Algorithmic Collusion Detection*

Matteo Courthoud†

September 16, 2021

Abstract

Recent studies on algorithmic pricing have shown that algorithms can learn sophisticated grim-trigger strategies with the intent of sustaining supracompetitive prices. This paper focuses on algorithmic collusion detection. One frequent suggestion is to look at the inputs of the strategies, for example on whether the algorithm conditions its own pricing strategy on past competitors' prices. I first show that this approach might not be sufficient since algorithms can learn reward-punishment schemes that are fully independent from the rival's actions. The mechanism that ensures stability of supra-competitive prices in this environment is self-punishment. In the second part of the paper, I explore a novel test of algorithmic collusion that builds on the intuition that a crucial ingredient for algorithmic collusion is synchronous learning. When one algorithm is unilaterally retrained, it learns new strategies that exploit collusive behavior. Since this change in strategies happens only when algorithms are colluding, retraining can be used as a test to detect algorithmic collusion. Lastly, I show how one can get the same insights on collusive behavior using only observational data from a single algorithm, making the implementation of the test feasible and low-cost.

Keywords: Artificial Intelligence, Collusion, Antitrust

JEL Classification: D21, D43, D83, L12, L13

*Preliminary, latest version here.

†University of Zürich, email: matteo.courthoud@econ.uzh.ch.

1 Introduction

Algorithms are slowly substituting human decision making in many business environments. While some applications such as playing games (Silver et al., 2016) or driving cars have attracted most of the attention of the press, algorithms are also more and more frequently used to make pricing decision. Since these algorithms need experience in the form of data to be trained, their applications mostly reside in high-frequency markets, such as gasoline markets (Assad et al., 2020), Amazon marketplace (Chen et al., 2016) or financial markets.

A recent strand of literature has highlighted the ability of these algorithms to learn complex dynamic collusive strategies, in the form of reward-punishment schemes. In particular, Calvano et al. (2020b) found that reinforcement learning algorithms are able to learn complex reward-punishment schemes with the intent of keeping supracompetitive prices. Their seminal paper focuses on reinforcement learning algorithms deployed in a controlled, simulated environment where algorithms compete in prices, facing a static demand function. Algorithms observe their own and their competitors' past prices and, based on the experience accumulated at that point in time, they set prices simultaneously. They authors show that the algorithms are able to learn collusive strategies autonomously, without any human intervention nudging them towards collusion.

Policymakers are currently discussing what can be done either to prevent these collusive behaviors or to detect and punish them. Ezrachi and Stucke (2017) and Ezrachi and Stucke (2019) suggest to create testing hubs for algorithms in order to classify them into potentially collusive and competitive. Another ex-ante solution, proposed by Harrington (2018), is to look at the algorithms' inputs and ban algorithms that are provided information that would allow them to collude, such as observational actions of their rivals. In contrast to this ex-ante solution, Calvano et al. (2020a) propose an interim approach: computer scientists should make algorithms more interpretable, so that humans could observe their behavior in real time and identify the rationales behind their decision making. Another potential approach is to intervene ex-post and sanction algorithms based on their observed behavior.

The first contribution of this paper is to show that, by solely inspecting the inputs of the algorithm, it is hard to draw conclusions on whether the algorithms are colluding. Even in a controlled setting, such as the one in Calvano et al. (2020b), it might not be obvious which inputs are necessary to define the collusive potential of an algorithm. While Harrington (2018) argues that in this simple setting it would be sufficient to forbid algorithms to observe competitors' previous actions to prevent collusion, I show that it is not enough. In particular, pricing algorithms can learn reward-punishment schemes with the sole intent of keeping supracompetitive prices using strategies that are fully independent from competitors' actions. In the same setting of Calvano et al. (2020b), I allow algorithms to base their strategies only on their *own* past prices. Even with this restriction on the algorithms inputs, algorithms are able learn to punish themselves for deviating from collusive prices and are hence able to obtain supracompetitive profits and sustain them in the long run. While being an extreme case, this exercise underlines the fact that even full independence of algorithms' strategies with respect to their

opponents' past actions does not prevent the emergence of tacit collusive schemes. Categorizing algorithms into potentially-collusive and potentially-competitive based on their inputs is not trivial, even in the most controlled environment.

Given that detecting collusion from the inspection of the algorithms' strategies might not be feasible, in the second part of this paper I propose a model-free method to detect algorithmic collusive behavior. My approach is based on the insight that if an algorithm is sufficiently sophisticated to learn reward-punishment schemes with the intent of keeping supra-competitive prices, it should also be able to learn to exploit such schemes. I show that keeping fixed the strategies of its competitors, one algorithm is able to unilaterally exploit the collusive strategy of its opponent, if it is not competitive. It is important to underline that the new strategy is not a simple one-shot price undercut but, since reinforcement learning allows algorithms to build history-dependent strategies, the deviation is more complex. Importantly for the possibility of this insight to be used for a test of collusive behavior, when the algorithms are behaving competitively, they are not able to learn more profitable strategies through retraining.

The main drawback of this method is that it is infeasible in practice: requiring firms to retrain their algorithms is too expensive. Moreover, the test would require other algorithms to not modify their behavior while the test is being performed, effectively requiring control over all algorithms active in the market. Because of this major drawback, the last part of this paper studies how to get the same insight using only the algorithms themselves and observational data. This test does not require any manipulation of market outcomes, but only requires observational data from a single firm. Indeed, this method can be applied unilaterally to each algorithm, without requiring the aggregation of data from multiple firms, nor the control of multiple algorithms simultaneously. The intuition behind the test is the following: the most recent sequences of payoffs and state transitions are representative of future interactions. Therefore, one can re-train one algorithm on its most recent observed data to mimic the current behavior of its competitors. In case the algorithms are colluding, I show that the retrained algorithm is able to learn to exploit the collusive behavior of its competitors. On the other hand, when algorithms are playing competitive strategies, re-training does not lead to significant changes in strategies. To the best of my knowledge, this constitutes the first attempt to build a model-free test for algorithmic collusion, using only observational data.

The paper is structured as follows. Section 2 contains a review of artificial intelligence algorithms in decision making. In particular, I cover recent advancements in computer science, with a focus on Q-learning algorithms and the specific formulation used in the simulations. In Section 3, I explore the extreme scenario in which a Q-learning pricing algorithms do not observe their competitor's price but still manage to adopt a reward-punishment schemes with the intent to keep supra-competitive prices. This exercise serves as a warning for policies aimed at defining collusive algorithms solely based on their inputs. Section 4 focuses on collusion detection. I first show how unilaterally retraining one algorithm leads to profitable deviations in case the algorithms were colluding. Then, I show how one can obtain the same insight using only observational data. Section 5 concludes.

2 Q-Learning

Reinforcement learning algorithms are a class of learning algorithms that try to solve optimization problems where rewards are delayed over multiple time periods. Moreover, these rewards depend on the sequence of actions that the algorithm has to take over different time periods. These two characteristics make reinforcement learning fundamentally different from standard supervised learning problems. The algorithm objective is not simply to learn a reward function but also to take an optimal sequence of actions. The most important feature of reinforcement learning is indeed the dual role of the algorithm: prediction and optimization.

