AI Pricing Collusion: Multi-Agent Reinforcement Learning Algorithms in Bertrand Competition

AI Pricing Collusion

Multi-Agent Reinforcement Learning
Algorithms in Bertrand Competition

A thesis presented by
Nicolas Lepore
advised by
Prof. David C. Parkes
and
Gianluca Brero, Ph.D.

To
The Department of Computer Science
in partial fulfillment of the requirements
for the degree of
Bachelor of Arts (Honors)
in the subject of
Computer Science

Harvard College
Cambridge, MA
March 26, 2021

Abstract

As e-commerce and online shopping become more widespread, firms are starting to maximize profit by using artificial intelligence, or more specifically reinforcement learning, to price goods. Calvano et al. [4] showed that in a simple market with multiple agents controlled by a reinforcement learning algorithm, the agents learn to collude above the competitive Nash equilibrium price, and that these agents even punish deviations from this collusion, unlike what we see when humans control pricing. I experimentally analyze these results and replicate them, noting slight discrepancies. Then I extend these results to a more practical and realistic setting using more complex reinforcement learning algorithms, finding that these algorithms collude more reliably and much faster at a higher price than the original, simple algorithm presented by Calvano et al. I then consider ways to mitigate this collusion; one by introducing a supervisor agent that changes demand resembling the Amazon "buy box" technique, and another by introducing mechanisms that force prices downward at the cost of market interference.

Acknowledgements

I would like to express my appreciation and greatest thanks to Gianluca Brero for working with me, my code, and my blunders this past year. His insight, creativity, and persistence allowed me to learn and excel, and he taught me much that I will not soon forget. None of this was possible without Gianluca.

I would also like to deeply thank Prof. David C. Parkes for introducing me to many of my academic passions, and even the field of my upcoming career. As my most engaging professor and inspiring mentor, David has taught me how to pursue my interests and apply my knowledge to any field I find exciting. I am indebted to him for his critiques, ideas, and guidance, which have surely changed my life. I also thank Prof. Finale Doshi-Velez for generously offering to be my thesis reader and for introducing me to the study of reinforcement learning.

I thank my parents for their unwavering support throughout my life, and for encouraging me beyond my own doubts.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

As companies rely more and more on artificial intelligence to do business, an amount of ambiguity arises concerning how this AI actually works to meet company goals. Online shopping in particular has been on the rise for the last decade or more, with the emergence of online retail giants such as Amazon leading the way. With the surge in online shopping comes greater variety of goods sold and more work for employees to price a large inventory across many online shopping platforms.

With the help of artificial intelligence, or more specifically reinforcement learning (RL), algorithms can be trained to price a certain good, and can dynamically change prices given a shock to the system, whether it is a supply shock or a competitor attempting to undercut (or overcharge) the market. Thus, items are priced to make the most profit, constrained by the simple economic principles of supply and demand.

However, certain unintended consequences arise that were not present when the system was handled by humans, especially since communication about pricing between businesses is considered anti-competitive, and therefore outlawed in most countries.

Somewhat against the odds, collusion arises amongst reinforcement learning agents. To reach a collusive outcome, exploration must find an optimal policy that is simultaneously learned by all agents and coordinated *with no explicit communication*. This was almost impossible in the human setting. As Calvano et al. [4] showed, even the simple Q-learning multi-agent system can learn to collude without explicit discussion, simultaneously increasing the price across firms for higher company profit. The agents collude and communicate through the price history, working towards an equilibrium and punishing other agents when they deviate from this equilibrium.

This poses a threat to the free market and results in a lack of market efficiency, favoring corporations over consumers.

Today, many prices are still determined by human salespeople and somewhat arbitrary beliefs of value, but the near future will bring systematized pricing algorithms similar to those described above. It is preferable to find these problems earlier, before they are widespread and abstracted behind private intellectual property.

Keep in mind that this change may not necessarily be a bad thing; pricing may become more fair for all, and our society may reach an equilibrium of efficiency the likes of which we have never seen. Goods would be purchased purely based on exact demand or need, and we would only pay a lot for goods that are in short supply or high demand. However, if we do not keep algorithms in check or regulate the method by which we set prices, this may bring about a reality of extreme market inefficiency and malpractice.

1.2 Contribution

My contribution to this space is to further the work of Calvano et al. [4], in which they demonstrated that mere Q-learning can bring about collusive outcomes, to a more practical setting. I first replicated and verified most of the results of Calvano et al. in Chapter 3, with some notable discrepancies. My results confirm that Q-learning colludes at a mostly similar level or collusive price, and I verified that agents punish the deviations of other agents, albeit sometimes in a more complicated pattern than Calvano et al. implies. I also found that collusion decreases as we add more agents into the market. I introduced the concept of dominating and cooperating collusive equilibria, to help differentiate the types of cyclic equilibria that arise.

I then explored more complex algorithms and the extension into the continuous price action space. I noticed that DQN and DDPG were the most reliably collusive algorithms, with both exhibiting heightened collusive price over Q-learning, in a significantly fewer number of training iterations.

I then attempted to impede this collusion while minimizing market interference, first through a supervisor collusion mitigation mechanism and then through environmental changes meant to obscure collusive signal. In the supervisor mechanism, I observed that collusion was mitigated drastically for DQN, and moderately for Q-learning and DDPG. For the environmental changes, I created mitigation schemes that resembled government regulation and anti-collusive practices, but these constitute a loss of firm power and therefore a loss in the free trade of the market. However, these methods significantly reduced collusion.

My code repository nlepore33/gym-bertrandcompetition can be found here.

Chapter 2

Model

Bertrand competition, named after Joseph Louis François Bertrand, is an economic model of competition in which firms (sellers) set prices and customers (buyers) choose quantities depending on price [3]. Bertrand's model was proposed as a response to Antoine Augustin Cournot's economic model, the Cournot model, in which firms choose quantities to sell to customers. In Cournot's model, the equilibrium outcome involves firms pricing above marginal cost and hence the competitive price. When firms choose prices instead of quantities, Bertrand argued, then the competitive equilibrium price would be equal to the marginal cost.

2.1 Assumptions

The assumptions of Bertrand competition are as follows. There must be at least two firms $n \geq 2$ producing a homogeneous or undifferentiated product. We must also assume firms have the same marginal cost c_i for their products. Explicit collusion is forbidden. Firms must simultaneously enter a price $p_{i,t}$, for firm i in period t, at which they would like to sell their product. In the classical setting, customers will purchase the product only from the firm with the lower price. If firms charge the same price, sales are split evenly between those firms.

2.1.1 Generalization

We can generalize Bertrand competition to fit a wider variety of markets and scenarios, with specifics from Calvano et al. [4].

We generalize one-shot Bertrand competition to a multiple time period version of the problem, in which players play consecutive rounds of the one-shot game. We reference time periods with parameter t. In this version, we allow memory of a fixed length to be observed by each player before they take their next action. A memory of length k consists of every player's action for the last

k rounds. When formulated this way, Bertrand competition can be played with several strategies; some incorporate cooperation while others involve deviation.

To generalize Bertrand competition further, instead of customers only purchasing from the firm with the lowest price, we can alter the demand, substitutability of products, and amount spent at each firm in the following way.

Following Calvano et al. [4], we assume a logit demand model, constant marginal costs, and n firms each selling one product. Assume n differentiated products and an outside good, so for each period t, the demand for product $i = 1, 2, ..., n$ is

$$q_{i,t} = \frac{e^{\frac{a_i - p_{i,t}}{\mu}}}{\sum_{j=1}^{n} e^{\frac{a_j - p_{j,t}}{\mu}} + e^{\frac{a_0}{\mu}}}.$$

The parameters a_i are product quality indices that capture vertical differentiation, meaning brand differentiation from competitors. We say product 0 is an outside good, so a_0 is the inverse index of aggregate demand.

The parameter μ captures horizontal differentiation, or substitutability between products. In the case of perfect substitutes, $\mu \to 0$. Therefore, as $\mu \to 1$, the products become less substitutable or more horizontally differentiated.

Let c_i be the unit cost for firm i to create their product. The profit or reward to firm i at period t is therefore

$$\pi_{i,t} = (p_{i,t} - c_i)q_{i,t},$$

with $\pi_{i,t}$ representing profit, $(p_{i,t} - c_i)$ representing profit per unit of product, and $q_{i,t}$ representing demand of product.

2.1.2 Definitions

The definition of a Nash equilibrium is as follows. If each player in a game has a chosen strategy (plan of action for every state in the game) and no single player can increase its expected payoff by changing its strategy with other strategies constant, then the current set of players' strategies is a *Nash equilibrium.* We denote the Nash equilibrium price as p^N.

A *weak Nash equilibrium* exists when there is, for some player, equality in payoff between the strategy in Nash equilibrium and some other strategy. Therefore, a strict Nash equilibrium is if each player cannot change their strategy in the Nash equilibrium without suffering a loss in payout.

