Algorithmic collusion in multi-agent reinforcement learning

By L.Q. Segers *

This paper explores whether the presence of algorithmic heterogeneity changes the degree of tacit collusion in multi-agent reinforcement learning systems. An experiment is conducted where two distinct reinforcement learning algorithms - notably Q-learning and Tree-Backup(λ) - are paired in four different settings to play a pricing game. When the algorithms are paired with an identical opponent, the two algorithms manage to reach similar levels of tacit collusion and learn incentive compatible deviation and punishment strategies to sustain coordination. The Tree-Backup(λ) algorithm learns more efficiently, but it is less robust than Q-learning. In the heterogeneous setting of the experiment market outcomes are asymmetric. The Q-learning agent adopts a dominant market position compared to the Tree-Backup(λ) algorithm. Overall, this implies that high levels of algorithmic collusion may be a less sustainable market outcome than earlier experimental work on algorithmic pricing in multi-agent Q-learning suggests.

Keywords: *Reinforcement learning, Pricing Duopoly, Algorithmic Collusion*

I. Introduction

Humans have dictated market dynamics since the beginning of time. However, recent advances in computational power and in *AI-driven* algorithms, increase the likelihood of algorithms entering markets as autonomous decision-makers. This raises concerns for competition policy. Current competition law focuses on proof of explicit or implicit anti-competitive agreements, which is assumed to be a sufficient requirement to exclude tacit collusion, because firms find it difficult to coordinate without communication (OECD 2017). However, experimental findings suggest this might not hold for reinforcement learning pricing algorithms, which learn how to cooperate through trial-and-error (Calvano et al. 2019).

This study contributes to the growing body of literature on algorithmic collusion in multi agent reinforcement learning systems. In an early study Tesauro and Kephart (2000) already reason that agents who learn how to value future consequences of their actions, should be less inclined to engage in the price wars. Subsequent research confirmed this view and finds that reinforcement learning algorithms can reach collusive market outcomes in controlled environments. Reinforcement learning agents manage to tacitly collude on prices in a Bertrand competition setting (see for example Tesauro and Kephart (2002), Calvano et al. (2018), Klein (2019)), or on quantities in a Cournot competition setting (see for example Kimbrough et al. (2005), Waltman and Kaymak (2008)). Still, these controlled environments are far from realistic market settings. For example, previous work on pricing in multi-agent reinforcement learning focuses almost exclusively on settings where the competing reinforcement learning algorithms

* Supervisor: Dr. C. Argenton, Second reader: Prof. Dr. J. Boone (both Tilburg University). This paper is a shortened version of the thesis I wrote to complete the MSc. Economics program at Tilburg University.

are identical. Therefore, this study extends the existing literature by considering the dynamics of distinct reinforcement learning algorithms.

II. Experiment design

In this similuation-based experiment, two computational agents are paired in four different settings to study the differences in emergent market dynamics. The economic environment in each setting is identical. The agents are constructed by means of two distinct reinforcement learning algorithms: Q-learning and TB(λ) (see the technical appendix for a discussion of the algorithms). Any differences in market interaction among the four settings thus result from algorithmic heterogeneity. The two agents operate in a duopoly where each agent sells one product. The environment consists of n differentiated products where horizontal differentiation is captured by μ and perfect substitutes are obtained in the limit as $\mu \to 0$. The mean utility of consumers for a specific product is i given by d_i. Market demand is modelled using the binomial logit function depicted in Equation 1, where $d_0 = 0$ represents the part of demand that remains unfilled (Belleflamme and Peitz 2015).

$$q_{i,t} = \frac{e^{\frac{d_i - p_{i,t}}{\mu}}}{\sum_{j=1}^{n} e^{\frac{d_j - p_{j,t}}{\mu}} + e^{\frac{d_0}{\mu}}} \tag{1}$$

The action space of the agents is discretized and bounded by the one-shot Bertrand Nash-Equilibrium (p_B) and the monopoly (p_M) price. It is assumed that sellers act rationally and do not set a price above the monopoly or below the one-shot Bertrand Nash-Equilibrium level. All feasible actions to an agent are $m = 10$ equally spaced prices that lie on the interval $[p_B, p_M]$. The interaction that results in the system is depicted in Figure 1. At each time step t the agents take an action A_t based on the representation of the environment state, where this representation consists of the previous price of both the agent and its competitor $S_t = \{A_{i,t-1}, A_{j,t-1}\}$. At the beginning of $t + 1$ both agents receive a reward R_{t+1} - their profit for period t - and a new state representation S_{t+1}. Firms are assumed to have constant marginal costs and there is no possibility to enter or exit the market. In this specification marginal costs are irrelevant to the decision-making process and profits are given by $\pi_{i,t} = p_{i,t} q_{i,t}$.

