Game Theory

Algorithmic Collusion*

Tetsuya Hoshino

March 27, 2022

Algorithmic pricing is the practice of setting the prices of goods by computer algorithms—not by humans. It is not just a pricing rule but rather a learning algorithm, powered by artificial intelligence, that experiments with different pricing rules in search of one that yields the highest profits.

In this note, we explore the collusion by multiple pricing algorithms. The pricing algorithms adopted by multiple firms may autonomously collude to set supracompetitive prices even without any communication (Calvano et al., 2020).

What Is Wrong with Algorithmic Collusion? Suppose that firms 1 and 2 are competing in an online market. Their demand depends on prices. The manager of firm 1 decides to install a software program to perform the pricing task. This program uses all available information, including all past prices (which are available online). The software program is not just a pricing rule but a learning algorithm that experiments with different pricing rules in search of the most profitable one. The manager of firm 2, in fact, has recently adopted a similar pricing algorithm, which also experiments with different pricing rules. Neither manager knows that the rival firm has been using a pricing algorithm, because a pricing strategy is confidential to the firm. Neither manager also knows how his or her pricing algorithm works, because recent algorithms are too complex to understand. All the managers know is that the algorithms are purported (by the software developers) to result in highly profitable pricing rules.

Assume that the two pricing algorithms autonomously collude for supracompetitive prices—with no communication between the two algorithms, let alone between the two managers. If you were a regulator, would you say that these two firms are (illegally) colluding?

This form of tacit collusion would defy current antitrust policy, which typically targets only explicit agreements among would-be competitors (Harrington, 2018).

Whether or not you determine that they are (illegally) colluding, if you want to prevent the supracompetitive algorithmic pricing, what would you do?

1 Single-Agent Reinforcement Learning

Reinforcement learning is a machine-learning method that an intelligent agent, called a machine hereafter, is rewarded for desirable behavior and punished for undesirable behavior. In reinforcement learning, we consider a machine that faces a game-like environment. The machine receives either rewards or penalties for the actions that it performs. It attempts to maximize the total reward through a process of trial and error. We, as program designers, set the rules of the game, including rewards and penalties, but we do not program how the machine should play the game. Thus, it is up to the machine to decide how to perform the task to maximize the total reward, starting with totally random attempts and hopefully ending with sophisticated tactics and superhuman skill.

In what follows, we study the (simplified) **Q-learning**, which is one of the most widely used reinforcement-learning algorithms (Watkins & Dayan, 1992). No preliminary knowledge is required.

1.1 Markov Decision Process

To begin with, we set up a dynamic environment with which a single machine interacts.

Definition 1. A **Markov decision process (MDP)** in discrete time $\mathbb{T} = \{1, 2 \ldots\}$ is a tuple $M = \langle S, s_1, A, \tau, \pi \rangle$ such that:

1. S is a finite state space.
   - $s_1 \in S$ is an initial state.
2. A is a finite action space.
3. $\tau : S \times A \to S$ is the transition function.
   - τ assigns to time-t state-action pair (s, a) time-$(t+1)$ state $\tau(s, a)$.
4. $\pi : S \times A \to \mathbb{R}$ is the payoff function (often called the reward function).
   - π assigns to time-t state-action pair (s, a) time-t payoff $\pi(s, a)$.

In an MDP, we define a machine's behavior as follows:

Definition 2. In an MDP M, a **strategy** is a function $\sigma : \bigcup_{t=1}^{\infty} H^t \to \Delta(A)$ that assigns to history h^t a distribution $\sigma(h^t)$, where H^t is the set of all time-t histories $h^t = (s_1, a_1, \ldots, s_t)$ consisting of the past states $(s_1, \ldots, s_t)$ and actions $(a_1, \ldots, a_{t-1})$.[1]

In an MDP, the following class of strategies are important:

Definition 3. In an MDP M, a **Markov strategy** is a function $\sigma : S \to \Delta(A)$ that assigns to state s an action distribution $\sigma(s)$. Let Σ denote the set of all Markov strategies.[2]

Remark 1. A Markov strategy is a special case of strategies. Indeed, a strategy σ is a Markov strategy if and only if every time t, it depends on time-t state s_t but not on the past information $(s_1, a_1, \ldots, s_{t-1}, a_{t-1})$. $\square$

In an MDP, the goal of a machine is to maximize its discounted sum of payoffs by optimize its choice of (Markov) strategy. This leads to the following concept.