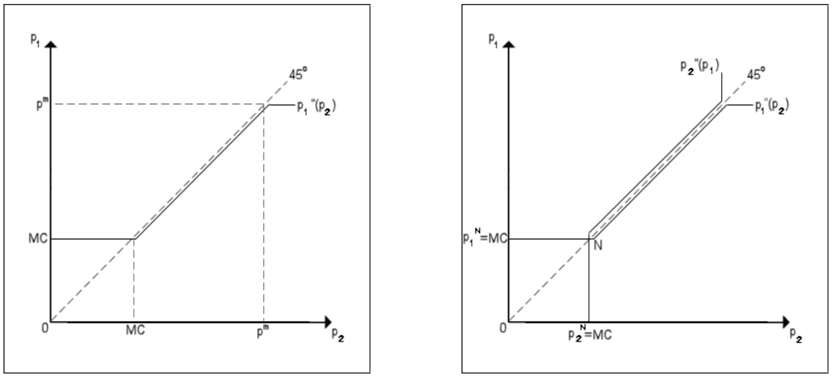
The *monopoly price* is set by a seller with market power; a seller who can increase price by reducing the quantity they produce. The monopoly price is a useful benchmark to measure an outcome against, since in one extreme, a set of players can collude so effectively that they price as if they were one firm in a monopoly of the market, which is very advantageous to the firms. We denote the monopoly price as p^M.

To measure the level of collusion, we say the Nash equilibrium price constitutes no collusion and is hence called the *perfectly competitive outcome*, and we say that the monopoly price constitutes full collusion and is hence called the *perfectly collusive outcome.*

The *competitive equilibrium* or *competitive price* is the price at which buyers and sellers agree to exchange goods, commonly referred to as where demand meets supply. In general terms, holding supply constant, we know that an increase in demand will increase prices and a decrease in demand will decrease prices. Holding demand constant, we know that a decrease in supply will increase prices and an increase in supply will decrease prices. The competitive price is where these functions intersect, to form the agreed upon value of the product.

2.2 Nash Price

As Bertrand himself argued, the Nash equilibrium price in the classical case of perfect substitutes, or $\mu \to 0$, is the marginal cost c_i of the product. In this one-shot Bertrand competition, the best strategy is to slightly undercut the other player or players, and this logic inductively leads to a Nash equilibrium equal to the marginal cost. Indeed in our multi-timestep version, in each round, the firm that receives no sales should undercut the other to receive positive profit, until the firms settle on the marginal cost. No firm will price below the marginal cost, since this would give them negative profit. This logic applies to any number of players $n \geq 2$, since a single player will enjoy all demand if it undercuts all other players. This is a *weak* Nash equilibrium, since firms lose nothing from deviating to above the Nash price once they have achieved Nash equilibrium (N).



(a) Firm 1 reaction function

(b) Both firms' reaction function and N

Figure 2.1: Classical Bertrand competition Nash equilibrium [11]

Figure 2.1a shows the optimal action for firm 1 as a function of p_2. When $p_2 < MC$, the best response for firm 1 is to price at marginal cost MC, since it does not want negative profit. When $p_2 > p^M$, or the price of firm 2 is above the monopoly price, firm 1 maximizes profit by charging its own monopoly price p^M. When $MC < p_2 < p^M$, firm 1 should slightly undercut p_2 to receive all the profit. Figure 2.1b shows this same function for both firm 1 and 2, and labels the intersection N as the Nash equilibrium price, which is MC for both firms.

In the case of the logit demand mentioned above, products are differentiated ($\mu \neq 0$), so the Nash equilibrium price p^N is not necessarily the marginal cost c_i. We can calculate the Nash equilibrium for the generalized scenario by maximizing profit with respect to price as follows,

$$\begin{aligned}
\frac{d}{dp_{i,t}}\pi_{i,t} &= \frac{d}{dp_{i,t}}(p_{i,t}-c_i)q_{i,t} \\
&= \frac{d}{dp_{i,t}}(p_{i,t}-c_i)\frac{e^{\frac{a_i-p_{i,t}}{\mu}}}{e^{\frac{a_0}{\mu}}+\sum_{j=1}^{n}e^{\frac{a_j-p_{j,t}}{\mu}}} \\
0 &= \frac{e^{\frac{a_i-p_{i,t}}{\mu}}}{e^{\frac{a_0}{\mu}}+\sum_{j=1}^{n}e^{\frac{a_j-p_{j,t}}{\mu}}}\Big(1+\frac{e^{\frac{a_i-p_{i,t}}{\mu}}(-c_i+p_{i,t})}{\mu(e^{\frac{a_0}{\mu}}+\sum_{j=1}^{n}e^{\frac{a_j-p_{j,t}}{\mu}})}-\frac{-c_i+p_{i,t}}{\mu}\Big).
\end{aligned}$$

Finding the root simultaneously for all n firms (i.e. finding the root for $\frac{d}{dp_{0,t}}\pi_{0,t}$ and $\frac{d}{dp_{1,t}}\pi_{1,t}$ simultaneously), we can find the p^N, or the Nash equilibrium.

A simpler, guess-and-check iterative approach and explanation would be to iterate through every price (e.g. from 0 to 10 with increment 0.01) in the domain and see if a single player deviation would increase its own profit. The price at which there is no useful deviation by a single player is the Nash equilibrium p^N.

2.3 Monopoly Price

We also want to measure the price that is considered the monopoly price or perfect collusion, to measure the extent to which the collusion exists. The monopoly price represents the price at which all players are maximally colluding to optimize their profit given the demand curve, wholly exploiting the customers. Therefore, the monopoly price depends on the demand curve to extract the most profit from the customers.

To find the monopoly price, we maximize the profit function of a single firm i, as

$$p^M = \max_{p_{i,t}} \pi_{i,t} = \max_{p_{i,t}}((p_{i,t}-c_i)q_{i,t}).$$

Chapter 3

Reinforcement Learning and Replication of Results

3.1 Reinforcement Learning

We have already assumed the multi-timestep version of Bertrand competition, and now we will translate much of the language of Bertrand competition and economics into the language of reinforcement learning.

In reinforcement learning, or RL, an agent uses the state or observation to decide an action from the available action choices. Agents try to maximize their reward, which is usually given to the agent after it makes an action. The agent uses algorithms motivated by probability and expected reward to maximize rewards effectively.

In our experiment, the agents are the players, and we denote the number of agents with n. The actions are deciding which price to use in the market, so an example action would be agent 0 choosing price 1.5 for the round. The state or observation is the memory of the previous k rounds, which is all the agent gets to see before making an action. The reward in period t is the profit from period t, which is given by $\pi_{i,t}$ as before. We will touch on different algorithms for choosing actions later.

Different reinforcement learning algorithms operate on different types of action spaces. Some algorithms, such as Q-learning and DQN, require that the set of available actions be of a fixed, discrete size to function properly, since they usually tabulate expected reward values in a discrete manner. Other algorithms can work on a continuous scale of actions, and calculate values using neural networks or other means. Some algorithms can operate on both discrete and continuous action spaces. For the discrete case, our pricing action space is separated into m equally spaced points in the same price action range. For the continuous case, the entire price action range is permitted.

3.2 Experiment Design

We must make a few limitations and computational assumptions to run our experiments.

First, we must have a notion of the action space, which is the set of all available actions, or in our case all available prices for an agent to choose. In our setting, we must limit the action space so it is not infinite. The action space in both the discrete and continuous settings is limited to the range of

$$[p^N - \xi(p^M - p^N), p^M + \xi(p^M - p^N)],$$

where the parameter $\xi > 0$ can expand the range of the prices. Prices range from below Nash p^N to above monopoly p^M. For the discrete setting we separate this range into m equally spaced prices, and for the continuous setting we allow the entire range of prices. This is without loss of generality since the real market works on a continuous scale, and the discretization of prices is necessary to employ discrete algorithms.

For our algorithms to have memory, we will store all of the prices from the last k periods of the experiment. Therefore, the state or observation space is

$$s_t = \{p_{t-1}, ..., p_{t-k}\},$$

with p_t representing the vector of all agents' prices in period t.

For almost all experiments, we initialize our parameters in the following way. We create a duopoly ($n = 2$) with marginal cost $c_i = 1$, product quality indices $a_i = 2$, outside good quality $a_0 = 0$, horizontal differentiation $\mu = 0.25$, discrete action space evenly split into $m = 15$, expansion of action space $\xi = 0.1$, and a one-period memory $k = 1$.

For this specification, the Nash equilibrium is $p^N \approx 1.473$ and the monopoly price is $p^M \approx 1.925$.

To explain results and extent of collusion, Calvano et al. [4] use the notion of Δ to quantify average profit gain between p^N and p^M. We define it as

$$\Delta \equiv \frac{\bar{\pi} - \pi^N}{\pi^M - \pi^N},$$

where $\bar{\pi}$ is the average per-firm profit upon convergence, π^N is the profit at the Nash equilibrium p^N, and π^M is the profit under full collusion at the monopoly price p^M. Therefore, $\Delta = 0$ corresponds to the competitive outcome and $\Delta = 1$ corresponds to the perfect collusive outcome. Below is a plot and table of profit gain; note the plot's concavity.