Q-learning is a popular reinforcement learning algorithm and constitutes the baseline model for the most successful advancements in Artificial Intelligence in the last decade (Igami, 2020). The most popular algorithms such as Bonanza or AlphaGo are based on the same principle, while using deep neural networks to provide a more flexible functional approximation of the policy function (Sutton and Barto, 2018).

One advantage of Q-learning algorithms is their interpretability, especially for what concerns the mapping from the algorithm to the policy and value functions. The policy function has a matrix representation that can be directly observed and interpreted. This makes it possible to understand not only the logic behind any decision of the algorithm at any point in time, but also to know what the algorithm would have done in any counterfactual scenario.

In this section, I first explore the general formulation of Q-learning algorithms. Since these algorithms have been developed to work in single-agent environments, I comment on their use in repeated games. Lastly, I analyze the baseline algorithm used in the simulations throughout the paper.

2.1 Single Agent Learning

Reinforcement learning algorithms try to solve complex dynamic optimization problems by adopting a model-free approach. This means that the algorithm is not provided any structure regarding the relationship between the state it observes, its own actions, and the payoffs it receives. The algorithm only learns through experience, associating states and actions with the payoffs they generate. Actions that bring higher payoffs in a state are preferred to actions that bring lower payoffs. Since the state can include past states or actions, reinforcement learning algorithms can learn complex dynamic strategies. The more complex the state and action space of the learning algorithm, the more complex the strategies it can learn.

The objective function of the learning algorithm is the expected discounted value of future payoffs

$$\mathbb{E}\left[\sum_{t=0}^{\infty}\delta^t\pi_t\right]. \tag{1}$$

In many domains faced by computer scientists, payoffs have to be hard-coded together with

the algorithm. For example, with self-driving cars, one has to establish what is the payoff of an accident, or the payoff of arriving late. Clearly, the decision of these payoffs directly affects the behavior of the algorithm. However, in some cases, the payoffs are directly provided by the environment. For example, in the setting analyzed in this paper, the algorithm sets prices and observes the profits coming from the sales of an item.

The main trade-off faced by reinforcement learning algorithms is the so-called *exploration-exploitation* trade-off. Since the algorithm is not provided with any model of the world, it only learns through experience. In order to learn different policies, the algorithm explores the action space by taking sub-optimal actions. However, since the objective of the algorithm is to maximize the expected discounted sum of future payoffs, at a certain point, the algorithm needs to shift from exploration to exploitation, i.e., it needs to start taking optimal actions, given the experience accumulated so far.

In each period, the algorithm observes the current state of the world, $\boldsymbol{s}$ and takes an action a with the intent to maximize its objective function. I refer to the total discounted stream of future payoffs under optimal actions as the value function. I can express the value function of algorithm i in state s recursively as

$$V_i(\boldsymbol{s}) = \max_{a_i \in \mathcal{A}} \left\{ \pi_i(\boldsymbol{s}, \boldsymbol{a}) + \delta \mathbb{E}_{\boldsymbol{s}'}[V_i(\boldsymbol{s}')|\boldsymbol{s}, \boldsymbol{a}] \right\}. \tag{2}$$

Q-learning algorithms are based on a different representation of the value function, the action-specific value function, which is called the Q function. I can write the action-specific value function of algorithm i in state $\boldsymbol{s}$ when it takes action a_i as

$$Q_i(\boldsymbol{s}, a_i) = \pi(\boldsymbol{s}, \boldsymbol{a}) + \delta \mathbb{E}_{\boldsymbol{s}'} \left[\max_{a_i' \in \mathcal{A}} Q_i(\boldsymbol{s}', a_i') \Big| \boldsymbol{s}, \boldsymbol{a} \right]. \tag{3}$$

When the state space $\mathcal{S}$ and the action space $\mathcal{A}$ are finite, we can express the Q function as a $|\mathcal{S}| \times |\mathcal{A}|$ matrix. Many of the advancements in reinforcement learning involve a functional representation of the Q function that can encompass more complex state or action spaces. The most successful function approximations involve deep neural networks.

The objective of the algorithm is to learn the Q function. Once the algorithm has learned the Q function, in each period it will take the optimal action $a_i^* = \arg\max_{av \in \mathcal{A}} Q(\boldsymbol{s}, a_i)$. Learning and taking the optimal actions are the two main tasks of the algorithm, and the choice between the two behaviors constitutes the main trade-off of reinforcement learning: *exploration* versus *exploitation*.

Exploitation. Since the final objective of the algorithm is to maximize the expected discounted sum of payoffs, in the exploitation phase, the algorithm picks the best available action, given the experience accumulated so far. After a sufficient amount of exploration, in a stationary environment, the algorithm expected discounted sum of future payoffs is guaranteed to converge to a local optimum of the value function (Beggs, 2005). Exploration is needed in order to discover other better locally optimal policies.

Exploration. During the exploration phase, the objective of the algorithm is to explore

the state-action space in order to discover new policies. Since exploration involves testing sub-optimal actions, there is a trade-off between exploration and exploitation. More exploration implies lower short-term profits but can lead to the discovery of policies than bring higher long-term profits. Moreover, when reinforcement learning algorithms are deployed in a dynamic environment, exploration gives the algorithm the flexibility to adapt to changes in the environment.

There exists may different ways in which one algorithm can explore the state-action space and the simplest one is the ε-greedy model. In each period, the algorithm decides to explore with probability ε and to exploit with probability (ε). In the exploration phase, the algorithm chooses one action uniformly at random.

Optimality Results. In a Markov single agent environment, under mild conditions, it has been proven that a reinforcement learning algorithm learns locally optimal strategies, given that the exploration rate ε converges to zero as time goes to infinity (Sutton and Barto, 2018).

Learning Speed. Since learning is a noisy process, the Q matrix is only partially updated. In particular, given an action a_i^*, irrespectively of whether the action comes from exploration or exploitation, the update policy is

$$Q_i(\boldsymbol{s}, a_i^*) = \alpha Q_i^{OLD}(\boldsymbol{s}, a_i^*) + (1-\alpha) Q_i^{NEW}(\boldsymbol{s}, a_i^*) \tag{4}$$

where we refer to α as the *learning rate*. A higher α implies a faster but noisier learning. The policy function takes less time to converge but it is less likely to adopt better strategies.

2.2 Repeated Games

In the setting of this paper, multiple Q-learning algorithms play a repeated game in which they set per-period prices with the objective to maximize the expected discounted sum of profits. Most of the reinforcement learning literature in computer science focuses stationary environments. One common example is video games, where algorithms receive the video feed as an input and have to pick the optimal actions in order to perform best in the game. Another common area of research in reinforcement learning is robotics, from logistics to self-driving cars. All these examples involve mostly stationary environments.

The behavior of reinforcement learning algorithms competing with each other in a repeated game is still under research and there exist no general result concerning their behavior. In particular, there is no result on whether algorithmic behavior will converge on collaborative behavior, depending on the context.