[1] In the machine-learning literature, strategies are usually called policies.
[2] In the machine-learning literature, Markov strategies are usually called Markov policies.

Note (Recursive Structure of Value Function): We do not prove that a value function satisfies the Bellman equation in an MDP, but the reader can convince themselves by noticing that the defining equation has the following recursive structure:

$$V(s) = \max_{a \in A} \left\{ \begin{array}{c} \pi(s,a) + \delta \max_{\sigma \in \Sigma} \displaystyle\sum_{t=1}^{\infty} \delta^{t-1} \pi(s_t, a_t) \\ \text{subject to } \begin{cases} s' = s_1 \\ a_t = \sigma(s_t) \quad \text{for each } t \in \mathbb{T} \end{cases} \end{array} \right\}.$$

Definition 4. In an MDP M with a discount factor $\delta \in (0,1)$, we define the **value function** $V : S \to \mathbb{R}$ as follows:

$$V(s) = \max_{\sigma \in \Sigma} \sum_{t=1}^{\infty} \delta^{t-1} \pi(s_t, a_t)$$
$$\text{subject to } \begin{cases} s = s_1 \\ a_t = \sigma(s_t) \quad \text{for each } t \in \mathbb{T}. \end{cases}$$

This concept evaluates the value of a given state $s \in S$. For example, if a machine can do well when starting from a given state $s \in S$, the machine evaluates this state s highly.

Remark 2. Note that the maximization is taken over Markov strategies but not over all strategies. In the MDP, it is known that an optimal strategy belongs to the class of Markov strategies. That is, it suffices to take the maximum over Markov strategies. □

1.2 Q-Function

How can we find the value function V? It is known that the value function V satisfies the Bellman equation in our MDP setting:

Definition 5. In an MDP M with a discount factor $\delta \in (0,1)$, the **Bellman equation** of a function V is the following equation:

$$V(s) = \max_{a \in A} \Big\{ \pi(s,a) + \delta V(s') \Big\} \qquad \forall\, s \in S, \tag{1}$$

where $s' = \tau(s,a)$ is a state in the next period. On the right-hand side, the first term denotes the today's payoff and the second term denotes the continuation value after tomorrow.

Q-Function Once we have the Bellman-equation representation, the question is reduced to how to approximate the Bellman equation. Then, we define a Q-function such that it "imitates" the value function V of the Bellman equation.

Definition 6. In an MDP M with a discount factor $\delta \in (0,1)$, the **Q-function** $Q : S \times A \to \mathbb{R}$ as follows:

$$Q(s,a) = \pi(s,a) + \delta \max_{a' \in A} \Big\{ Q(s', a') \Big\}, \tag{2}$$

where the first term on the right-hand side is the today's payoff and the second term is the continuation Q-value (after tomorrow).

By definition, the Q-equation (2) is similar to the Bellman equation (1). Moreover, it is related to the value function as follows:[3]

$$V(s) \equiv \max_{a \in A} Q(s,a).$$

Since both S and A are finite, we can represent the Q-function as an $|S| \times |A|$ matrix. We call it the **Q-matrix**.

1.3 Q-Learning Algorithm

If the machine knew the Q-matrix, it could learn the value function V just by finding the optimal action for any given state s. The Q-learning is essentially a method for estimating the Q-matrix without knowing the transition function τ or the payoff function π.

Definition 7 (Q-learning)**.** The Q-learning algorithm estimates the Q-matrix by the following iterative procedure:

1. Fix an arbitrary initial matrix $Q_1 \in \mathbb{R}^{|S| \times |A|}$.
2. Every time $t \geq 1$, the machine chooses "certain" action $a_t = a$ in state $s_t = s$.
   (a) The machine observes a payoff $\pi(s,a)$ and a state $s' = \tau(s,a)$.
   (b) The machine updates the (s,a)-th entry as follows:[4]

$$Q_{t+1}(s,a) \leftarrow (1-\alpha) Q_t(s,a) + \alpha \left[\pi(s,a) + \delta \max_{a \in A} Q_t(s', a) \right],$$

   where $\alpha \in (0,1)$ is a learning-rate parameter. The machine changes no other entries.