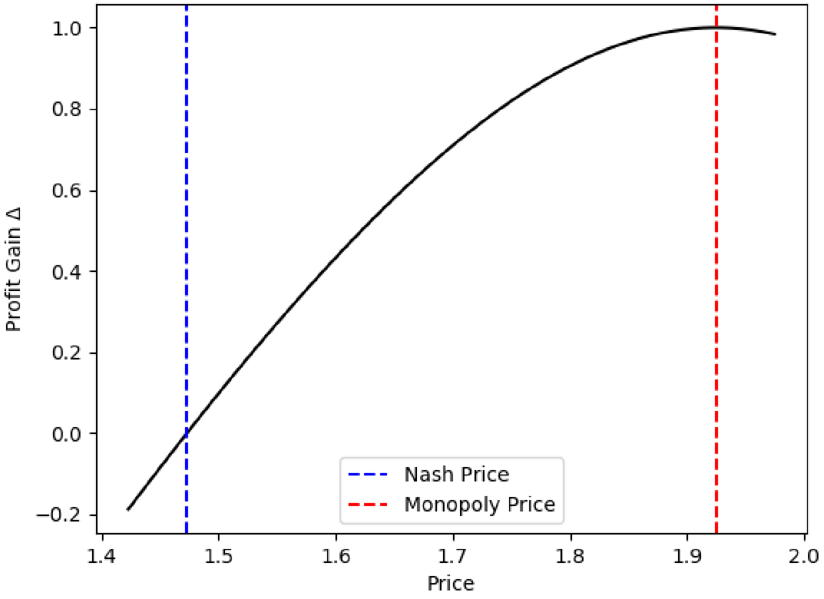


Figure 3.1: Profit gain Δ for specified parameters

Price	Profit	Δ (Profit Gain)
1.45	0.213	-0.085
1.5	0.234	0.098
1.55	0.254	0.271
1.6	0.272	0.433
1.65	0.289	0.580
1.7	0.304	0.709
1.75	0.317	0.819
1.8	0.327	0.905
1.85	0.333	0.965
1.9	0.337	0.996
1.95	0.337	0.996

Table 3.1: Profit gain Δ

3.3 Replication of Results

3.3.1 Q-learning

First, we replicated the results of Calvano et al. [4] by programming the Q-learning algorithm from scratch in Python. In this setting, each agent is performing Q-learning independently of one another, and tabulates their own respective Q-table and values, which we will explain soon. Q-learning is a model-free reinforcement learning algorithm that is considered a simple and efficient way to solve RL problems. It is an off-policy algorithm, so it maximizes across potential next actions instead of following a policy and updating after, as with an on-policy algorithm.

The Q-learning algorithm is as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)),$$

where Q is the Q-table for determining optimal actions, α is the learning rate, t denotes the timestep/period, r_t is the reward, and γ is the discount factor.

For deciding how to take the next action, we use the *epsilon-greedy algorithm* for Q-learning, with ϵ probability of taking a random action from those available. We define ϵ as

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

where β parameter controls the speed at which randomization decays. When we do not take a random action, we choose the next action in state s_t by taking the argmax over the Q-table as follows,

$$a_{t+1} = \text{argmax}_a Q(s_t, a).$$

With this process defined, we took the following initializations and assumptions from the Calvano et al. [4] paper.

We let discount factor $\gamma = 0.95$.

The learning rate can vary, but we mostly use a value of $\alpha = 0.1$.

The epsilon decay can vary, but we mostly use a value of $\beta = 0.00001 = 1 \times 10^{-5}$.

For Q-learning, we define convergence as requiring each agent to keep the same action for 100k consecutive rounds. This ensures that stability is obtained, unless we see a cycle around the real equilibrium value, since the discretized action space limits the algorithm's expressiveness and ability to be precise. If we do not converge after 5 million repetitions, we stop the experiment.

3.3.2 Results

As Calvano et al. proved, if all agents adopt Q-learning, we observe a significant collusion above the Nash equilibrium, for several different environment specifications. Note that for all plots, we include the actual actions taken as well as their moving average (MA) of window size 1000. This helps gather information about action profile trends, and is especially useful for more advanced algorithms.

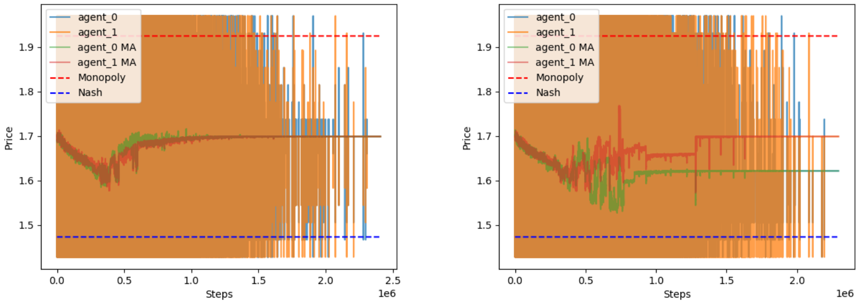
We will broadly define the term *collusive equilibrium* as the agents' collusive behavior leading up to convergence, or for algorithms that show cycles or more complicated strategies, the agents' collusive behavior leading up to the termination of the training run. This definition is purposely general, as we will see that collusion exists in different ways depending on the algorithm.

The collusive equilibrium is defined and measured in two ways: one by how near it is to the monopoly price, and the other by the ability to punish deviations, which can be shown through a price perturbation and the ensuing actions of both agents. Therefore, we refer to the *collusive price* as the price or moving average price in the last few iterations before we end the training run.

Dominating vs. Cooperating Collusive Equilibria

For ease of interpretation, we will define the novel terms of cooperating collusive outcome and dominating collusive outcome. A *cooperating collusive outcome* is one where agents price the same or very nearly the same as each other at their long-run collusive equilibrium; this way, agents are receiving nearly the same reward or profit. In a *dominating collusive outcome*, one agent prices below the other consistently while still significantly above the Nash price, and hence is still colluding. We will define the "dominating" agent as the agent pricing below the other, and the "subordinating" agent as the agent that prices above the other, since in the classical Bertrand setting it is dominant to price below the other player. However, this dominating collusion is still usually mutually beneficial, since the dominating agent is receiving more demand at a lower price while the subordinating agent is receiving less demand at a higher price. A *cyclic collusive outcome* is a collusive equilibrium in which the price of one or more agents cycles between discrete prices, in a pattern of 2 to 4 or more stops. Dominating equilibria mostly occur when we observe a cyclic collusive equilibrium, which we will present later. However, this is not always the case.

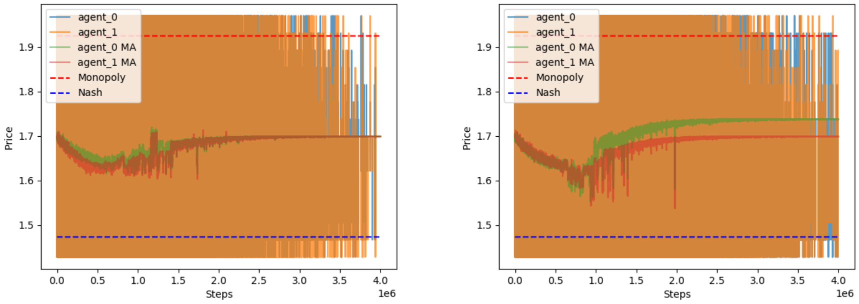
As stated earlier, for this specification of parameters in duopoly ($n = 2$), the Nash equilibrium is $p^N \approx 1.473$ and the monopoly price is $p^M \approx 1.925$, as shown in the figures.



(a) Cooperating collusive equilibrium

(b) Dominating collusive equilibrium

Figure 3.2: Two examples of Q-learning training for 2.5m iterations with $\alpha = 0.1$ and $\beta = 5 \times 10^{-6}$. The moving average prices for each agent are the middle two lines. Here, it takes roughly 2m iterations to converge, with collusive behavior arising near 1m iterations.



(a) Cooperating collusive equilibrium

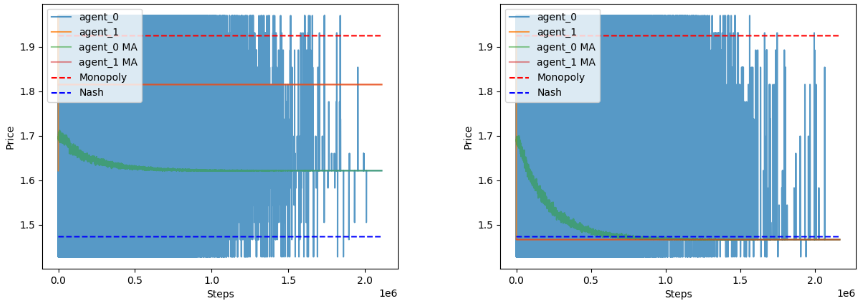
(b) Dominating collusive equilibrium

Figure 3.3: Two examples of Q-learning training for 4m iterations with $\alpha = 0.05$ and $\beta = 2.5 \times 10^{-6}$ (more thorough exploration). The moving average prices for each agent are the middle two lines. Here, it takes roughly 4m iterations to converge, with collusive behavior arising near 2m iterations.