3 Blind Learning

Differently from humans, algorithms have strategies that are hard-coded and hence directly observable. I cannot observe whether humans decide to increase prices because of higher de-

mand, higher input prices, or an agreement with competitors. This is indeed the reason why collusion per-se is not prohibited by law which, on the other hand, targets communication with the intent to collude. However, we can observe the decision-making process of the algorithm. Therefore, one might wonder whether it is possible to detect collusion by the inspection of the algorithms' strategies.

The main problem of detecting collusion from the inspection of algorithms' strategies comes from the fact that these strategies might be extremely complex and hard to understand for a human. In practice, most reinforcement learning algorithms do not rely on a matrix representation of the Q function but approximate it through deep neural networks which are known to be extremely difficult to interpret.

One approach that has been proposed by Harrington (2018) is to look at the inputs of the algorithm strategies. In particular, commenting on Calvano et al. (2020b), he suggests that one possible metric to flag collusive algorithms is to look at whether the algorithm prices are conditional on rivals' past prices.

> *"In this simple setting, a pricing algorithm would be prohibited if it conditioned price on a rival firm's past prices. AAs* [Artificial Agents] *would be allowed to set any price, low or high, but just not use pricing algorithms that could reward or punish a rival firm based on that firm's past prices."*

And he adds

> *"I am not suggesting that, in practice, a pricing algorithm should be prohibited if it conditions on rival firms' past prices. However, within the confines of this simple setting, that would be the proper definition of the set of prohibited pricing algorithms."*

In this section, I am going to show that looking at the inputs of the algorithm strategies might not be sufficient to detect algorithmic collusion. In particular, I show that an algorithm whose strategy is based only on its own past action can still learn a reward-punishment scheme with the sole intent of keeping supra-competitive prices.

3.1 Model

I adopt the baseline model from Calvano et al. (2020b). First of all, this choice allows a direct comparison with their simulation results. Moreover, their setting is particularly simple and easy to interpret. Lastly, since many competition policy and law papers are based on their results, I can directly speak to that literature using the same exact model and parametrization.

Time is discrete and the horizon is infinite. At each point in time, n firms are active and compete in prices with differentiated products. Differently from Calvano et al. (2020b), the state of the game for each firm is its own history of pricing decisions up to k times periods in the past. I discretize the possible actions on a grid of dimension m: $\{p_1, ..., p_m\}$. Therefore, the subjective state of firm n is represented by a vector $\boldsymbol{s}_{i,t} = \{p_{i,t-1}, ..., p_{i,t-k}\} \in S = \{p_1, ..., p_m\}^k$,

where $s_{i,t}$ represents the subjective state of firm i at time t. I will refer to $\boldsymbol{s}$ as a subjective state and S as the subjective state space. Firms maximize their total future discounted profits. The discount factor is $\delta \in [0, 1)$.

Demand. There is a continuum of consumers of unit mass. Consumer j's utility from buying one unit of product i is given by

$$u_{j,i} = v_j - \mu p_i + \varepsilon_j \tag{5}$$

where v_j is the value of the product i for consumer j, p_i is the price of product i, μ is the price elasticity and ε_j is the idiosyncratic shock preference of consumer j for product i. The random shocks ε_j are assumed to be independent and type 1 extreme value distributed so that the resulting demand function has the logit form. For example, the demand of product i is:

$$q_i(\boldsymbol{p}) = \frac{e^{-\mu p_i}}{e^{-\mu p_i} + \sum_{-i} e^{-\mu p_{-i}}}. \tag{6}$$

The static game has a unique Nash Equilibrium which we refer to as the competitive outcome.

Exploration/Exploitation. I use a ε-greedy exploration method as in Calvano et al. (2020b). In each period, each algorithm has a probability ε_t of exploring and a probability $1 - \varepsilon_t$ of exploiting. I refer to ε as the exploration rate. The value of ε_t in period t is given by

$$\varepsilon_t = 1 - e^{-\beta t} \tag{7}$$

where β is the convergence parameter that governs how quickly the algorithms shift from exploration to exploitation. As shown in Calvano et al. (2020b), the probability of collusion is generally increasing in β. The more the algorithms are allowed to explore, the more likely it is that they "stumble upon" a reward-punishment scheme. Once they discover these schemes, they are likely to adopt them since they are more profitable than playing Nash Equilibrium in the long run.

Q Matrix. Each algorithm policy function is defined by a Q-matrix. The Q-matrix has dimension $m^k \times m$ where the first dimension indicates the state space, i.e., the sequence of own past actions, and the second dimension indicates the current action. I initialize the Q matrix to the sum of discounted value of future profits, given action a_i, and averaging over the possible actions of the opponents. Therefore, the initial values do not depend on the state $\boldsymbol{s}$, but only on the action a:

$$Q_i^0(\boldsymbol{s}, a_i) = \frac{1}{|\mathcal{A}|} \sum_{\boldsymbol{a}_{-i} \in \mathcal{A}^{n-1}} \frac{\pi_i(a_i, \boldsymbol{a}_{-i})}{1 - \delta} \tag{8}$$

Policy Update. Irrespectively of whether an algorithm is exploring or exploiting, the Q matrix is updated by averaging it's new value with the previous one, according to a parameter α, the learning rate. In particular, the updating formula is the following:

$$Q_i(\boldsymbol{s}, a_i^*) = \alpha Q_i(\boldsymbol{s}, a_i^*) + (1 - \alpha)\Big[\pi(\boldsymbol{s}, a_i^*) + \delta \max_{a_i'} Q_i(\boldsymbol{s}', a_i')\Big]. \tag{9}$$

The new value of Q is state $\boldsymbol{s}$ for action a_i^* is an average of the old value, $Q_i(\boldsymbol{s}, a_i^*)$, and the new one. The new value is given by the static payoff of action a_i^* in state $\boldsymbol{s}$, plus the discounted value of the best action the next state, $\boldsymbol{s}'$. There are different ways to update the continuation value of the Q-function (Sutton and Barto, 2018). In this setting, I select the optimistic updating given by the *max* operator to be consistent with the choice of Calvano et al. (2020b). Other possible solutions are weighted averages, where the softmax is a popular weighting function.

Algorithm. I summarize the full algorithm in Figure 1

Algorithm 1: Q-learning

initialize $Q_i^0(\boldsymbol{s}, a_i)\ \forall i = 1...n, \boldsymbol{s} \in \mathcal{S}, a_i \in \mathcal{A}$;
initialize $\boldsymbol{s}^0$;
while *convergence condition not met* **do**
  for $i = 1...n$ **do**
    exploration$_i = \mathbb{I}\left(r_i < e^{-\beta t}\right)$ where $r_i \sim U(0,1)$;
    if *exploration$_i$* **then**
      $a_i^* = a \in A$ chosen uniformly at random ;
    else
      $a_i^* = \arg\max_{a_i} Q_i(\boldsymbol{s}, a_i)$;
    end
  end
  observe $\boldsymbol{\pi}$ given $(\boldsymbol{s}, a^*)$;
  observe $\boldsymbol{s}'$ given $(\boldsymbol{s}, a^*)$;
  $Q_i(\boldsymbol{s}, a_i^*) =$
   $\alpha Q_i(\boldsymbol{s}, a_i^*) + (1-\alpha)\Big[\pi(\boldsymbol{s}, a^*) + \delta \max_{a_i'} Q_i(\boldsymbol{s}', a_i')\Big]\ \ \forall i$;
  $\boldsymbol{s} = \boldsymbol{s}'$;
end

Parametrization. I summarize the parameters of the baseline model in Table 1. The parametrization closely follows Calvano et al. (2020b) so that the simulation results are directly comparable with theirs.