Remark 3. Definition 7 does not specify how the machine chooses "certain" action $a_t = a$ in state $s_t = s$. Here we use the **ϵ-greedy algorithm**, which chooses the currently optimal action (i.e., the one with the highest Q-value in state s) with a fixed probability $1-\epsilon$ and randomizes

[3] To see this, we take $\max_{a \in A}$ on both sides of the defining equation (2):

$$\max_{a \in A} Q(s,a) = \max_{a \in A} \Big\{ \mathbb{E}[\pi(s,a) \mid s,a] + \delta \mathbb{E}\Big[\max_{a' \in A} \Big\{ Q(s',a') \Big\} \mid s, a \Big] \Big\}.$$

If we identify $\max_a Q(s,a) \equiv V(s)$ for each s, then this is exactly the same as the Bellman equation.

[4] This $\leftarrow$ notation is often used to mean that the left-hand side is updated by the right-hand side.

uniformly across all actions with probability ϵ. That is, ϵ is the rate of an exploration mode, in which the machine explores better actions, while $1-\epsilon$ is the rate of an exploitation mode, in which the machine exploits to obtain better rewards.

How should we choose the exploration rate ϵ? A popular choice is to use a time-declining exploitation-exploration rate ϵ_t such that

$$\epsilon_t = e^{-\beta t},$$

where $\beta > 0$ is a parameter. The time-decline property means that initially the machine aggressively explores many actions at random, but as time passes, it sticks to the greedy choice more and more often. $\square$

Remark 4. In the MDP setting, the Q-learning algorithm converges to the optimum under certain conditions. However, the convergence is typically slow. This is because the algorithm updates only one cell of the Q-matrix at a time, and the approximation of the true matrix generally requires that each cell be visited many times. The larger the state or action space, the more iterations will be needed. $\square$

2 Bertrand Competition with Differentiated Products

Next, we set up an economic model of firms' competition. Following the previous note, we employ a Bertrand game with differentiated products.

2.1 Stage Game

As we have studied the Bertrand game, we reproduce the model for reference. For details, see the previous note on collusion.

Consider a normal-form game $G = \langle I, (A_i, u_i)_i \rangle$ such that:

1. $I = \{1, 2\}$ is the set of firms.

2. $A_i = \mathbb{R}_+$ is firm i's action space.

3. $\pi_i : A \to \mathbb{R}$ is firm i's payoff function defined by $\pi_i(p_i, p_{-i}) = (p_i - 1)q_i(p_i, p_{-i})$, where the demand is in the logit form:

$$q_i(p_i, p_{-i}) = \frac{\exp\left\{\frac{2-p_i}{1/4}\right\}}{1 + \exp\left\{\frac{2-p_i}{1/4}\right\} + \exp\left\{\frac{2-p_{-i}}{1/4}\right\}}.$$

Here we summarize the following basic property of this game:

Lemma 1. *Consider the normal-form game G of Bertrand game.*

1. *It has a symmetric Nash equilibrium* (p^N, p^N)*:*
   - *Nash equilibrium price* $p^N \approx 1.473$*.*
   - *Nash equilibrium profit* $\pi^N \equiv \pi_i(p^N, p^N) \approx 0.223$*.*
2. *It has a symmetric "monopoly" price profile* (p^M, p^M) *that maximizes the total profit:*
   - *Monopoly price* $p^N \approx 1.925$*.*
   - *Monopoly profit* $\pi^N \equiv \pi_i(p^N, p^N) \approx 0.337$ *(after division).*

Proof. See the previous note on collusion. ■

2.2 Repeated Game

Now we assume that firms 1 and 2 repeat the Bertrand game G under perfect monitoring. In this repeated game, if a discount factor δ is high enough, then the firms collude to set supracompetitive prices (i.e., prices above any stage-game Nash equilibrium prices) in some subgame perfect equilibrium. The proof is constructive. That is, we construct a particular grim trigger strategy profile and then show that it is a subgame perfect equilibrium.