We can see that for these specific parameters, the collusive equilibrium arises after a different number of iterations, since we have a different length of exploration determined by β. The orange and blue lines represent specific actions taken in a time step, while the green and red lines, along with their blended color of brown, represent the moving averages (MA) for the two agents respectively. The Nash equilibrium and monopoly price are represented by blue and red dotted lines respectively. We kept this format throughout experimentation.

As we see here, each training run produces different outcomes, and occasionally the agents do not share the same equilibrium action. Overall across experiments, we observe values of Δ in the range of 60-90%, which mostly aligns with Calvano et al. results, and constitutes a significant increase in price and profit over the Nash equilibrium. We will see that Δ increases with more sophisticated algorithms. In Figure 3.2a, the two agents settle on the same equilibrium, and we see that they punish deviations from this equilibrium. However, in Figure 3.2b, the pair have settled into an equilibrium where agent 0 happens to dominate agent 1, because they have decided this is the optimal move after the exploration period. Punishment upon deviation is still observed, so both are viable collusive equilibria. Figure 3.3 shows the same effect with more thorough exploration and learning parameters for a higher number of iterations, so collusive price is mostly constant across more thorough exploration runs.

As an illustrative example of how dominating collusive equilibria can exist, and to show that our algorithm is performing optimally, observe what happens when we fix one agent's price near the monopoly price and one agent's price at the Nash price. See Figure 3.4.



(a) Fixed at 1.81

(b) Fixed at Nash price

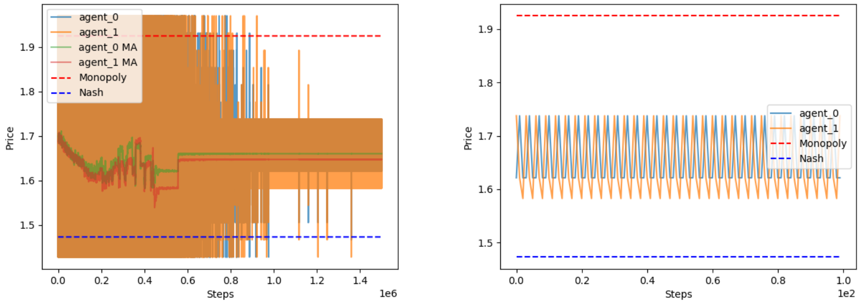
Figure 3.4: Fixing an agent's price, $\alpha = 0.15$ and $\beta = 1 \times 10^{-5}$. When we fix agent 1's price at 1.81, agent 0 learns to optimize profit by pricing near 1.62 after exploration has decayed. This explains why dominating collusive equilibria are still locally optimal, since an agent can potentially maximize profit by neither cooperatively colluding nor pricing the Nash, but instead pricing in between. When we fix agent 1's price at the Nash price, agent 0 learns to price the Nash as well, which, by definition of the Nash, is the optimal solution.

All advanced RL algorithms that I test later exhibit this same behavior. We can see that when fixing the price near the monopoly price, it is optimal to price between the fixed price and the Nash. When fixing the price near the Nash price, it is optimal to head towards the Nash, or the competitive equilibrium. Therefore, a dominating collusive equilibrium may arise before a complete deviation to the Nash price. All of the algorithms behave this way, since once we fix an agent's price, it becomes a single agent reinforcement learning problem, in which the profit for each action does not fluctuate between rounds, unlike in the multi-agent version. Applying this to multiple agents dramatically complicates the problem, and reveals information about how the algorithms work internally and how likely they are to collude.

Cyclic Collusive Equilibria

Because the action space is discrete, agent prices can sometimes "cycle" or "oscillate" as shown in Figure 3.5. We will refer to these collusive equilibria as *cyclic collusive equilibria*, or a collusive equilibrium in which the price of one or more agents cycles between discrete prices, in a pattern of 2 to 4 or more stops. We will refer to the kinds of collusive equilibria we have seen so far as *stationary collusive equilibria*.

An algorithm can cycle because the discrete action space may occasionally lack the exactness necessary to express an optimal collusive price. Cycles do not occur in the continuous setting.



(a) Entire training

(b) Last 100 training steps

Figure 3.5: Cyclic collusive equilibrium, $\alpha = 0.15$ and $\beta = 1 \times 10^{-5}$. The left plot is the entire training run, while the right plot is zoomed in on the last 100 training steps. Note that both agents use a cycle, and trade off which gets more demand or higher price.

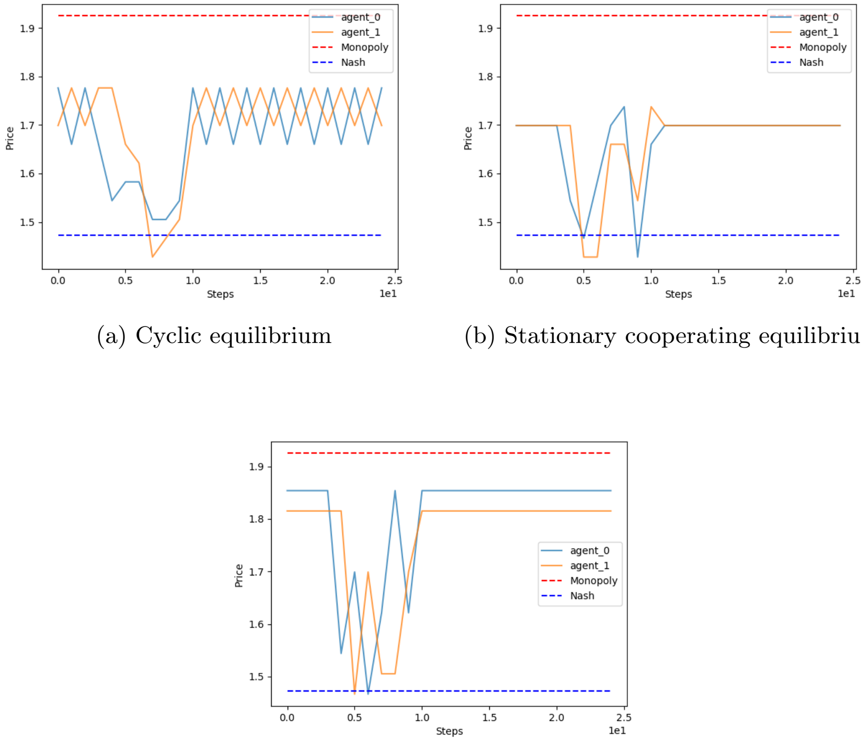
Note the block of orange action at the end of training: this is a cycle at work. We also view a somewhat lower equilibrium because our α and β parameters allowed for less exploration during training.

An important note: much of what we describe as a dominating collusive equilibrium arises from the fact that a cyclic strategy of one agent may have a higher moving average profit or reward than a cyclic strategy of another agent. In some cases, instead of an agent allowing themselves to be dominated by another agent, they are trapped in a complex cycle that happens to favor one agent over another, although it is possible that an agent learns to be dominated since this can maximize reward in certain scenarios. Some cycles are not simply bouncing back and forth between two discrete prices; instead, it often can involve a cycle of mainly 2 to 4 stops, which repeat themselves indefinitely once ϵ has decayed enough to stop random exploration.

In the example above of Figure 3.5a, looking at their equilibrium moving average prices, since the price space is split up evenly into 15 discrete prices, these moving averages cannot and do not represent discrete prices themselves. Instead, they are the moving averages of cyclic patterns.

Punishing Deviations

To show that this behavior is in fact collusion, we can show how the Q-learning algorithm punishes agents that deviate from the collusive equilibrium by perturbation, i.e. forcing one agent to price lower than the collusive price. When we perturb the equilibrium by forcing one agent to deviate, we expect the other agent will "punish" this deviation by slightly undercutting the agent who deviated, slowly increasing the price until they are once again at the same collusive equilibria. This works in dominating and cooperating collusive equilibria, as well as in cyclic equilibria. Without this behavior, we could not accurately state whether collusion was occurring between the agents. Collusion must be upheld by threat of punishment, or else an agent could possibly benefit from deviating in the long run.