To study the differences in behavior in homogeneous settings and heterogeneous settings, four different specifications are used. The baseline - or control - case of each agent is a homogeneous setting in which two identical agents are paired. Setting 1 is the Q-learning versus Q-learning version of the experiment, setting 2 is the TB(λ) versus TB(λ) setting. To the knowledge of the author, TB(λ) has not been used in the literature on dynamic pricing in multi-agent reinforcement learning and is theoretically more sophisticated than Q-Learning. In the remaining settings, one Q-learning agent is paired with one TB(λ) agent. First the agents go through a *training phase* in which they are able to learn pricing strategies through interaction with an opponent and the environment. Generally, agents reach stable behavior at the end of the training phase. After the training phase, the agents engage in the actual *experiment*. In setting 4 and 4b, a trained Q-learning agent from setting 1 and a trained TB(λ) agent from setting 2 are paired. In setting 4b these pre-trained agents engage in a second training phase that lasts

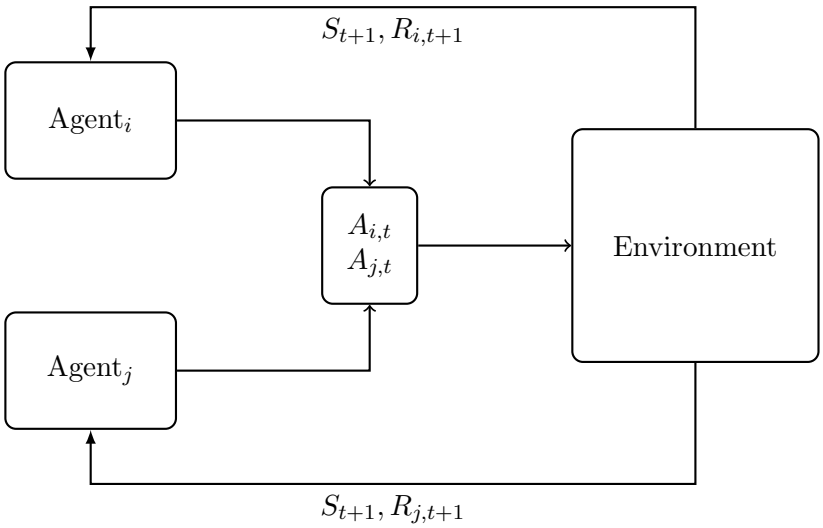

Figure 1. : Schematic overview of the agent-environment interaction.

for another 250.000 iterations, during which the behavior of the agents can stabilize. In setting 4 no additional time to stabilize behavior is provided. Consequently, their strategies are not necessarily stable at the beginning of the experiment. A schematic overview of the training phase and the experiment is provided in Figure 2.