One possible interpretation of this collusion is "explicit collusion" in the sense that players might have communicated to agree to play the triggering strategy profile.

3 Algorithmic Collusion

We are interested in whether two machines, powered by the Q-learning algorithms, may be able to autonomously learn to collude for supracompetitive prices.

The Q-learning reviewed above is a single-agent Q-learning, but our repeated-game application inevitably involves multiple agents. The multi-agent Q-learning is very difficult, and indeed there are no general convergence results, unlike the single-agent Q-learning (Remark 4). Hence, we do not know whether the pricing algorithms converge or not, and even if so, whether it converges to collusive or competitive behavior. Then, we. We will rely on experiments.

3.1 Experiment Design

Assume that each firm i is a Q-learning machine, called machine i hereafter. Each machine i has the same action space A_i and state space S, which we shall define below.

Discretized Action Space In the theoretical model, firm i's action space $A_i = \mathbb{R}$ is continuous, but since the Q-learning requires finite action spaces, we have to discretize them. Specifically, we take the following finite set, denoted A_i:

$$\mathsf{A}_i = \Big\{ \text{15 equally spaced points in the interval } \Big[p^N - \xi\Big(p^M - p^N\Big), p^M + \xi\Big(p^M - p^N\Big)\Big]\Big\},$$

with a parameter $\xi > 0$. The lowest price is below p^N, while the highest price is above p^M. Note that A_i may not include the exact price p^N or p^M. By definition, $|\mathsf{A}_i| = 15$.

Memory In a repeated game, we usually assume that each player has perfect recall, allowing her strategy to depend on the entire history $h^t \in \bigcup_t H^t$ of past actions. However, since it requires an infinite state space, we cannot keep this assumption.

To ensure that a state space is finite, we suppose a bounded memory for each machine. Specifically, we posit that each machine "remembers" only for one period. That is, machine i's (Markov) strategy at time t depends only on the time-$(t-1)$ action profile $a^{t-1} = (a_1^{t-1}, a_2^{t-1})$. Since our discretized action space A_i has 15 actions, we have the state space $\mathsf{S} = \mathsf{A}_1 \times \mathsf{A}_2$, with cardinality $|\mathsf{S}| = 15^2$.

3.2 Experiment Result

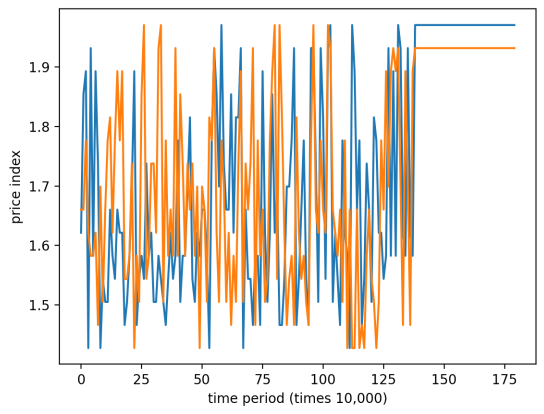
Remark 5. Calvano et al. (2020) provide their codes mainly in `Fortran`, which I cannot read or write. The reported results are based on my own `Python3` code (which may contain coding errors). I attempt to replicate only two experiments among many experiments. Note that the original results are the average of many sessions, but my results are not. As addressed in the paper, the limit strategies display considerable variation from session to session, so that the reader should not consider the reported results representative ones. □

Convergence Criterion For games played by Q-learning algorithms, there are no general convergence results. Hence, we do not know whether the algorithms converge, or, if so, whether they converge to a (stage-game) Nash equilibrium. To verify convergence (by experiment), we deem the Q-learning algorithms to "successfully converge" if for each machine, the optimal strategy does not change for a long enough periods (e.g., 100,000 consecutive periods). More specifically, if for each machine i and each state s, action $\operatorname{argmax}\{Q_{it}(a, s)\}$ stays constant for many periods (e.g., 100,000 periods), we say that the algorithms attains stable behavior.

