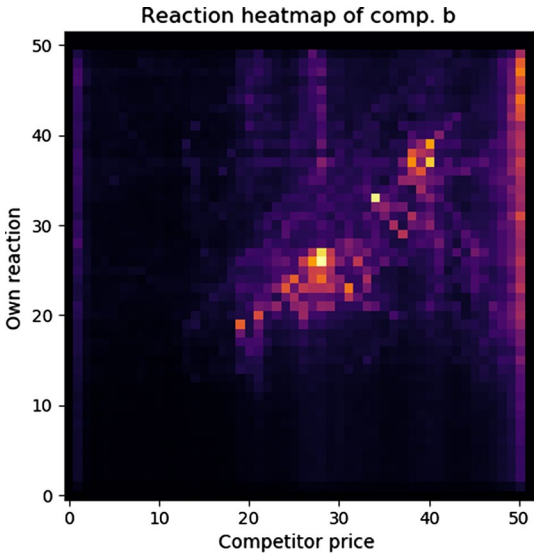

Fig. 14 Price response probabilities of DQN during a training phase against prices of SAC, "Competition between different self-adaptive learning strategies" section

Table 2 Competitor setup in the oligopoly case, for more implementation details see Source Code (2020)

#	Type	Parameters	Delay μ
1	Two-bound	$p_l = 1$, $p_u = 50$	0.7
2	Two-bound	$p_l = 15$, $p_u = 45$	1.2
3	Fixed	$p = 19$	1.1
4	Random	$p_l = 1$, $p_u = 50$	0.6
5a	50% two-bound	$p_l = 20$, $p_u = 40$	1.6
5b	& 50% random	$p_l = 10$, $p_u = 50$	-

Application in oligopoly competition

In practice, many markets do not exist in the form of duopolies. For most products, there exists more than a single competitor, and each competitor in the market is going to follow slightly different pricing strategies. Such situations cannot be analyzed by the solution approach that was used to evaluate the duopoly market because the curse of dimensionality makes it intractable. The oligopoly market can only be analyzed in terms of achieved average reward per episode.

In the duopoly it was assumed that at the end of an episode all market participants had performed the same amount of price updates, this assumption is dropped for an oligopoly. As many more price updates occur, an intuitive strategy for each market participant is to observe the market at some point, compute a reaction to that and then wait for some time. Technical limits, for example API update rates, might make it impossible to react to each competitor's price updates, especially because the market participants might compute their price updates simultaneously. Because of this, it is assumed that every market participant updates his price after a fixed period in reaction to the current market situation at this point. The competitors' update rates are displayed in Table 2.

Customers arrive at a certain rate and then decide for a market participator. The customer usually decides for the cheapest competitor. Random noise is applied to the prices before the selection to compute the rating of the customer; see Source Code (2020). This allows it for two competitors at a similarly low price level to have a realistic chance of achieving a sale. An interested customer arrives on average every 0.1 units of time (cf. episodes).

In the example shown in Fig. 15, several competitors do compete in the market. The strategies used by the competitor include fixed price strategies, for example by competitor 3, as well as exploitable and unlimited two-bounds. Also, there is one competitor that randomly chooses between two strategies at each decision point. Table 2 provides a detailed overview.

We allow the agent to update on average every $\mu = 1.0$ units of time, while the competitors have higher/lower (slightly randomized) update frequencies (all normal

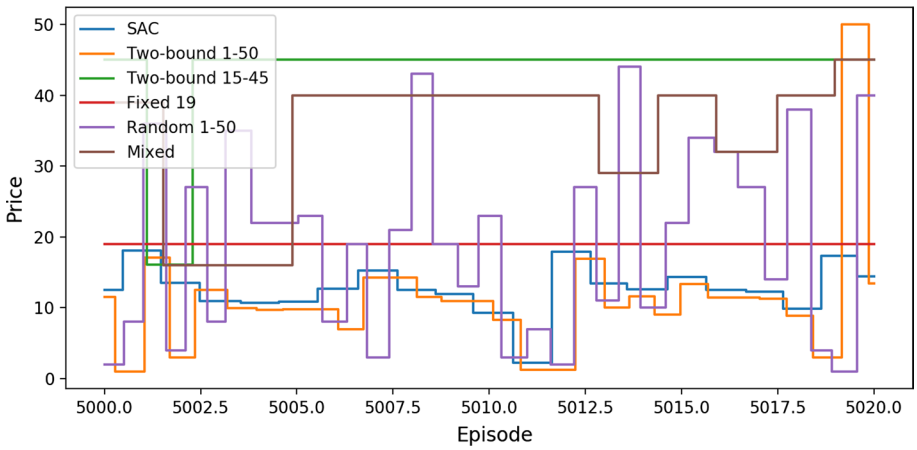

Fig. 15 Price trajectories recorded during the training phase of a SAC agent in an oligopoly; "Application in oligopoly competition" section