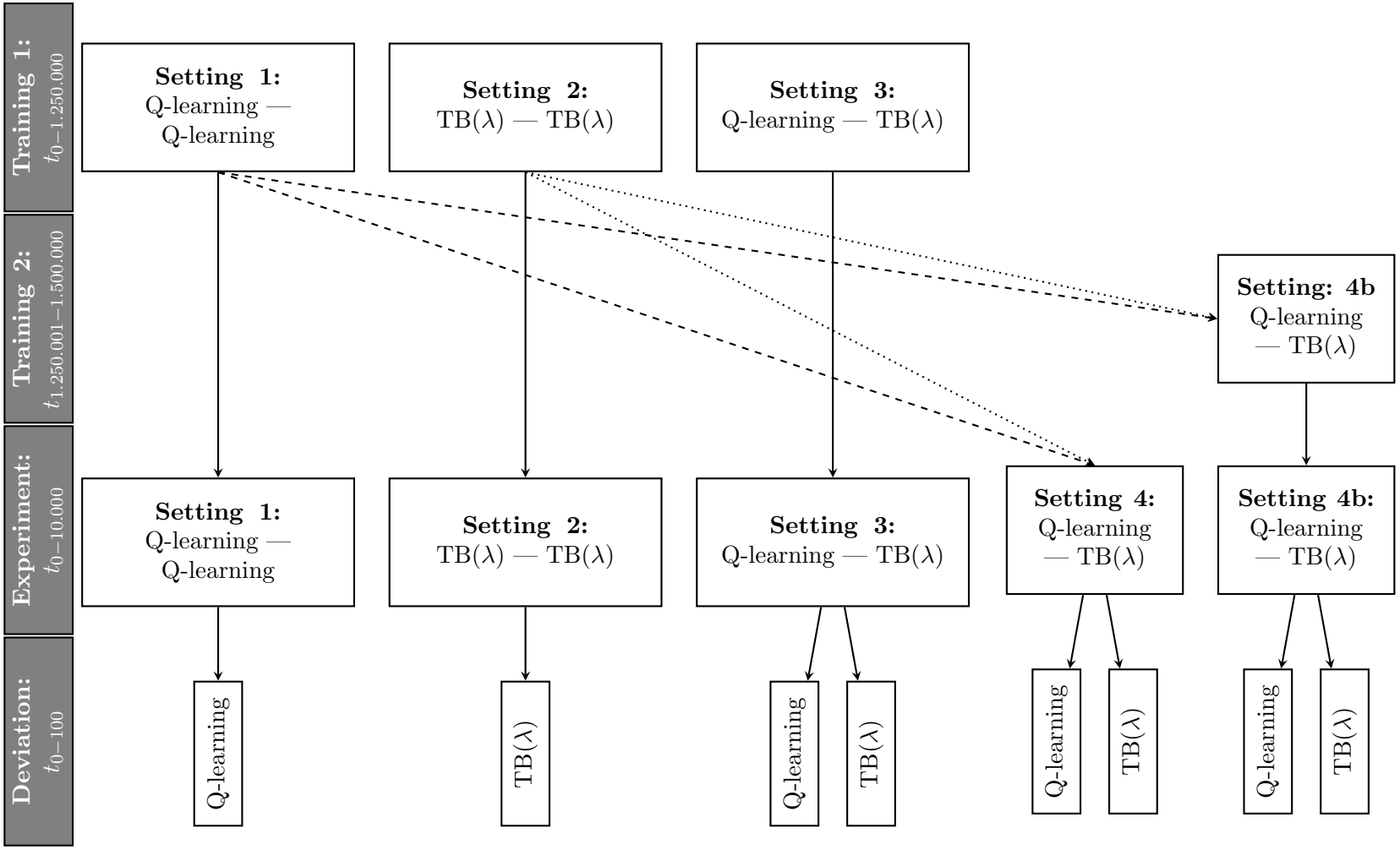

Figure 2. : Schematic overview of the set-up of the study. During the first round of the training phase, setting 1, 2 and 3 are trained until $t = 1.250.000$. During the second round of the training phase only setting 4b is trained. In this setting the trained representation of the first Q-learning agent and the trained representation of the second TB(λ) agent are paired and trained for another 250.000 iterations. After this, the experiment is initialized, and the agents play the pricing game from $t = 0$ until $t = 10.000$. In setting 1, 2, 3 and 4b the agents effectively continue where they stopped after training. With respect to setting 4, trained representations of a Q-learning and TB(λ) agent are paired, the start state is constructed from the combined history. During the second part of the experiment, time is reset to t_3 and for the stable sessions of the experiment one of the agents is forced to defect. In the settings 3, 4 & 4b both agents defect consecutively, and the experiment is reset between deviations.

III. Results

Algorithms are said to engage in tacit collusion when two conditions are satisfied; they set supra-competitive prices and manage to sustain these using incentive compatible deviation-punishment strategies. The degree of supra-competitive prices is measured using standardized average profit TC_π and standardized average price TC_p. Using the interval from Bertrand to Monopoly price level, a value of 0 coincides with a fully competitive situation and a value of 1 with joint-monopoly behavior. The distinction between the measures is subtle, but the combination provides insights in any algorithmic dominance in asymmetric outcomes; A lower level of TC_p and a higher level of TC_π means an agent manages to abuse its opponent. Table 1 summarizes the differences in market outcomes across the settings.

Only part of the sessions per experimental setting result in stable - one-cycle - behavior during the experiment. The fraction of sessions that do not display stable behavior, either do not converge or consist of recurring price cycles. This outcome is not uncommon in a MARL implementation for sequential social dilemmas, where part of the sessions are often found to converge to cyclic strategies (Sandholm and Crites 1996). The average degree of collusion of Q-learning and TB(λ) is comparable in the baseline settings (1 and 2). However, TB(λ) is far more likely to reach a stable and symmetric outcome. In line with theoretical expectations, TB(λ) also manages to converge to a stable outcome at a faster pace than the Q-learning agent. Hence the algorithm can be seen as a more experience-efficient learner. However, when the heterogeneous Q-learning vs TB(λ) settings are compared (setting 3, 4 and 4b), the Q-learning agent outperforms the TB(λ). The Q-learning agent systematically undercuts the price of its opponent and as a result its profit increases. The results of setting 3 indicate that the degree of tacit collusion remains high, despite the asymmetric division of profit. A similar situation can be observed in setting 4b, although the market outcome is on average a bit more symmetric and the degree of tacit collusion is somewhat lower. The performance of the algorithms in setting 4 and 4b show that the trained algorithms are only robust to encountering new opponents, when they are given enough opportunity to interact for coordination to re-establish.