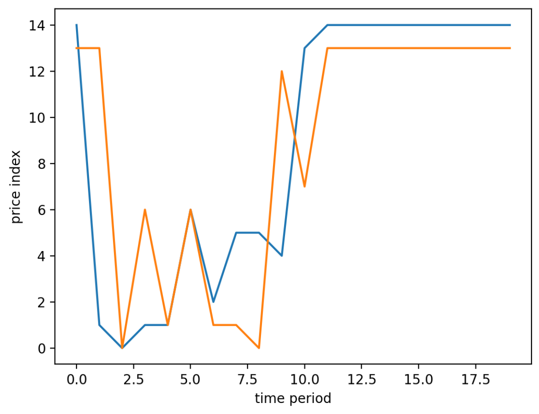
Convergence Result In many sessions, *the machines have converged to supracompetitive prices.* A large number of the converged sessions have ended at supracompetitive prices, while a small number of them have ended at (nearly) stage-game Nash equilibrium prices.

We observe two kinds of convergence results:

1. The machines charge constant prices p_1^* and p_2^* (Figure 1a).
    - They sometimes offer the same price $p_1^* = p_2^*$.
    - They sometimes offer different prices $p_1^* \neq p_2^*$, but the difference is typically small.
2. The machines display price cycles, many of which have a period of 2 (Figure 2a).

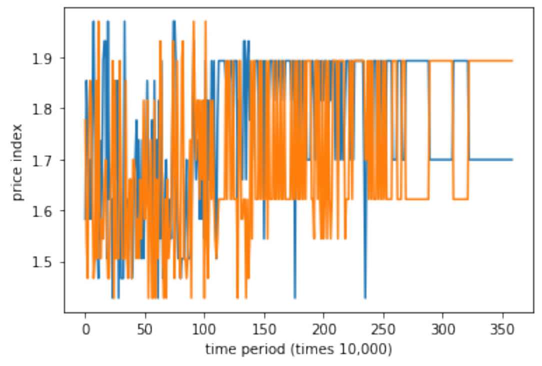


(a) supracompetitive price

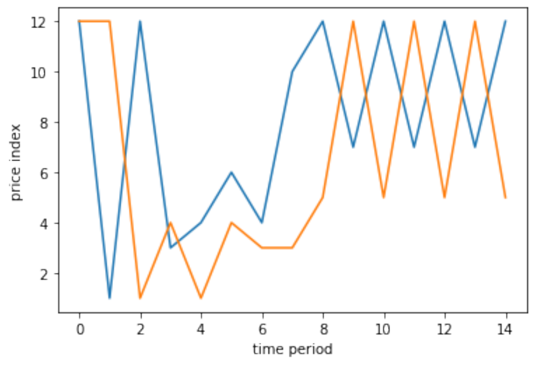


(b) reward-punishment scheme

Figure 1: algorithmic collusion with constant prices



(a) supracompetitive price



(b) reward-punishment scheme

Figure 2: algorithmic collusion with price cycles

Reward-Punishment Scheme Collusion is not a synonym of high prices but crucially involves a reward-punishment scheme (Harrington, 2018). To see if the algorithmic pricing strategies involve a reward-punishment scheme, we examine how a unilateral deviation affects the future play. Using the trained Q-algorithms (with supracompetitive prices), we program machine 1 to deviate to a nearly stage-game Nash equilibrium price for one period. Then, we observe how machine 2 responds to this deviation.

We observe that *the algorithmic collusion involves a reward-punishment scheme*. Right after the programmed deviation, the two machines enter into a "price war" but revert to the collusive prices after a while (Figures 1b and 2b). This is consistent with the reward-punishment scheme, which punishes the deviator with low prices.[5]

[5]My result is not as "clean" as Calvano et al. (2020, Figure 4), in which the machines revert to the collusion prices gradually. Nevertheless, I am not sure if this difference is important or not. According to Calvano et al. (2020, Figure 5), they have observed a variety of prices after the deviation, even including prices higher than the collusion levels. It is suggested that post-deviation prices might have varied considerably.

References

Calvano, E., Calzolari, G., Denicolo, V., & Pastorello, S. (2020). Artificial intelligence, algorithmic pricing, and collusion. *American Economic Review*, *110*(10), 3267–97.

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

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*(3), 279–292.