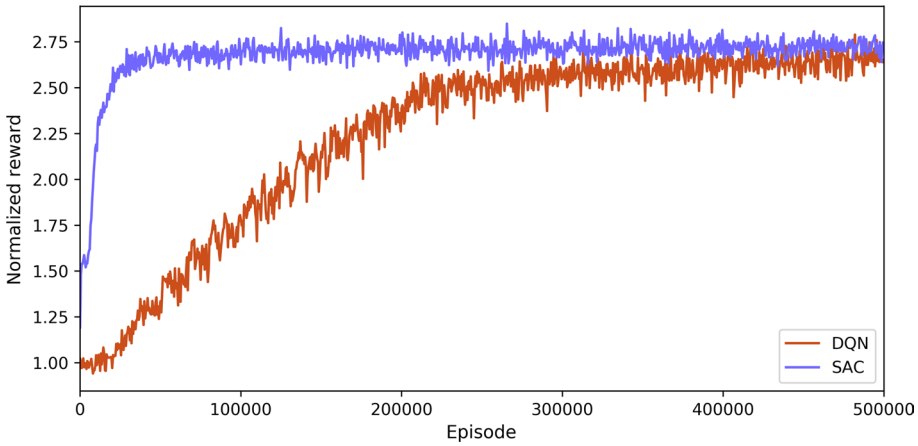

Fig. 16 Reward averaged over 500 episodes and 10 runs throughout a learning time of 50K episodes of SAC and DQN in an oligopoly. Rewards are normalized by the reward of 7.63 per episode achieved by a uniform $U([1, 50])$ random price choice; "Application in oligopoly competition" section

distributed with standard deviation of 0.05). He can make use of the exploitation strategy to diminish the performance of competitor 2 while he has to directly compete with the unlimited two-bound. The randomized competitor adds noise to the state information and might diminish the overall price level on the market by randomly choosing low prices.

Given those preconditions, both agents have been challenged with competing in that market. In general, both outperformed their fixed strategy competitors at some point, cf. Fig. 16, where the results are normalized by the rewards achieved by a benchmark strategy playing random prices via uniform $U([1, 50])$. Other market setups might yield different results. In general, DQN requires more episodes but overall achieves a similar level of average performance at some point. SAC is the algorithm of choice, as it converges to a high level of reward fast and can keep this level over long periods.

Conclusions

We studied pricing competition motivated by online markets in order to provide insights for practitioners to assess in how far reinforcement learning can be used to automate frequent repricing in a self-adaptive way. The first set of experiments showed several strengths and weaknesses of DQN and SAC when applied to pricing simulations. While non-deterministic competitor behavior does not offer a noticeable challenge for both algorithms, several special cases could be found that easily diminish an algorithm's performance. SAC struggles in those cases, where a simple, fixed price strategy would be optimal, while the major challenge for DQN lies in those cases where complex strategies are required to react to a lot of different situations accordingly. Both algorithms handle more complex scenarios with many competitors well.

We find that RL is a suitable alternative to specialized pricing algorithms. Both algorithms do not require any domain knowledge to be set up. Further, hyperparameter tuning was required to a smaller extent for SAC than for DQN. With the configuration used in the experiments, both models work well on different market setups. The most notable disadvantage of both algorithms is that they require a large number of observations to perform well. While both algorithms found well-performing strategies fast, in some use cases it took 400K episodes to achieve decent performance.

In comparison to classical approaches like dynamic programming (DP), RL requires less domain knowledge and input data. DP requires several estimations as input, for example, an estimation about the competitor behavior and the demand model. The optimal solution will only be optimal within those estimations. Recall, RL does not require any such specialized models to work. DP becomes impractical for any complex market setups, e.g., in large oligopolies.

For future research, a deeper evaluation of more recent algorithms should be performed. It should be investigated, if an increase in data efficiency is possible for DQN or SAC, as the state-space is smaller than the number of observations required to achieve peak performance. Further, the analyzed RL strategies can also be applied in finite horizon problems when selling a finite inventory of perishable products. Then, for collecting data and adapting strategies pooled or repeated simulation runs will have to be considered. In this context, in future work, we will investigate whether information gained in similar markets for different products can be used for multi-task reinforcement learning. If possible, this magnifies the amount of data available for training. Several alternative approaches to implement such a system are available at this point in time, as they have been mentioned in the "Related work" section.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

Bondoux, N., A.Q. Nguyen, T. Fiig, and R. Acuna-Agost. 2020. Reinforcement learning applied to airline revenue management. *Journal of Revenue and Pricing Management* 19: 332–348.

Calvano, E., G. Calzolari, V. Denicolò, and S. Pastorello. 2019. Artificial intelligence, algorithmic pricing and collusion. *American Economic Review* 110: 3267–3297.

den Boer, A.V. 2015. Dynamic pricing and learning: Historical origins, current research, and new directions. *Surveys in Operations Research and Management Science* 20: 1–18.