These results show that tacit collusion can arise in all settings of the experiment, provided that the algorithms are given sufficient time to stabilize. At the same time, market outcomes are not necessarily symmetric. Consistently, the Q-learning agent learns to assert a dominant market position with respect to the TB(λ) agent. The results of the Wilcoxon signed rank test affirm this finding. They indicate that the differences in the scores on TC_p and TC_π between the two agents are statistically significant in all heterogeneous settings (except 4, likely because of the small sample size).

To test that agents have learned deviation-punishment strategies and supra-competitive market outcomes are not just an artifact from an accidental sub-optimal optimization, one of the agents is forced to deviate. The defecting agent plays a myopic best-response to the expected price. To assess the effectiveness of the punishment, the present value of the profit of the defecting agent after a deviation is compared to the counterfactual in which no deviation has taken place. The punishment is incentive-compatible when the present value of the counterfactual is larger and a deviation thus effectively reduces agent-welfare. Only the stable sessions are considered in this analysis.

Setting 1 is depicted in Figure 3, the present value of the loss is -0.09 and the corresponding

Table 1—: Tacit collusion across settings

Setting	**Stab.**	**Sym.**	TC_π		**WSR**	TC_p		**WSR**
			A1	A2		A1	A2	
S: 1			Q-learning	Q-learning		Q-learning	Q-learning	
All	0.615	0.235	0.7036	0.7197	5419.5	0.5024	0.498	5577.5
			(0.2374)	(0.2417)	(0.4678)	(0.1548)	(0.1521)	(0.7654)
Stable	$n = 123$	0.365854	0.7188	0.7505	1382.5	0.514	0.5032	1431.5
			(0.2427)	(0.2473)	(0.4294)	(0.1618)	(0.1599)	(0.5857)
S: 2			TB(λ)	TB(λ)		TB(λ)	TB(λ)	
All	0.965	0.645	0.7107	0.7116	1192.5	0.4861	0.4856	1196
			(0.18)	(0.1754)	(0.6226)	(0.1634)	(0.1668)	(0.6369)
Stable	$n = 193$	0.668394	0.7058	0.7069	965.5	0.4796	0.479	947.5
			(0.1787)	(0.1748)	(0.6161)	(0.1595)	(0.163)	(0.5336)
S: 3			Q-learning	TB(λ)		Q-learning	TB(λ)	
All	0.775	0.155	0.8745	0.5509	166.5***	0.4447	0.5571	162.5***
			(0.2105)	(0.1976)	(0.0)	(0.1738)	(0.1728)	(0.0)
Stable	$n = 155$	0.2	0.8865	0.539	0***	0.4344	0.5556	0***
			(0.2142)	(0.1929)	(0.0)	(0.1728)	(0.1753)	(0.0)
S: 4			Q-learning	TB(λ)		Q-learning	TB(λ)	
All	0.085	0.05	0.4632	0.2808	5702.5***	0.2159	0.2893	5594.5***
			(0.3609)	(0.3616)	(0.0)	(0.1876)	(0.1859)	(0.0)
Stable	$n = 17$	0.588235	0.6719	0.7429	12.5	0.4837	0.4575	7.5
			(0.2053)	(0.1717)	(0.4406)	(0.114)	(0.1314)	(0.2676)
S: 4b			Q-learning	TB(λ)		Q-learning	TB(λ)	
All	0.455	0.195	0.6973	0.5459	2902.5***	0.3958	0.4495	2743***
			(0.218)	(0.2011)	(0.0)	(0.1253)	(0.1336)	(0.0)
Stable	$n = 91$	0.417582	0.7614	0.5766	198.5***	0.4078	0.4737	189.5***
			(0.2234)	(0.2172)	(0.0)	(0.1359)	(0.1415)	(0.0)

The table depicts values for stability (Stab: number of sessions where both agents select one price) and symmetry (Sym: number of sessions in which both agents select the same prices). Mean metrics for tacit collusion are provided per agent, with the standard deviation in parenthesis. With respect to differences in agent behavior the two-sided non-parametric Wilcoxon signed rank test with H_0 : of no group differences is used. P-values are provided in parenthesis. *p-value ¡ 0.1, **p-value ¡ 0.05, ***p-value ¡ 0.001. A missing value denotes that there were too many differences of zero to implement the test.

reduction in welfare for the punishing agent is -0.18. Setting 2 is depicted in Figure 4, the present value of the loss for the defecting agent is -0.02 and the corresponding cost of punishment for the opponent is -0.11. Hence, both settings have punishment strategies that are incentive compatible. The welfare reduction to the TB(λ) agent as a result of the punishment is lower than for the Q-learning case, which implies the agent has learned more cost-effective punishment-deviation strategies.