(a) Cyclic equilibrium

(b) Stationary cooperating equilibrium

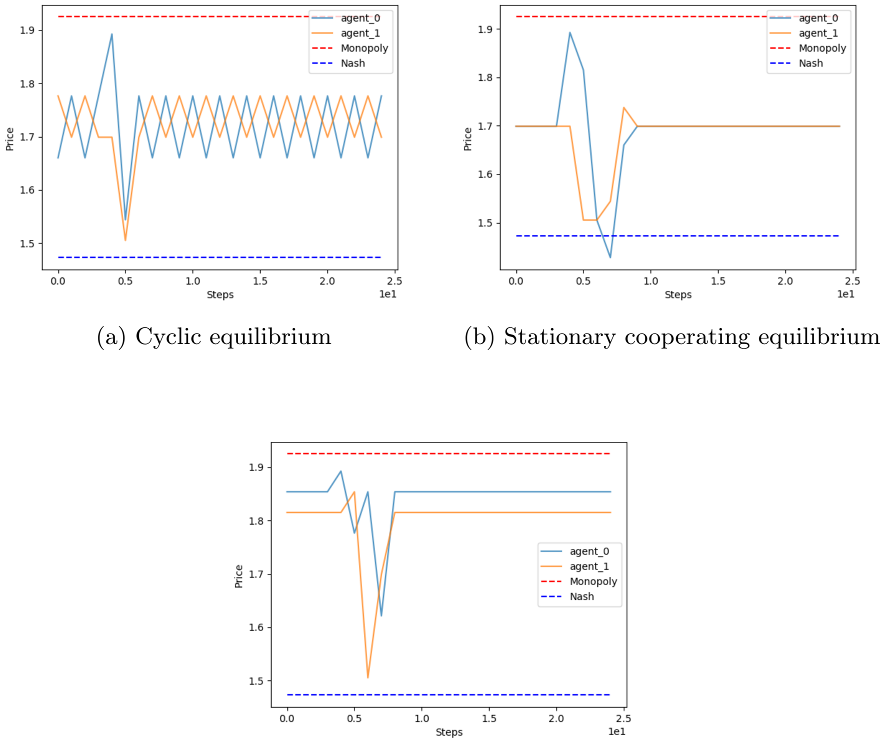
(c) Stationary dominating equilibrium

Figure 3.6: Punishing downward deviations from the collusive equilibrium, $\alpha = 0.1$ and $\beta = 5 \times 10^{-6}$. We notice in the cyclic equilibrium (a) that once agent 0 deviates, agent 1 punishes the deviation by pricing below the subsequent moves of agent 0, and coaxes agent 0 to return to the original equilibrium by staying just below agent 0's prices, subtly punishing on the way up. In both the cooperating (b) and dominating (c) equilibria, we see a similar punishment and return to original collusive equilibrium, but these punishment strategies may take 2 tries as in (b) or display a punishment pattern that is harder to interpret as in (c). These all come from single runs, and punishment strategies change depending on the run.

We force agent 0 to deviate and price low, and keep the other agent constant when we make our perturbation. Note how in Figure 3.6a, agent 1 decreases its price three times to retrieve agent 0, who starts to enjoy a higher profit because of its low price. In Figures 3.6b and 3.6c, we notice a more erratic punishment phase, in which agents punish in less obvious ways and jump between prices before finally returning to the collusive equilibrium. These represent the punishment agent attempting to bring the deviating agent upward, but failing maybe one or two times. These figures also depart slightly from Calvano et al.'s results, which we will touch on later.

When agent 1 decreases price in both scenarios, agent 1 may even price below the Nash to punish agent 0. Then agent 1 stays beneath agent 0 on their way back up, to punish until the original collusive equilibrium is achieved, depriving the other agent of any incentive to stay pricing low.

One interesting result that was studied after the results of Calvano et al. by Abada and Lambin [1] was the behavior of agents when the deviation was *upward*, instead of downward. Since it is dominant to price below the other agent, what would happen if we priced above the collusion? As we mentioned before, it is not necessarily always better to price below the other agent, since an agent can enjoy a higher price with less demand and still make comparable profit. So, would an upward deviation still constitute deviation that is "punishable"? Upward deviations for the same three training runs are shown in Figure 3.7.



(a) Cyclic equilibrium

(b) Stationary cooperating equilibrium

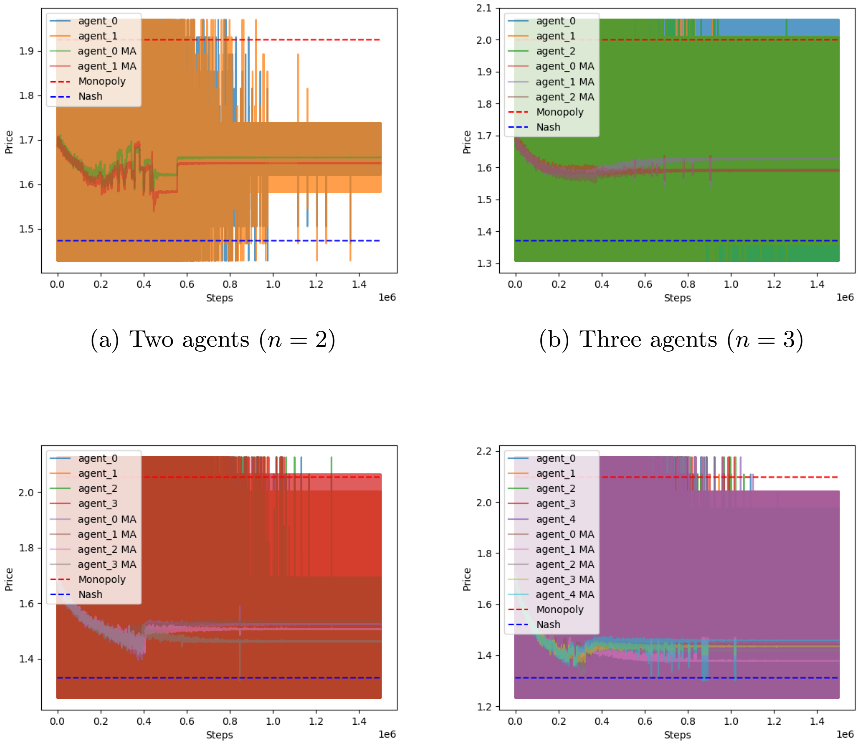
(c) Stationary dominating equilibrium

Figure 3.7: Punishing upward deviations from the collusive equilibrium, $\alpha = 0.1$ and $\beta = 5 \times 10^{-6}$. In each scenario, agent 1 punishes agent 0 by pricing low, which causes a quick return to the original collusive equilibrium. In each instance, agent 0 is quite willing to price low after the upward deviation, most likely because it does not want to suffer from low demand.

In each case, we see agent 1 respond to the upward deviation by pricing low, as they had for the downward deviation. These punishments more quickly result in the original collusive equilibrium, perhaps because the deviating agent does not gain profit from deviating or because most collusive equilibria are already close to the monopoly price. An upward deviation is not advantageous to agent 0 since it is accepting lower demand, unless agent 1 greets it at a new higher collusive equilibrium, which does not happen. Again, some of the punishing strategies are not as clear as others, such as in Figure 3.7b where agent 0 prices below agent 1 as they return to equilibrium, even though agent 0 deviated. However, Figures 3.7a and 3.7c show a clear punishment by agent 1, and then return to equilibrium.

Multiple Agents

We also see that collusion exists for more agents in the market, or $n \geq 2$. However, the collusion equilibrium price approaches the Nash equilibrium as we add more agents, since it becomes harder to communicate through past actions.



(a) Two agents ($n = 2$)

(b) Three agents ($n = 3$)

(c) Four agents ($n = 4$)

(d) Five agents ($n = 5$)

Figure 3.8: Multiple agents' equilibria for $\alpha = 0.15$ and $\beta = 1 \times 10^{-5}$. Notice that as we add more agents, the collusive price becomes lower.

Even though the Nash equilibrium and monopoly price change as we add more agents, the Nash price p^N stays roughly between $1.3 < p^N < 1.5$ and the monopoly price p^M stays roughly between $1.9 < p^M < 2.2$, so the figures are mostly on the same scale. In each, the agents settle on a cyclic equilibrium, and a few agents do not settle on the same equilibrium as others (slightly dominating). As we add more agents, the range of Δ slowly drops off, starting Δ between 60-90% for two agents, to 40-70% for three agents, to 20-40% for four agents, to 5-20% for five agents.

This is an important point of emphasis: as we increase the number of agents, collusion is harder to achieve at a higher price. One potential solution for collusion mitigation is including more agents, which may seem easy to do in a real market. However, this may not always be possible due to lack of firms in a specialized market, so we will investigate alternative ways to mitigate duopoly collusion with methods that apply to markets with more agents as well.

3.3.3 Discrepancies with Calvano et al.

There are a few notable discrepancies between our results and the results of Calvano et al. [4].

One discrepancy is the price at which Calvano et al.'s Q-learning agents learn to collude, which seems to be slightly higher than the collusive prices of our agents. Calvano et al. finds 70-90% Δ across different α and β parameters, which roughly translates to a collusive price from 1.7 to 1.8. When running our Q-learning, we found Δ more similar to 60-90%, which is closer to a collusive price of 1.65 to 1.8. Overall, this is not that notable of a difference, but our Q-learners collude slightly less efficiently across the suite of α and β parameters.

We also notice that our punishments to deviation are much less straightforward than Calvano et al. shows, with more erratic price changes before the original collusive equilibrium is achieved. Even in the scenarios with stationary collusive equilibria, we note complex patterns of punishment.