Parameter Description	Parameter	Value
Learning rate	α	0.15
Exploration rate	β	$4 \cdot 10^{-6}$
Discount factor	δ	0.95
Marginal cost	c	1
Number of past observed states	k	1
Dimension of the action grid	m	5
Number of firms	n	2
Convergence parameter	T	$U[0,1]$
Price elasticity	μ	0.05

Table 1: Model parametrization

3.2 Convergence

There is no guarantee of convergence for the learning algorithm. Algorithms react to each others' policies and therefore it is possible that they get stuck in a loop. Moreover, as long as there is a non-zero probability of exploration, there is always a chance that one algorithm suddenly adopts a totally different policy.

I use the same convergence criterion of Calvano et al. (2020b): convergence in actions. The algorithm stops if for T periods the index of the highest value of the Q matrix in each state has not changed i.e. $\arg\max_{a_i} Q_{i,t}(\boldsymbol{s}_t, a_i) = \arg\max_{a_i} Q_{i,t+\tau}(\boldsymbol{s}_{t+\tau}, a_i)\ \forall i, \boldsymbol{s}, \tau = 1...T$. In practice, I choose a value of $T = 10^5$.

The advantage of this convergence criterion is that it does not require the algorithms to adopt a full exploitative strategy in order to converge. Algorithms' actions can stabilize even if the algorithms regularly take random actions, but with low probability. In fact, as long as the learning-rate α is lower than 1, firms only partially update their Q function. In practice, we observe convergence around $500,000$ to $1,500,000$ iterations, i.e. when the exploration probability ε is around e^{-5} to e^{-10}.

It is important to remark that one can achieve convergence in a shorter amount of periods. What is crucial to achieve convergence is a sufficiently little exploration rate. However, with a lower exploration rate, the algorithms are less likely to discover reward-punishment collusive strategies.

3.3 Results

In my first experiment, I take the same setting of Calvano et al. (2020b) and I change the state space of the algorithms, so that they do not observe each others' past prices and are therefore not able to condition their actions on their rival's actions. In Figure 1, I plot the distribution of equilibrium prices and profits over 100 simulations. As we can see, the distribution is closer

to monopoly prices and profits than to the Nash Equilibrium values.

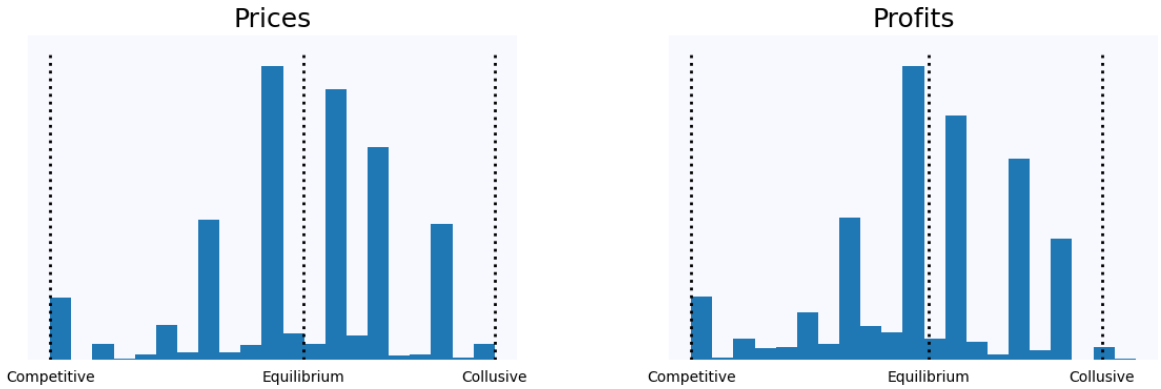

Figure 1: Distribution of equilibrium prices and profits of Algorithm 1 in the model with blind algorithms, with higher competition than Calvano et al. (2020b), $mu = 0.05$, over 100 simulations. The vertical lines indicate the static collusive values, the static Nash Equilibrium values and average equilibrium values.

These supra-competitive profits are achieved thanks to a reward-punishment scheme. Algorithms set higher prices until they observe a deviation. Once they observe a deviation, a punishment scheme starts and they earn lower profits for a couple of periods. Afterwards, they go back to the supracompetitive prices.

In order to see the reward-punishment scheme, I take the equilibrium Q function and manually manipulate the pricing action of one firm in one period and observe the reaction of both firms in the following periods. In particular, I make Algorithm 1 take the static best reply to Algorithm 2's action in the previous period. Then, I let the two algorithms react to this unilateral deviation in the following periods. In Figure 2, I report the average sequence of prices of the two algorithms, over 100 simulations.

Figure 2: Equilibrium actions of Algorithm 1 and 2 in the model with blind algorithms, before and after a unilateral deviation of Algorithm 1. Algorithm 1 is manually set to best-reply to the equilibrium policy of Algorithm 2, in period 10 (*Deviation*). From period 11 onwards (*Punishment*), both algorithms act according to their equilibrium policies. The horizontal lines indicate the static collusive prices, static Nash Equilibrium prices and average equilibrium prices. Each time series has been standardized with respect to its average equilibrium values. The non-standardized distribution of values is reported on the right. The vertical bars represent interquartile ranges while the vertical lines represent minimum and maximum values, over 100 simulations.

As we can see, when Algorithm 1 deviates from the stable collusive play, it sets a lower price. Since both algorithms observe only their own actions, Algorithm 2 does not deviate and keeps its collusive price. However, Algorithm 1 reacts to its own deviation and, in the following periods, it sets an even lower price before getting back to the stable collusive play.

To verify whether this was a reward-punishment scheme, we would need to observe that the profits of Algorithm 1 have increased in the deviation period, while they have decreased in the subsequent punishment periods. In Figure 3, I plot the profits of the two algorithms over 100 simulations.