When in setting 3 the Q-learning agent is forced to defect (Figure 5), the present value of its loss in welfare is -0.13. The corresponding cost of this punishment to the TB(λ) agent is -0.07. When alternatively the TB(λ) agent is forced to defect (Figure 6), a small gain in welfare

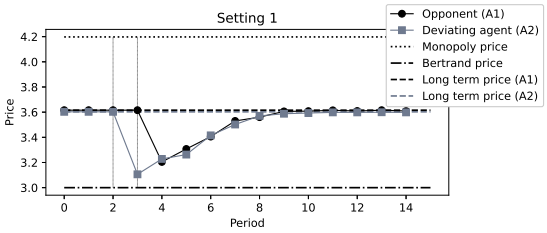

Figure 3. : Setting 1: Agent 2 deviates

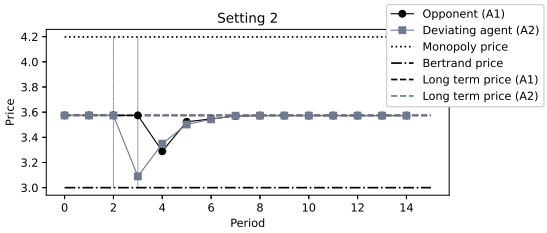

Figure 4. : Setting 2: Agent 2 deviates

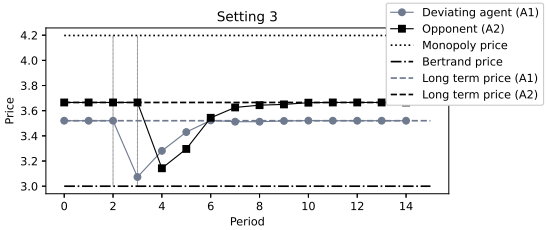

Figure 5. : Setting 3: Q-learning agent deviates

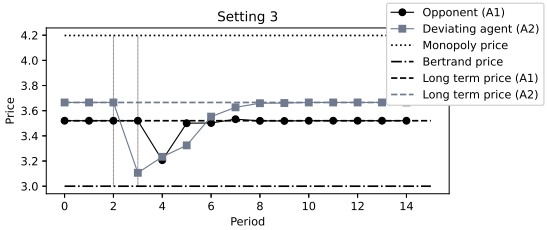

Figure 6. : Setting 3: TB(λ) agent deviates

is made consisting of 0.02. The cost of this gain to the Q-learning agent is substantial with a present value of -0.21. Interestingly, this reduction in profit is not enough to yield an incentive-compatible punishment strategy. Still, the algorithm adjusts its behavior back to collusion after the deviation. There are two possible explanations for this counter-intuitive result. First, it could be that the algorithm has not managed to learn optimal behavior. Second, the punishment was large enough during the training phase of the algorithms. The TB(λ) agent is equipped with a mechanism, called eligibility traces, to back-propagate credit for actions through time. This could result in a compounded effect of punishment on profit during the training phase compared to the Q-Learning agent.

The results of setting 1, 2 and 3 imply that when the agents are trained together, they learn to punish a defecting agent for a few iterations until the deviation is rendered unprofitable. This corroborates the findings by earlier research for the homogeneous Q-Learning case (see Klein (2019), Calvano et al. (2018)). However, setting 4 and 4b - in which the agents were not or only partly trained together - show a completely different picture. In Figure 7 to Figure 10 a deviation is not followed by a return to the pre-deviation price level. Instead, prices continue to oscillate at a level below the long-run price. In setting 4b post-deviation prices recuperate slightly, but do not bounce all the way back to the long-run price. Concluding, agents that are not trained together, may not be able to learn effective deviation-punishment strategies.

IV. Conclusion

This study conducts a pricing experiment to explore whether the presence of algorithmic heterogeneity changes the degree of tacit collusion in multi-agent reinforcement learning systems. Earlier work focused on experiments where a Q-learning agent was paired with one or more