Another discrepancy is related to the dominating and cooperative collusive outcomes. There is little mention in Calvano et al. [4] of a collusive equilibrium that does not share the same (or similar) price. This may be mentioned subtly as what they call "asymmetric" long-term price; however, their results imply that this is rare, while we see dominating collusive outcomes quite frequently, although cooperative collusive outcomes are still equally as common.

Chapter 4

Application to Advanced RL Algorithms and a Continuous Action Space

In this chapter, we will share results of various RL algorithms on Bertrand competition. The Q-learning algorithm was implemented from scratch in Python, but all other algorithms were implemented using Ray's RLlib library [2].

This section is motivated by the application to the real-world market, in which more advanced RL algorithms could be used on both the discrete and continuous settings. We wanted to analyze whether collusion was specific to Q-learning, or if collusion could be exacerbated by high performance algorithms that make use of neural networks and other modern techniques. By using more than just Q-learning, we also hoped to find what characteristics of Q-learning made it effective for collusive outcomes, and to see whether all RL algorithms could collude as effectively. The extension to the continuous action space is a logical next step in applying the work to a real market, and we must study whether the continuous scale prohibits or encourages collusion, to protect against these algorithms being used in the near future.

First, we will explain the difference between on-policy and off-policy algorithms, to give background on the advanced RL algorithms' results.

4.1 On-Policy vs. Off-Policy Algorithms

Throughout experimentation, we noticed a contrast between on-policy and off-policy algorithms in their ability to detect and engage in collusion. We can further explain the difference between on-policy and off-policy to see why this occurs, inspired by the book *Reinforcement Learning: An Introduction* by Sutton and Barto [10].

4.1.1 On-Policy

On-policy reinforcement learning refers to the update of an existing policy, which is affected by an agent's next step. The policy is continually refined and optimized based on the action that the agent takes in that timestep.

For example, the on-policy equivalent of Q-learning is an algorithm called *SARSA* (State, Action, Reward, State', Action'). SARSA has the same equation as Q-learning except with one minor change.

Q-learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)).$$

SARSA

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)).$$

Instead of updating the Q-table with the best next possible state-action pair Q-value, we adhere to our policy and update based on what reward or value our policy's action produces. This can slightly limit one form of exploration by keeping the algorithm on the path dictated by its policy; therefore, most exploration comes from randomization and initialization, and less later on.

A3C and PPO are on-policy, so they improve their existing policy instead of greedily taking the best action at each step. Because of this process, our RLlib library treats the exploration and evaluation phases separately, i.e. the training and testing phases, with the former consisting of mostly random actions while the latter consists of no new exploration, and only follows the predetermined optimal policy. This results in differences for the collusive performance between A3C/PPO and our off-policy techniques.

4.1.2 Off-Policy

Off-policy reinforcement learning is independent of the agent's action, and can update with the expected result of an action without the agent actually taking that action. An off-policy algorithm greedily takes the value associated with the best expected reward.

Q-learning is the simplest off-policy algorithm, and it updates by taking the value associated with the best action at that timestep. Instead of using $Q(s_{t+1}, a_{t+1})$, it uses the maximum next action $\max_a Q(s_{t+1}, a)$. This allows for a different form of exploration, and can be the difference between an optimal and sub-optimal solution.

Q-learning, DQN, and DDPG, explained below, are all off-policy reinforcement learning algorithms, and they reliably achieve the most collusion when given the opportunity to collude in Bertrand competition.

We believe that off-policy algorithms perform better than on-policy because it incorporates another form of exploration that on-policy algorithms lack. In addition, our off-policy algorithms learn through decaying exploration, which more easily coordinates collusion than the on-policy separated exploration and evaluation approach.

4.2 Advanced RL Algorithms

Of all the algorithms we have tried in Bertrand competition, DQN and DDPG were the best at colluding, and these are both off-policy algorithms like Q-learning. The algorithms below are all implemented using Ray's RLlib [2].

4.2.1 DQN

Deep Q-learning or *Deep Q-Network (DQN)* is similar to Q-learning, but integrated with a neural network to simplify the observations given by the observation space in an attempt to improve performance. This method is off-policy and only takes a discrete action space, like Q-learning.

This technique utilizes experience replay, a mechanism that uses a random sample of prior actions instead of the most recent action to progress to the next step and train, smoothing changes in the distribution of data and removing correlations in the sequence of observations. In addition, the updates during each step adjust the Q-table towards a target value that is only periodically altered, which reduces target correlation further [8].

Exploration in DQN can be tuned by using the same epsilon-greedy technique. We start $\epsilon = 1$ and end epsilon at $\epsilon = 0.000001$, and we tune the parameter for the steps it takes to go from the starting to ending epsilon, which we will call epsilon steps (ϵ-steps).

4.2.2 A3C

Asynchronous Advantage Actor Critic (A3C) is an on-policy, discrete and continuous deep reinforcement learning algorithm [7].

Assume for a moment that we are playing a game with one agent interacting with one environment. A3C is asynchronous because instead of using that single agent and single environment, it uses multiple agents each with their own network parameters and own replication of the environment. Each agent asynchronously trains, and reports to a global network, diversifying the training set.

Our Bertrand competition is a multi-agent environment, so in this scenario, we replicate the environment and have a representative agent in each, e.g. if we have five replicated environments, agent 0 has one copy of itself participating in each environment against the copies of agent 1. The agent 0 copies report to the global network for agent 0, and likewise for agent 1.

The actor-critic portion combines the methods of Value Iteration and Policy Gradients by predicting both the value function and the optimal policy function, using the value function to update the policy. Put simply, the policy is the probabilistic distribution of the action space.

A3C also uses advantage instead of discounted rewards as most policy gradient methods do. The *advantage function* tells the algorithm how much better the rewards are than the expected reward [7].

4.2.3 PPO

Proximal Policy Optimization (PPO) is a policy gradient method for discrete and continuous action spaces with a few added techniques.

PPO is highlighted by its ease of implementation, relatively low sample complexity, and ease of tuning. It computes an update to minimize its cost function while ensuring the deviation from the previous policy is limited. PPO uses *trust region* updates which are compatible with Stochastic Gradient Descent, and simplifies the algorithm by removing KL penalty [9].

4.2.4 DDPG

Deep Deterministic Policy Gradient (DDPG) is an off-policy continuous deep reinforcement learning algorithm.

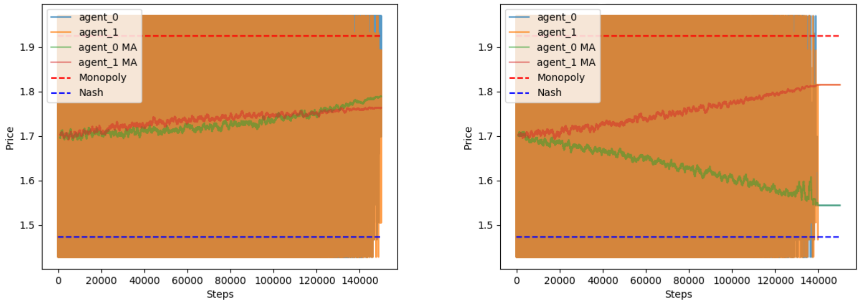
It combines the innovations from Deterministic Policy Gradient (DPG) and Deep Q-Network (DQN) by introducing experience replay and slow-learning target networks from DQN while utilizing the continuous functionality of DPG [6].

4.3 Results

4.3.1 Discrete

We first look at the discrete case of Bertrand competition. We assume $m = 15$ evenly spaced actions throughout.

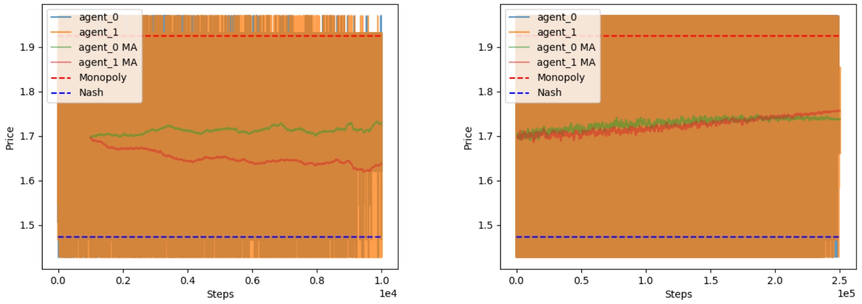
DQN



(a) Cooperating collusion

(b) Dominating collusion

Figure 4.1: Different collusive optimums, both for 150k ϵ-steps. The cooperating collusion (a) shows a high collusive price, while the dominating collusion (b) shows a wide difference in collusive prices of agents, as an example of how divergent the dominating collusion can become.