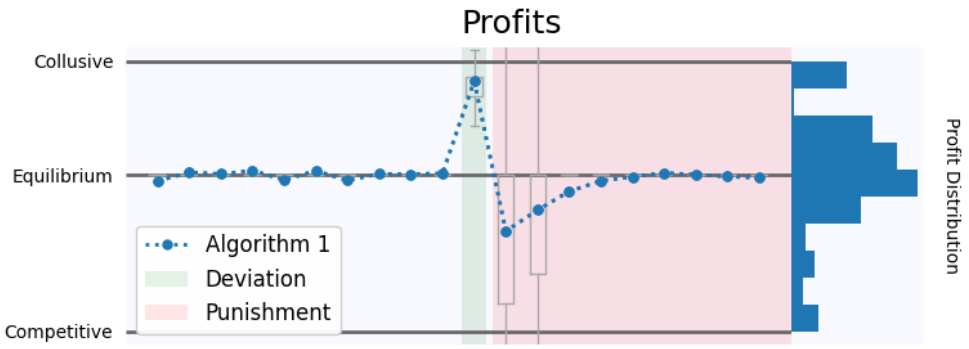

Figure 3: Equilibrium profits of Algorithm 1 in the model with blind algorithms, before and after a unilateral deviation of Algorithm 1. Algorithm 1 is manually set to best-reply to the equilibrium policy of Algorithm 2, in period 10 (*Deviation*). From period 11 onwards (*Punishment*), both algorithms act according to their equilibrium policies. The horizontal lines indicate the static collusive profits, static Nash Equilibrium profits and average equilibrium profits. Each time series has been standardized with respect to its average equilibrium values. The non-standardized distribution of values is reported on the right. The vertical bars represent interquartile ranges while the vertical lines represent minimum and maximum values, over 100 simulations.

Figure 3 confirms our hypothesis of collusive play: the deviating algorithm achieves higher static profits in the deviation period, but it is punished in the following periods.

3.4 Discussion

In the previous section, we have seen how firms can learn to play reward-punishment schemes with the intent to keep supra-competitive prices even when their strategy is independent from the opponent action. Whether this behavior is collusive, it is subject to debate. Harrington (2018) tackles the issue of defining collusion in the presence of algorithmic pricing and lays down the following definition.

> *"**Definition**: Collusion is when firms use strategies that embody a reward–punishment scheme which rewards a firm for abiding by the supracompetitive outcome and punishes it for departing from it."*

According to this definition, the behavior we observed in the previous section, would be defined as collusive. What is puzzling is the fact that the strategy of the firm is independent of the opponent strategy and the punishment comes from the algorithm itself and not from its competitor. I conclude this section by examining how the firm learns these strategies and why they are stable.

First of all, how do the algorithms learn these reward-punishment schemes? As we have already seen in Section 2, reinforcement learning algorithms can potentially learn any strategy that is allowed by their state-action space. This means that their limits reside on what they can observe (the state space) and what they can do (the action space). In this case, the algorithms observe only their own past action, their own price in the previous period, and use this information to set the current price. Therefore, they can learn any strategy in which the

current price depends on the last price. The more they explore the state-action space, the more likely they are to learn strategies that bring higher payoffs.

Second, why are these strategies stable and why firms do not learn to undercut their rivals? The fact that algorithms are bounded by their state-action space is both a constraint and an opportunity. In fact, in the exploration phase, the algorithm can potentially learn any strategy compatible with its state-action space. However, when the algorithm shifts towards exploitation, the probability of exploring for multiple periods, decreases quickly. This means that the algorithm tests its current strategy under against and shorter deviations, in terms of how many sequential periods are explored. At a certain point, the algorithm will test its own current strategy only against one-shot deviations.

Let us explore in detail the case in which Algorithm 1 explores the possibility of best-replying to the static price of its rival. This happens with probability ε. I assume we are in "exploitation-most" mode so that ε is small and hence $\varepsilon^2 \approx 0$.

1. Algorithm 1 best replies to the supracompetitive price of Algorithm 2 setting $p_1^{BR}(p_2^C)$
2. It gets a higher static payoff

$$\pi_1\left(p_1^{BR}\left(p_2^C\right), p_2^C\right) > \pi_1\left(p_1^C, p_2^C\right) \tag{10}$$

   where p_1^C and p_2^C are the collusive prices of the two algorithms and $p_1^{BR}\left(p_2^C\right)$ is the best reply of Algorithm 1 to the collusive price of Algorithm 2.

3. Algorithm 1 updates its Q function according to Equation 9:

$$Q_1\left(\boldsymbol{p}^C, p_1^{BR}\left(p_2^C\right)\right) = \alpha Q_1\left(\boldsymbol{p}^C, p_1^{BR}\left(p_2^C\right)\right) + \tag{11}$$

$$(1-\alpha)\left[\pi_1\left(p_1^{BR}\left(p_2^C\right), p_2^C\right) + \delta \max_{p_1'} Q_1\left(p_1^C, p_1^{BR}\left(p_2^C\right), p_1'\right)\right] \tag{12}$$

   where $\boldsymbol{p}^C$ is the vector of collusive prices.

4. As we can see, the update depends of two terms: the static profits $\pi_1\left(p_1^{BR}\left(p_2^C\right), p_2^C\right)$ and the future value $\delta \max_{p_1'} Q_1\left(p_1^C, p_1^{BR}\left(p_2^C\right), p_1'\right)$. Even if static profits are bigger than current profits, the future value is not. Why? Because of the punishment scheme. Algorithm 1 will choose $p_1' = p_1^P$ so that it will punish itself and, as a consequence, it will not see $p_1^{BR}\left(p_2^C\right)$ as a better strategy.
5. This would be different if Algorithm 1, in the next period would explore again and pick $p_1^{BR}\left(p_2^C\right)$ instead of p_1^P. However, as we have said at the beginning, this is extremely unlikely since we are in "exploitation-most" mode so that $\varepsilon^2 \approx 0$.

Third, how do algorithms converge to these strategies? My claim is that the crucial ingredient so that algorithms simultaneously converge to collusive strategies is synchronous

learning. If the algorithms were learning asyncronously[1], i.e., one algorithm is in "exploitation-most" mode while the other is in "exploration-most" mode, we would not obtain the same result. The algorithm in "exploration-most" mode would learn a new strategy to exploit the collusive algorithm. I test this hypothesis in the next section.

4 Detecting Algorithmic Collusion

In the previous section, we have seen that detecting algorithmic collusion from the inspection of the algorithm's inputs might not be feasible. Even in a very simple setting, where inspection of the algorithm's inputs seems sufficient to determine whether they can learn reward-punishment schemes with the intent to keep supracompetitive prices, simple rules might not be sufficient. Algorithms learn to collude even when they cannot base they strategy on past rival's actions.

One solution could be to detect collusion from observational data. Economists know signs that could hint at collusive behavior such as coordinated prices. However, these methods rely heavily on the underlying models of market interactions. In fact, one has to be able to distinguish collusion from a wide variety of confounding factors such as demand, aggregate shocks, input prices or simply noise.

In this section, I study a model-free method to detect algorithmic collusion. My method is based on the observation that algorithms are not trained to learn Subgame Perfect Equilibrium strategies. Therefore, there exists the possibility that equilibrium strategies can be exploited. This should not happen if the algorithms were playing competitively.