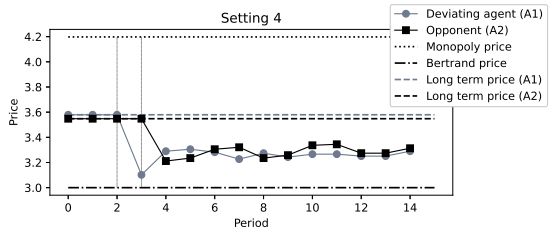

Figure 7. : Setting 4: Q-learning agent deviates

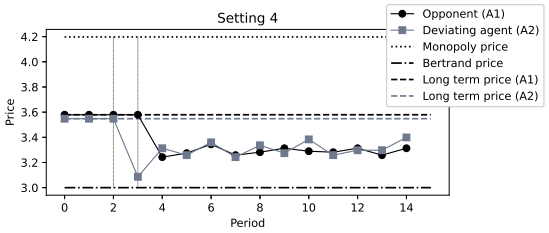

Figure 8. : Setting 4: TB(λ) agent deviates

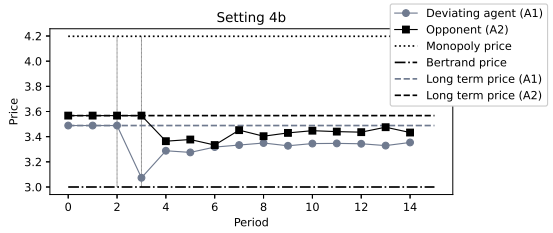

Figure 9. : Setting 4b: Q-learning agent deviates

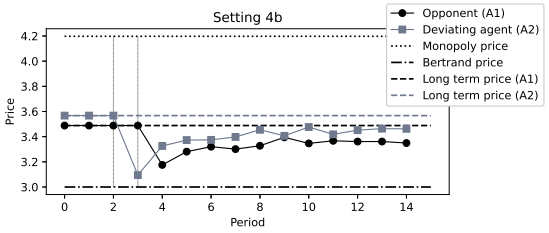

Figure 10. : Setting 4b: TB(λ) agent deviates

identical algorithms. The prevalent finding in these studies is that agents set supra-competitive prices and learn deviation-punishment strategies to sustain cooperation. The experiment conducted in this study extends this line of research by pairing a Q-learning agent with a different type of reinforcement learning algorithm called Tree-Backup(λ).

Both the Q-Learning and Tree-Backup(λ) algorithms learn to set supra-competitive prices when paired with an identical opponent and the degree of tacit collusion in these settings is comparable. The Tree-Backup(λ) algorithm is a more efficient learner. However, contrary to expectations the Tree-Backup(λ) does not have an edge over the Q-Learning agent. Instead, the Q-learning agent manages assume a dominant market position. This finding implies that inter-algorithm dynamics are not easily predictable and the resulting additional uncertainty with respect to performance makes adoption of these algorithms in e-retail markets less likely.

In the settings where the agents are trained together, agents learn to use punishment strategies to sustain cooperation. The deviating agent is undercut for multiple periods to render the deviation unprofitable. When agents are not trained together, post-deviation prices do not return to their long-run levels: Although supra-competitive pricing behavior is relatively robust to being paired to a new opponent if sufficient additional learning time is provided, the punishment strategies that are learned are not. These initial results imply that tacit collusion is not that easily sustainable outside of experimental settings. At the same time, these issues might be solved in case of algorithms that are better at generalizing. More research in this direction is needed before any definitive conclusions can be drawn.

Word count: 2166

References

[1] Belleflamme, Paul and Peitz, Martin. *Industrial organization: markets and strategies*. Cambridge University Press, 2015.

[2] Calvano, Emilio, Calzolari, Giacomo, Denicolò, Vincenzo, and Pastorello, Sergio. "Artificial intelligence, algorithmic pricing and collusion". In: (2018).

[3] Calvano, Emilio, Calzolari, Giacomo, Denicolò, Vincenzo, and Pastorello, Sergio. "Algorithmic pricing what implications for competition policy?" In: *Review of Industrial Organization* 55.1 (2019), pp. 155–171.

[4] De Asis, Kristopher, Hernandez-Garcia, J Fernando, Holland, G Zacharias, and Sutton, Richard S. "Multi-step reinforcement learning: A unifying algorithm". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[5] Kimbrough, Steven O, Lu, Ming, and Murphy, Frederic. "Learning and tacit collusion by artificial agents in Cournot duopoly games". In: *Formal modelling in electronic commerce*. Springer, 2005, pp. 477–492.

[6] Klein, Timo. "Autonomous algorithmic collusion: Q-learning under sequential pricing". In: *Amsterdam Law School Research Paper* 2018-15 (2019), pp. 2018–05.

[7] OECD. "Algorithms and Collusion: Competition Policy in the Digital Age". In: (2017).

[8] Precup, Doina. "Eligibility traces for off-policy policy evaluation". In: *Computer Science Department Faculty Publication Series* (2000), p. 80.

[9] Rana, Rupal and Oliveira, Fernando S. "Real-time dynamic pricing in a non-stationary environment using model-free reinforcement learning". In: *Omega* 47 (2014), pp. 116–126.

[10] Sandholm, Tuomas W and Crites, Robert H. "Multiagent reinforcement learning in the iterated prisoner's dilemma". In: *Biosystems* 37.1-2 (1996), pp. 147–166.