Espeholt, L., H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu. 2018. IMPALA: Scalable distributed deep-rl with importance weighted actor-learner architectures. *Proceedings of the 35th International Conference on Machine Learning, in PMLR* 80: 1407–1416.

Ghasemkhani, A., and L. Yang. 2018. Reinforcement learning based pricing for demand response. In: *2018 IEEE international conference on communications workshops, ICC workshops 2018-proceedings*. 2018 (pp. 1–6). Kansas City, MO, USA: IEEE.

Giannoccaro, I., and P. Pontrandolfo. 2002. Inventory management in supply chains: A reinforcement learning approach. *International Journal of Production Economics* 78: 153–161.

Gosavi, A., N. Bandla, and T.K. Das. 2002. A reinforcement learning approach to a single leg airline revenue management problem with multiple fare classes and overbooking. *IIE Transactions (Institute of Industrial Engineers)* 34: 729–742.

Gosavi, A. 2004. A reinforcement learning algorithm based on policy iteration for average reward: Empirical results with yield management and convergence analysis. *Machine Learning* 55: 5–29.

Haarnoja, T., A. Zhou, P. Abbeel, and S. Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th international conference on machine learning, ICML 2018* (pp. 1861-1870). Stockholm.

Hessel, F.N., et al. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), New Orleans, LA: AAAI.

Kephart, J.O., and G. Tesauro. 2000. Pseudo-convergent q-learning by competitive pricebots. In *Proceedings of the seventeenth international conference on machine learning, ICML '00* (pp. 463–470). San Francisco, CA.

Kim, B.G., Y. Zhang, M. Van Der Schaar, and J.W. Lee. 2016. Dynamic pricing and energy consumption scheduling with reinforcement learning. *IEEE Transactions on Smart Grid* 7: 2187–2198.

Kingma, D.P. and J. Ba. 2015. Adam: A method for stochastic optimization. In Bengio, Y. and Y. LeCun, eds., *3rd international conference on learning representations, ICLR 2015*, San Diego, CA, USA.

Könönen, V. 2006. Dynamic pricing based on asymmetric multiagent reinforcement learning. *International Journal of Intelligent Systems* 21: 73–98.

Kutschinski, E., T. Uthmann, and D. Polani. 2003. Learning competitive pricing strategies by multi-agent reinforcement learning. *Journal of Economic Dynamics and Control* 27: 2207–2218.

Lillicrap, T.P., J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. 2016. Continuous control with deep reinforcement learning. In *4th international conference on learning representations, ICLR 2016*, San Juan, PR, USA.

Maestre, R., J. Duque, A. Rubio, and J. Arevalo. 2019. Reinforcement learning for fair dynamic pricing. In Arai, K., S. Kapoor, and R. Bhatia, eds., Intelligent systems and applications.

Mnih, V., K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518: 529–533.

Mnih, V., A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In Balcan, M.F. and K.Q. Weinberger, eds., *Proceedings of machine learning research* (vol. 48, pp. 1928–1937). New York: PMLR.

Rana, R., and F.S. Oliveira. 2014. Real-time dynamic pricing in a non-stationary environment using model-free reinforcement learning. *Omega (United Kingdom)* 47: 116–126.

Schaul, T., J. Quan, I. Antonoglou, and D. Silver. 2016. Prioritized experience replay. In *4th international conference on learning representations, ICLR*, 2016. San Juan, USA: PR.

Schlosser, R., and K. Richly. 2019. Dynamic pricing under competition with data-driven price anticipations and endogenous reference price effects. *Journal of Revenue and Pricing Management* 18: 451–464.

Schlosser, R., and M. Boissier. 2018. Dynamic pricing under competition on online marketplaces: A data-driven approach. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 705–714).

Source Code. 2020. Input data and scripts to reproduce the experiments. https://www.dropbox.com/s/fpo9kis0r3c95nd/rlpricing-master.zip.

Teh, Y.W., V. Bapst, W.M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu. 2017. Distral: Robust multitask reinforcement learning. *Proceedings of the 31st international conference on neural information processing systems*. NIPS'17 (pp. 4499–4509). Red Hook, NY: Curran Associates Inc.

Van Hasselt, H., A. Guez, and D. Silver. 2016. Deep reinforcement learning with double Q-Learning. AAAI, Phoenix, AZ, USA (pp. 2094–2100).

Vengerov, D. 2008. A gradient-based reinforcement learning approach to dynamic pricing in partially-observable environments. *Future Generation Computer Systems* 24: 687–693.

Wang, Z., et al. 2016. Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd international conference on machine learning, ICML 2016* (pp. 1995–2003). New York, NY, USA.

Watkins, C.J., and P. Dayan. 1992. Q-learning. *Machine Learning* 8: 279–292.

Ye, P., J. Qian, J. Chen, C.-H. Wu, Y. Zhou, S. De Mars, F. Yang, and L. Zhang. 2018. Customized regression model for airbnb dynamic pricing. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 932–940). KDD '18. New York, NY, USA: Association for Computing Machinery.