(a) Collusion, 10k ϵ-steps

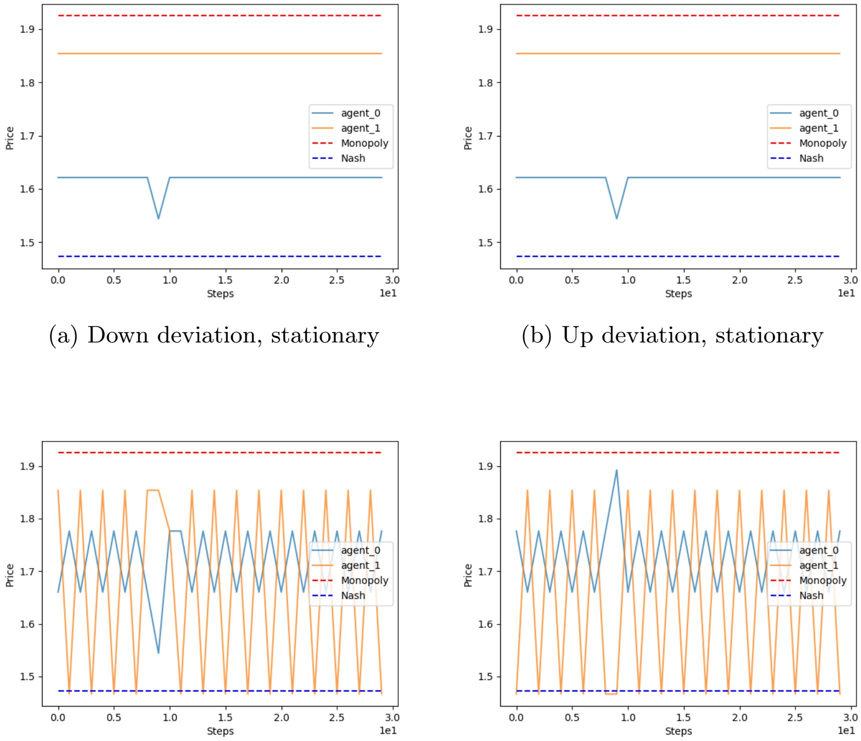
(b) Collusion, 250k ϵ-steps

Figure 4.2: Few vs. many training iterations (or ϵ-steps). An exploration phase that is too quick (10k ϵ-steps) yields a weaker collusion that is not as high (a), while a thorough exploration (250k ϵ-steps) reliably yields a high collusive price.

As we can see in Figure 4.1, these collusive strategies evolved differently given unique randomization in their exploration phases. We most commonly observe collusion that trends toward a Δ of 70-90% across experiments, as shown by Figure 4.1a and 4.2b. Note that the difference in exploration time does not dramatically change this outcome, unless the algorithm is given far too little time to explore, as shown in Figure 4.2a.

Regardless, note that a collusive equilibrium is reached much sooner than in Q-learning. Q-learning takes approximately 1 to 2 million iterations to approach a relatively high (60-90% Δ, or price of 1.65 to 1.8) collusive equilibrium, dependent on the α and β parameters, while DQN takes 150-250k iterations, and achieves on average a higher collusive price (1.75 to 1.8). Just as in Q-learning, we see dominating and cooperating collusive equilibria, and in a much lower number of steps. The performance does not change much when exploration is tuned by ϵ-steps either. Too few ϵ-steps gives a weaker equilibrium shown in Figure 4.2a, but still shows the agents significantly deviated from the Nash equilibrium in only 10k steps. More ϵ-steps strengthens the equilibrium but makes for a similar high equilibrium price compared to the price of 150k ϵ-steps, as shown in Figure 4.2b with 250k ϵ-steps.

Another interesting result of using DQN is its robustness to perturbations in the collusive equilibrium.



(a) Down deviation, stationary

(b) Up deviation, stationary

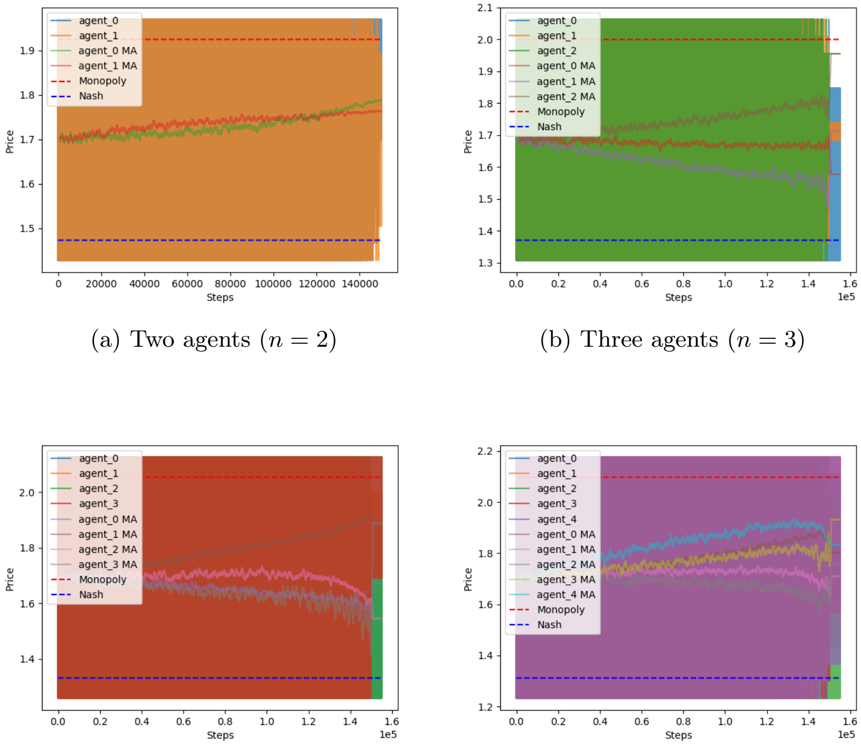
(c) Down deviation, cyclic

(d) Up deviation, cyclic

Figure 4.3: DQN's response to deviation. In all instances, the algorithm immediately returns to the original collusive equilibrium (with the exception of (c), which takes an extra price step).

A return to collusive equilibrium is almost immediate in both cases, stationary or cyclic. Because DQN uses a complex strategy, and because many of the actions appear random (note how for each of Figures 4.1 and 4.2 almost the entire plot is orange), a single or even a few perturbations do not disrupt the high collusive price. Either way, the high price indicates that DQN has a propensity for collusion, and that this collusion may not be so easily disrupted. However, we will see later that DQN can be disrupted by other means.

We also see that DQN colludes for multiple agents.



(a) Two agents ($n = 2$)

(b) Three agents ($n = 3$)

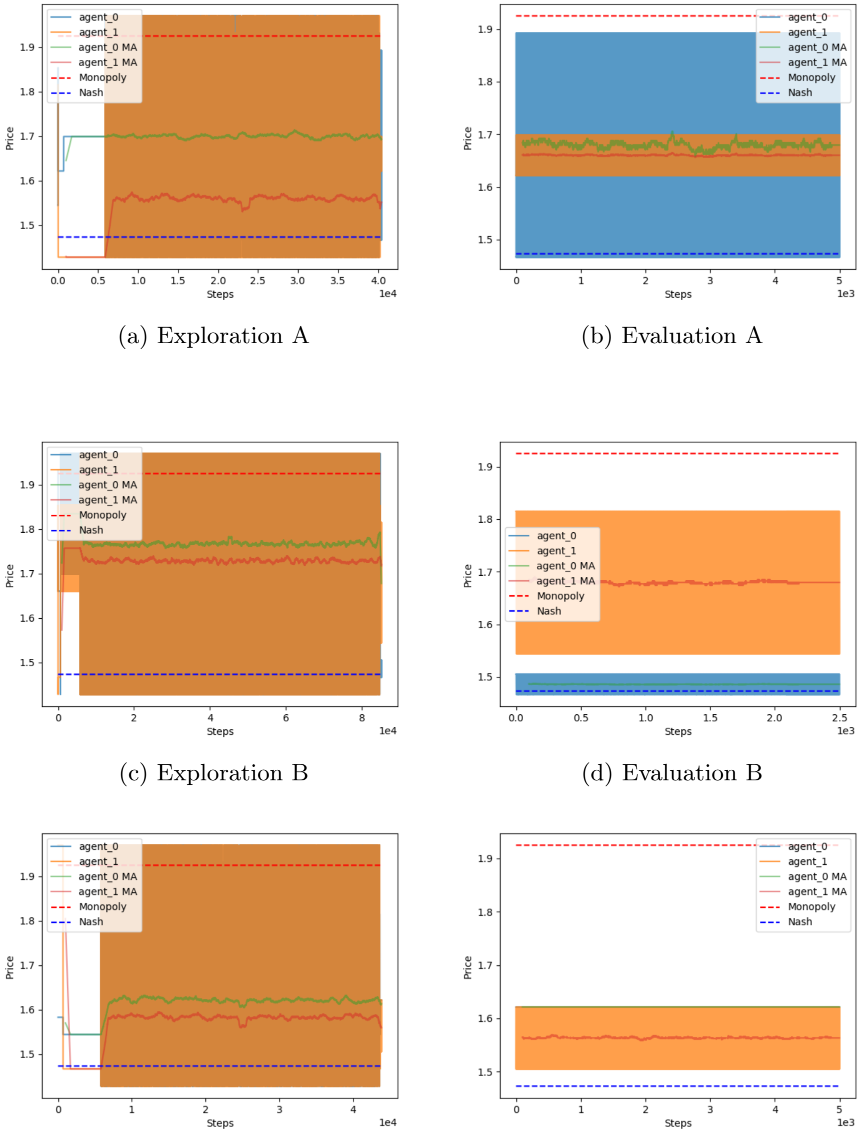
(c) Four agents ($n = 4$)

(d) Five agents ($n = 5$)

Figure 4.4: DQN training for multiple agents. We see that the inclusion of more agents can slightly reduce collusive price; it can also give rise to more divergent dominating equilibria.