[11] Sutton, Richard S and Barto, Andrew G. *Reinforcement learning: An introduction*. MIT press, 2018.

[12] Tesauro and Kephart, Jeffrey O. "Foresight-based pricing algorithms in agent economies". In: *Decision Support Systems* 28.1-2 (2000), pp. 49–60.

[13] Tesauro and Kephart, Jeffrey O. "Pricing in agent economies using multi-agent Q-learning". In: *Autonomous Agents and Multi-Agent Systems* 5.3 (2002), pp. 289–304.

[14] Waltman, Ludo and Kaymak, Uzay. "Q-learning agents in a Cournot oligopoly model". In: *Journal of Economic Dynamics and Control* 32.10 (2008), pp. 3275–3293.

Technical background on the reinforcement learning algorithms

This short technical background note highlights the main differences between the two algorithms used in this paper. For a full introduction to reinforcement learning, see Sutton and Barto (2018).

Reinforcement learning is a branch of machine learning where algorithms learn recursively through interaction with an environment and/or other agents. At every time step t, a reinforcement learning agent receives a representation of the environment, called the state $S_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of all possible states. After observing the state S_t, the agent selects an action

A_t. The selection of an action is governed by a policy $\pi(a|s)$, which represents a mapping from states to possible actions. Based on the behavior of the agent and the state of the environment, the agent receives a reward $R_{t+1} \in \mathbb{R}$ and transitions to the next state $S_{t+1} \in \mathcal{S}$ with a certain state-transition probability $p(s'|s,a) = P(S_{t+1} = s'|S_t = s, A_t = a)$, for $a \in \mathcal{A}$ and $s, s' \in \mathcal{S}$ where s' denotes the state at time $t+1$ and s, a are respectively the state and action at time t. This specification boils down to the tuple (S, A, R, T) that represents the Markov Decision Process. The objective of the agent is to learn an optimal policy π^* that maximizes expected return G_t, where $\gamma \in [0, 1)$ denotes the discount rate.

$$G_t = R_{t+1} + \gamma R_{t+2} ... \gamma^{n-1} R_{t+n} \tag{A1}$$

The value of G_t is estimated recursively by an action-value function $q_\pi(s, a)$. First, an action a is selected according to the policy π in a given state s. The reward r and new state s' provide a feedback signal at the beginning of $t+1$. Based on this information, the agent updates its estimate of the expected value associated with the state-action pair that was used to decide during time t. Using this approach, the estimates improve a little after each iteration in the direction of the estimation error (called temporal-difference or TD-error).

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \tag{A2}$$

The policy used to select actions in this paper is the ϵ-greedy policy. Agents *explore* the environment with probability ϵ, or *exploit* previously learned knowledge with probability $1 - \epsilon$.

Q-learning

Q-learning is a popular off-policy algorithm, which means the information that is gained during the exploration phase (behavior policy) is used to estimate the values of a target policy (Sutton and Barto 2018). The target policy is the policy the agent attempts to learn: The deterministic greedy policy in case of ϵ-greedy. This implies that the Q-learning agent does not use the actual state to which it transitions to update the action-value function, but instead uses the estimate of the maximum Q-value for a given action. Consequently, each Q-value represents the discounted sum of future rewards given that action A_t is selected at time t and the optimal policy is followed afterwards (Sandholm and Crites 1996).

The Q-learning agent selects an action based on its estimated value for the state-action pair $Q(S_t, A_t)$ and updates the estimate at the beginning of $t+1$, when the reward is observed. This update follows A3, where α is the learning rate which determines the speed with which estimates are updated. After observing the reward R_t and the new state S_{t+1}, the previous estimate of the state-action pair is updated by multiplying the learning rate with the difference between the expected cumulative return for selecting the optimal action in the new state S_{t+1} and the value of the previous estimate.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{A3}$$

Where the TD-error δ_t thus corresponds to:

$$\delta_t = R_t + \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \tag{A4}$$

Pseudo code of the algorithm is provided below.

Pseudo code: Q-learning

Initialize $Q(s, a)$, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$ at myopic values
Randomly initialize S_0
Loop for each time step t:
    Choose A from S using the ϵ-greedy policy
    Take action A, observe R, S'
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$
    $S \leftarrow S'$
until $t = T$

Tree Backup(λ)

Tree Backup(λ) - referred to as TB(λ) throughout this study - is an algorithm introduced by Precup (2000) that can be used off-policy with eligibility traces. The TB(λ) algorithm is useful when the behavior policy is non-Markov, non-stationary or completely unknown (De Asis et al. 2018). The algorithm differs from Q-learning in two ways; It has a different update function and it uses eligibility traces.