In the rest of this Section, I will show that the strategies that algorithms learn when colluding are exploitable by other qlearning algorithms. In practice, I show that re-training one of the two collusive algorithms against the other (holding the collusive strategy of the second fixed), leads to higher profits for the first one. Therefore, collusive strategies are not sub-game perfect and this insight can be used as a test for collusion. I show this result both in the original setting of Calvano et al. (2020b) and in the setting of Section 3. I also show that if instead the algorithms were playing the static competitive prices, their strategies cannot be exploited.

4.1 Blind Collusive Algorithm Retraining

If q-learning algorithms are sophisticated enough to learn complex grim-trigger strategies, they should also be able to learn to exploit another algorithm that is playing a collusive strategy, if this strategy is not sub-game perfect. In this Section, I explore this possibility. In particular, I explore what happens when one of the two algorithms is unilaterally re-trained.

First, I train the two algorithms as explained in Section 3: both algorithms only condition their current price on their *own* past price. Once convergence is achieved, I manually restart

[1]Note that this definition of "synchronous learning" differs from the one of Asker et al. (2021), in which algorithms learn payoffs also for actions they have not taken, thanks to economic modeling.

the timing t for one of the two algorithms. This means that, according to Equation 7, the probability of exploration is reset to 1 and it will gradually shift towards 0 as t increases. The algorithm will move from an "exploitation mode" mode to an "exploration most" mode. The Q function of the other algorithm is kept fixed over time, at its pre-retraining equilibrium value. I use the same criterion described in Section 3.2 to determine convergence.

In Figure 4, I plot the algorithm's profits before and after re-training.

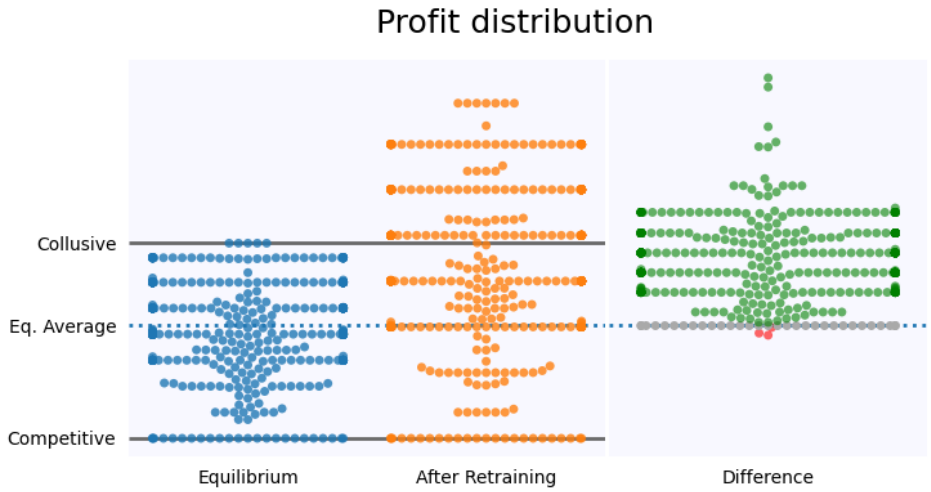

Figure 4: Equilibrium profits of Algorithm 1 in the model with blind algorithms, before and after retraining of Algorithm 1. On the right-hand side, the distribution of difference is normalized so that the zero corresponds with the equilibrium average. The values are normalized so that the zero corresponds with the equilibrium average, i.e. the dotted blue horizontal line. The two continuous black horizontal lines indicate the static collusive profits and the static Nash Equilibrium profits.

As we can see, retraining leads to a more profitable strategy for Algorithm 1. The algorithm learns to exploit the collusive strategy of its opponent, earning higher profits.

4.2 Non-blind Collusive Algorithm Retraining

In this Section, I show that the same results on algorithm retraining apply to the original framework of Calvano et al. (2020b), when the algorithms also observe the past actions of their opponent. In this case, the algorithms can actively enforce punishment in case of rival's deviation. As a consequence, the best strategy in response to the collusive policy might not be as simple as undercutting. In fact, the collusive policy includes a punishment action for undercutting. I plot the profits of Algorithm 1 before and after retraining in Figure 5.

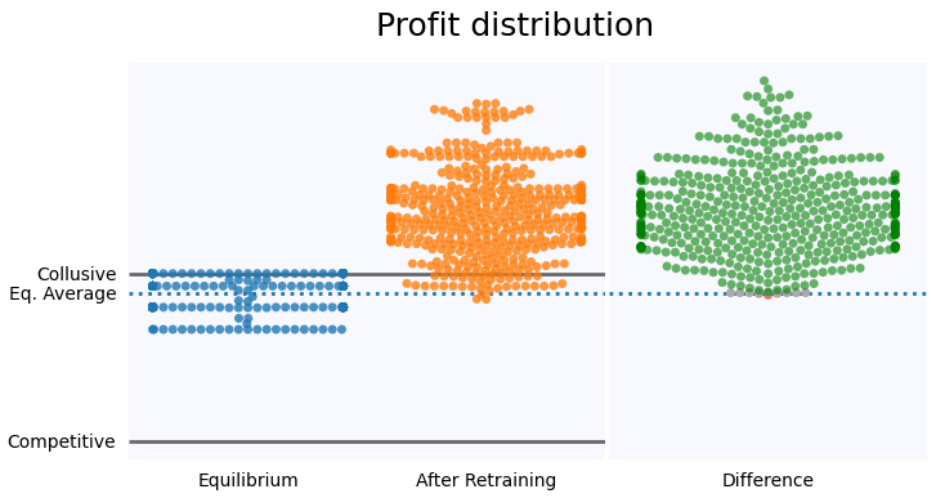

Figure 5: Equilibrium profits of Algorithm 1 in the model of Calvano et al. (2020b), before and after retraining of Algorithm 1. On the right-hand side, the distribution of difference is normalized so that the zero corresponds with the equilibrium average. The values are normalized so that the zero corresponds with the equilibrium average, i.e. the dotted blue horizontal line. The two continuous black horizontal lines indicate the static collusive profits and the static Nash Equilibrium profits.

Figure 5 confirms that the new policy of Algorithm 1 after retraining is indeed more profitable than the collusive policy. The insight from this and the previous Figure is that collusive strategies are open to exploitation from a new q-learning algorithm. In order to understand whether this insight could be used as a test for algorithmic collusion, we need to test it on a placebo case, in which the algorithms are not colluding.

4.3 Non Collusive Algorithm Retraining

Lastly, I am going to inspect what happens when one algorithm is unilaterally retrained in case the algorithms were not colluding. I would expect that if the algorithms were playing competitive strategies, one algorithm would not be able to learn any better strategy than the competitive one. In Figure 6, I plot the profits before and after retraining of Algorithm 1, in the scenario in which Algorithm2 is always playing the static Nash Equilibrium.

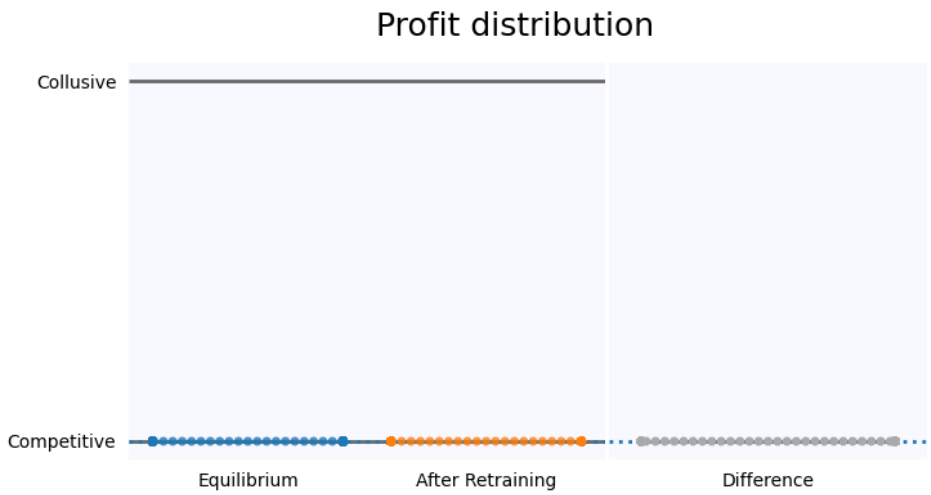

Figure 6: Equilibrium profits of Algorithm 1 in the model of Calvano et al. (2020b), where Algorithm 2 always plays the static Nash Equilibrium, before and after retraining of Algorithm 1. On the right-hand side, the distribution of difference is normalized so that the zero corresponds with the equilibrium average. The values are normalized so that the zero corresponds with the equilibrium average, i.e. the dotted blue horizontal line. The two continuous black horizontal lines indicate the static collusive profits and the static Nash Equilibrium profits.

As we can see from Figure 6, Algorithm 1 keeps playing the competitive strategy even after re-training. This intuitively makes sense, since by definition of static Nash Equilibrium, Algorithm 2 cannot be unilaterally exploited.

4.4 Re-training on observational data

As we have seen in the previous paragraphs, asynchronous learning allows q-learning algorithms to exploit their opponent strategies, in case they were colluding. However, the direct policy implication of this finding - enforcing retraining of algorithms - is clearly unfeasible. One cannot ask firms to reset their algorithms in order to check for collusion. Learning is not only time consuming but also expensive for firms that have to undergo a sequence of suboptimal actions. Moreover, this procedure requires that all other algorithms stop learning, i.e. it assumes control over all algorithms active in the market.

In this Section, I show that one can obtain the same result using only observational data. They key insight comes from the fact that algorithm exploration provides natural experiments to understand their behavior in counterfactual scenarios. In fact, thanks to algorithm exploration, algorithms generate counterfactual data on their behavior in alternative states. Given this data, one could directly compute implied value and policy functions and examinate whether the implied optimal behavior is different from the observed one.

To assemble this dataset, one needs the following ingredients: for each state-action pair $(\boldsymbol{s}, a_i)$, an estimate of payoffs $\hat{\pi}_i(\boldsymbol{s}, a_i)$ and transition probabilities $\hat{pr}(\boldsymbol{s}'|\boldsymbol{s}, a_i)$. With these ingredients, I compute the implied value and policy function. Then, I deploy such policy against the actual one to observe whether in indeed leads to different behavior and if this

behavior is indeed more profitable.

The choice of the periods to use to re-train the algorithm is important. The more recent an observation is, the more likely it is to be representative of current opponent's behavior. However, one would also prefer to include in the bootstrap sample older observations to increase the accuracy of the estimate. In other terms, the more time periods one includes in the estimation of $\hat{\pi}_i(\boldsymbol{s}, a_i)$ and $\hat{\boldsymbol{s}}'(\boldsymbol{s}, a_i)$, the more the variance of the estimates decreases, but the more the bias of the estimates increases. In the simulation results reported below, for each state-action pair $(\boldsymbol{s}, a_i)$, I use the single most recent observed state-action period to build the estimates of the payoffs and state transitions.

In Figure 7, I report the actions and profits of Algorithm 1 after retraining on the bootstrap sample, when the algorithm sees only its own past action, as in Section 3.

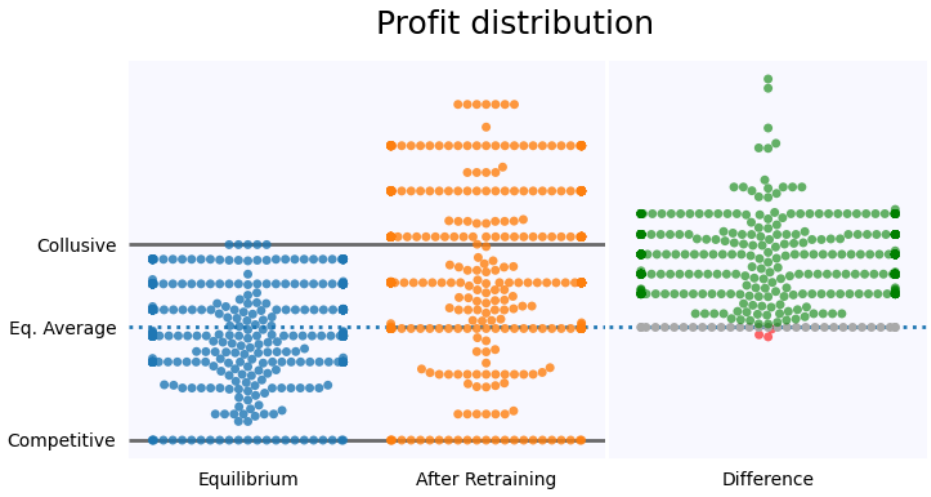

Figure 7: Equilibrium profits of Algorithm 1 in the model with blind algorithms and high competition, $\mu = 0.05$, before and after retraining of Algorithm 1 on observational data. On the right-hand side, the distribution of difference is normalized so that the zero corresponds with the equilibrium average. The values are normalized so that the zero corresponds with the equilibrium average, i.e. the dotted blue horizontal line. The two continuous black horizontal lines indicate the static collusive profits and the static Nash Equilibrium profits.

From Figure 7, we observe that the algorithm consistently changes its strategy adopting more competitive actions. The results are noisier than in the case of retraining, but this is expected since observed actions are only partially predictive of future behavior. Also profits seem to be consistently higher but the amount of noise is significant.

In Figure 8, I repeat the same exercise in the setting of Calvano et al. (2020b), when the algorithms observe both their own past action and their competitor's.