The update function of Tree Backup(λ) is similar to Q-learning, but instead of using the optimal action to back-up the estimate of its action-value, it uses the expectation over all actions in the back-up (De Asis et al. 2018). In the off-policy specification of the algorithm, actions that are taken when exploring are disregarded in the calculation of the probabilities of the actions. Consequently, the target policy represents a stochastic optimal policy, where the policy becomes greedy in the limit when exploration tends to zero and the probability distribution used to select the actions in each state will move to the optimal action selection distribution. The update function without considering eligibility traces boils down to Equation A5.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{A5}$$

Where the TD-error δ_t corresponds to:

$$\delta_t = R_t + \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) - Q(S_t, A_t) \tag{A6}$$

Eligibility traces are designed to assign credit of actions to previous decisions. They keep an overview of the n-states that have contributed to an estimate and adjust them relative to their importance in generating the estimation error. Effectively this means past actions are penalized or rewarded based on the returns of current actions. A study by Rana and Oliveira (2014) compares Q-learning with a variant of Q-learning including eligibility traces in the estimation of demand in a dynamic pricing problem. They find that the adoption of eligibility traces increases performance. Each state-action pair that is visited receives an eligibility score that fades over time. Hence state-action pairs that are visited frequently, and that are visited more recently have a higher eligibility. Eligibility fades away by means of the trace decay parameter λ, using an approach called replacing traces (Sutton and Barto 2018). Following, Precup (2000), the eligibility traces $e_t(s, a)$ are updated at each time step as follows:

$$e_t(s, a) = e_{t-1}(s, a)\gamma\lambda\pi(s, a), \qquad \forall s, a \tag{A7}$$

$$e_t(s, a) = 1 \text{ iff } s = S_t, a = A_t \tag{A8}$$

Then using the off-policy TD-error in Equation A6, the action-value function can be updated according to:

$$Q(s, a) \leftarrow Q(s, a) + \alpha e_t(s, a)\delta_t, , \qquad \forall s, a \tag{A9}$$

Initialization

Both algorithms use a tabular representation of their value function. The Q-matrix is initialized using the true Q-value that results from fully myopic behavior (see for example: Tesauro and Kephart (2002), Waltman and Kaymak (2008)):

$$Q_0(s, a) = \frac{\pi_B}{(1 - \gamma)}, \qquad \forall a, s \tag{A10}$$

This can be interpreted as both agents starting in a fully competitive market, from which the agents learn to collude over time. The parameters of both algorithms are picked by means of a grid search for the values $\alpha \in \{0.2, 0.3, 0.4\}$, $\epsilon \in \{0.05, 0.10, 0.15\}$ and $\lambda \in \{0.3, 0.4, 0.5\}$. For the Q-learning agent the values $\alpha = 0.2$ and $\epsilon = 0.05$ are selected. For the TB(λ) agent the values $\alpha = 0.2$, $\epsilon = 0.05$ and $\lambda = 0.3$ are selected. Both agents use a discount factor of $\gamma = 0.95$, which is set arbitrarily. Both the learning rate α and the exploration parameter ϵ

Pseudo code: Tree Backup(λ)

Initialize $Q(s,a)$, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$ at myopic values
Randomly initialize S_0
Uniformly initialize $\pi(a|S)$ for all $s \in \mathcal{S}$
Initialize state count $\mathbf{n}$, for all $s \in \mathcal{S}$
Initialize action count $\mathbf{c}$, for all $a \in \mathcal{A}(s)$, $s \in \mathcal{S}$
Initialize eligibility traces $\mathbf{e}$ with 0
Loop for each time step t:
    Choose A from S using the ϵ-greedy policy
    Take action A, observe R, S'
    Update eligibility traces: $e_t(s,a) = e_{t-1}(s,a)\gamma\lambda\pi(s,a), \qquad \forall s,a$
    Replacing traces: $e_t(S,A) = 1$
    $\delta_t = R_t + \sum_a \pi(a|S_{t+1})Q(S_{t+1},a) - Q(S_t,A_t)$
    $Q(s,a) \leftarrow Q(s,a) + \alpha e_t(s,a)\delta_t, , \qquad \forall s,a$
    if π is greedy $(1-\epsilon)$:
        Update probabilities $\pi(a|S) \rightarrow -\frac{c}{n(n+1)}$, for $a \in \mathcal{A}(S)$
        Update probability of selected action $\pi(A|S) \rightarrow +\frac{1}{n+1}$
    $S \leftarrow S'$
until $t = T$

follow a linear decay schedule over time. Exploration lasts from $t_0 - t_{1.000.000}$, where the final 250.000 iterations are used to let agent behavior stabilize. Learning decays gradually over the full duration of the training phase $t_0 - t_{1.250.000}$.