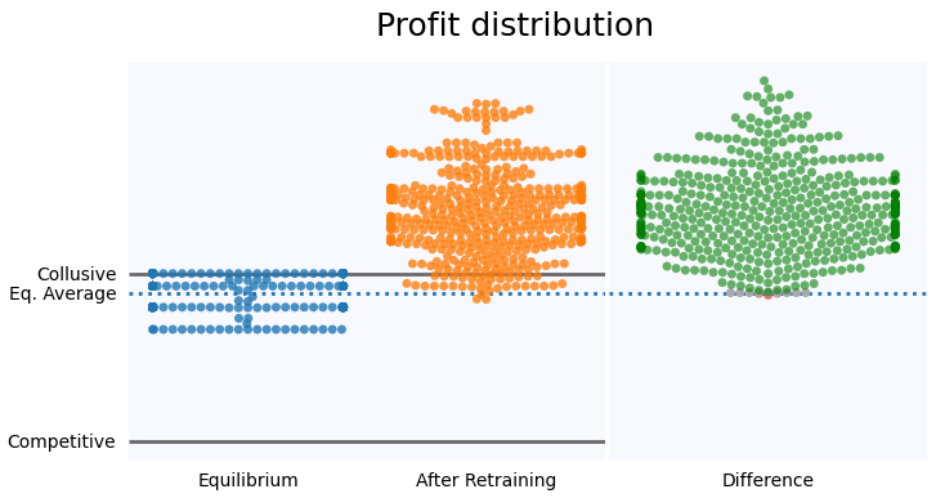

Figure 8: Equilibrium prices of Algorithm 1 in the model of Calvano et al. (2020b), before and after retraining of Algorithm 1 on observational data. On the right-hand side, the distribution of difference is normalized so that the zero corresponds with the equilibrium average. The values are normalized so that the zero corresponds with the equilibrium average, i.e. the dotted blue horizontal line. The two continuous black horizontal lines indicate the static collusive prices and the static Nash Equilibrium prices.

Again, Algorithm 1 consistently picks more competitive actions after retraining from the bootstrap sample of observed data. In this case, the pattern is more stark, but profits are not significantly higher, suggesting that the new algorithm is overfitting the bootstrapped data.

Lastly, in Figure 9, I repeat the same exercise in the case in which Algorithm 2 was always playing the static Nash Equilibrium prices. As we have previously seen, algorithms in equilibrium play competitive strategies, therefore we expect them not to find a profitable deviation through retraining.

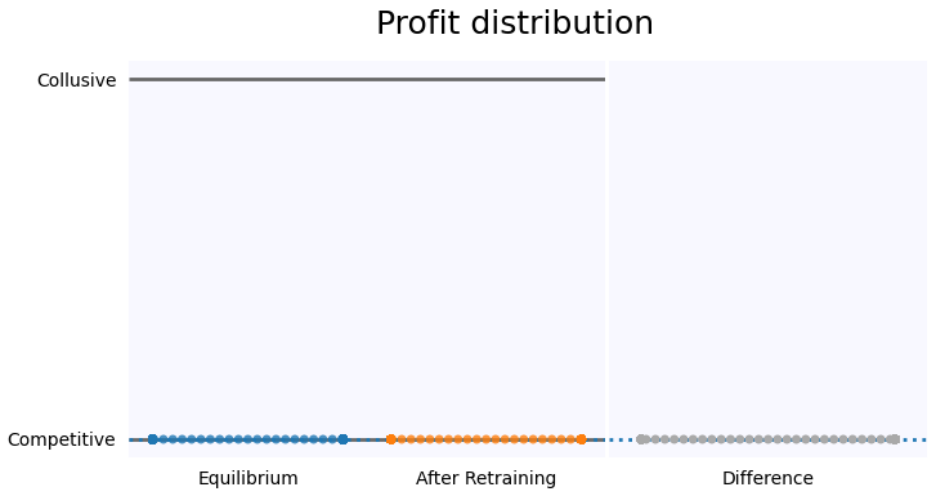

Figure 9: Equilibrium prices of Algorithm 1 when Algorithm 2 always plays the static Nash Equilibrium prices, before and after retraining of Algorithm 1 on observational data. On the right-hand side, the distribution of difference is normalized so that the zero corresponds with the equilibrium average. The values are normalized so that the zero corresponds with the equilibrium average, i.e. the dotted blue horizontal line. The two continuous black horizontal lines indicate the static collusive prices and the static Nash Equilibrium prices.

In this case, Algorithm 1 is not able to find a more profitable strategy.

5 Conclusion

In this paper, I have examined the issue of detecting algorithmic collusion. I have first shown that the inspection of the algorithm's inputs might not be sufficient in order to determine whether an algorithm can learn collusive strategies. Using the algorithmic collusion setting of Calvano et al. (2020b), I show that even if algorithms were not observing the competitor's actions, they are still able to learn reward-punishment schemes. While this is an extreme scenario, it highlights the fact that independence from competitors' actions is not sufficient to prevent algorithms to learn reward-punishment strategies with the purpose to set supra-competitive prices.

In the second part of the paper, I propose a model-free approach to detect algorithmic collusion. To the best of my knowledge, this is the first attempt to build a model-free test for algorithmic collusion, using only observational data. Building on the insight that algorithms are sophisticated but not designed to learn Sub-game Perfect equilibriums trategies, I show that unilaterally re-training one algorithm allows it to learn more profitable strategies. Remarkably, this happens only if the algorithms were colluding. The same does not occur if the algorithms were adopting competitive strategies.

Since retraining algorithms is expensive and difficult to enforce, I show that one can obtain similar results relying only on observational data. In fact, a key feature of reinforcement learning is that the algorithm keeps exploring new strategies in order to adapt to changes in a dynamic environment. These exploration phases provide natural experiments to test ex-post for collusive behavior.

References

John Asker, Chaim Fershtman, and Ariel Pakes. Artificial intelligence and pricing: The impact of algorithm design. Technical report, National Bureau of Economic Research, 2021.

Stephanie Assad, Robert Clark, Daniel Ershov, and Lei Xu. Algorithmic pricing and competition: Empirical evidence from the german retail gasoline market. 2020.

Alan W Beggs. On the convergence of reinforcement learning. *Journal of economic theory*, 122 (1):1–36, 2005.

Emilio Calvano, Giacomo Calzolari, Vincenzo Denicolò, Joseph E Harrington, and Sergio Pastorello. Protecting consumers from collusive prices due to ai. *Science*, 370(6520):1040–1042, 2020a.

Emilio Calvano, Giacomo Calzolari, Vincenzo Denicolo, and Sergio Pastorello. Artificial in-

telligence, algorithmic pricing, and collusion. *American Economic Review*, 110(10):3267–97, 2020b.

Le Chen, Alan Mislove, and Christo Wilson. An empirical analysis of algorithmic pricing on amazon marketplace. In *Proceedings of the 25th International Conference on World Wide Web*, pages 1339–1349, 2016.

Ariel Ezrachi and Maurice E Stucke. Artificial intelligence & collusion: When computers inhibit competition. *U. Ill. L. Rev.*, page 1775, 2017.

Ariel Ezrachi and Maurice E Stucke. Sustainable and unchallenged algorithmic tacit collusion. *Nw. J. Tech. & Intell. Prop.*, 17:217, 2019.

Joseph E Harrington. Developing competition law for collusion by autonomous artificial agents. *Journal of Competition Law & Economics*, 14(3):331–363, 2018.

Mitsuru Igami. Artificial intelligence as structural estimation: Deep blue, bonanza, and alphago. *The Econometrics Journal*, 23(3):S1–S24, 2020.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529 (7587):484–489, 2016.

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