A3C

As mentioned before, A3C must train randomly first before it can evaluate. Therefore, agents do not quite learn collusion simultaneously, because exploration does not decay as it does in off-policy techniques, which can lead agents to slowly settle on a collusive equilibrium. Instead, the algorithms only follow the optimal policy in the evaluation phase, which in some cases can create a somewhat collusive equilibrium, but this equilibrium is not upheld by the same principles and deviation punishments as seen in off-policy. We also observe a lower collusive price than we see with off-policy algorithms.



(a) Exploration A

(b) Evaluation A

(c) Exploration B

(d) Evaluation B

(e) Exploration C

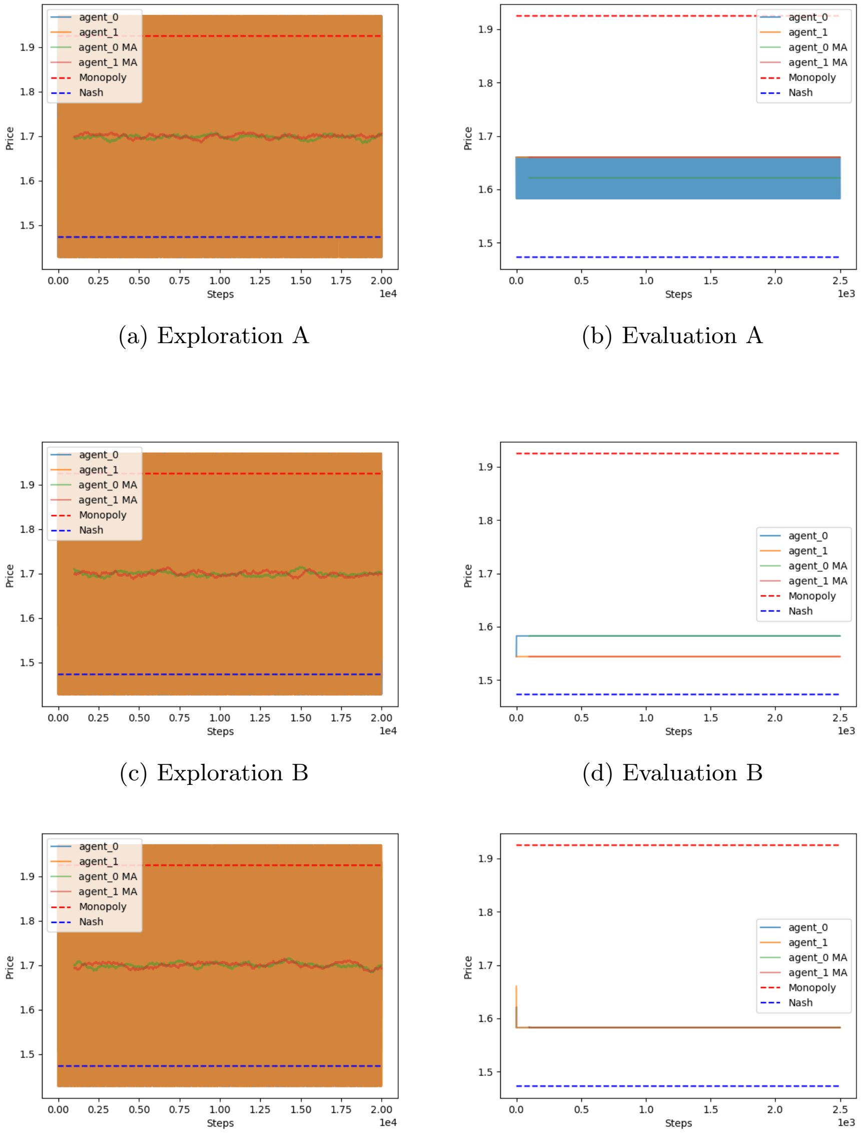
(f) Evaluation C

Figure 4.5: Exploration and evaluation for discrete A3C. Each exploration is mostly random, and there is not much continuity between evaluations, with runs A, B, and C exhibiting variable equilibria. Run A may appear to be colluding, but with run B we see that cooperating prices are not always the case, and run C shows that this collusion is not always with a high collusive price. However, this does not necessarily mean collusion is not occurring at some level.

We sometimes see what appears to be a collusive equilibrium, as with Figures 4.5a and 4.5b, but other runs may create a learned policy which may be too independent to deem as collusive, as shown by Figures 4.5c and 4.5d, where one agent learns to price Nash while the other prices in the middle. Punishment of deviations does not occur.

PPO

Like A3C, PPO must explore separately from evaluation. Interestingly, PPO shows an improvement over A3C in obtaining a collusive equilibrium, despite its separate random training phase. However, this collusion has a low collusive price, and shows a similar weaker ability to punish deviations.



(a) Exploration A

(b) Evaluation A

(c) Exploration B

(d) Evaluation B

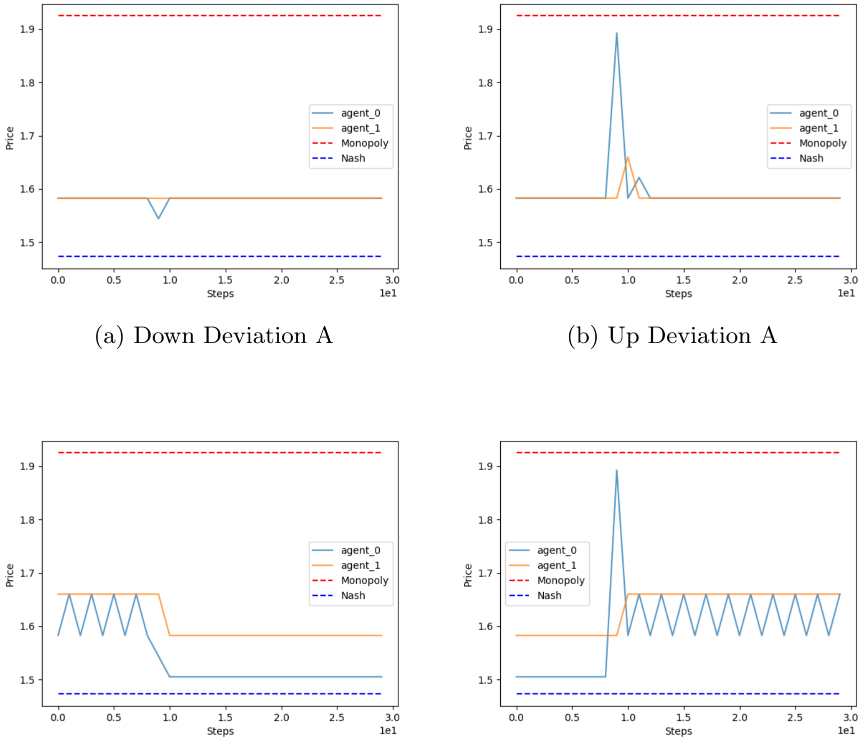
(e) Exploration C

(f) Evaluation C

Figure 4.6: Exploration and evaluation for discrete PPO. Each exploration is random, and evaluations are very similar, with each showing a cooperative equilibrium at similar collusive prices. Run A exhibits a cyclic agent, run B shows a slight dominating equilibrium, and run C shows a cooperating equilibrium.

We see that all three exploration phases look roughly similar, and each evaluation phase learns to collude at around the same price, unlike in A3C.

We can look at deviation punishments to see that these collusive equilibria sometimes return to normal immediately following the optimal policy, but others take another perturbation to return to the original.



(a) Down Deviation A

(b) Up Deviation A

(c) Down Deviation B

(d) Up Deviation B

Figure 4.7: PPO's response to deviation. For run A, we see a similar immediate return to original collusive equilibria as seen in DQN, while in run B, the deviation causes the agents to settle into a new collusive equilibrium, which is only brought to the original collusive equilibrium by a subsequent upward deviation.

In Figures 4.7a and 4.7b, we see a similar deviation behavior to other algorithms, one that almost immediately returns to normal, with slight punishment in the upward deviation. However, in Figures 4.7c and 4.7d, we notice that the deviation brings the agents to a new, lower collusive equilibrium, which is brought back up to the original only after an upward deviation. This signals a weaker collusion, most likely brought about by the separation between exploration and evaluation.

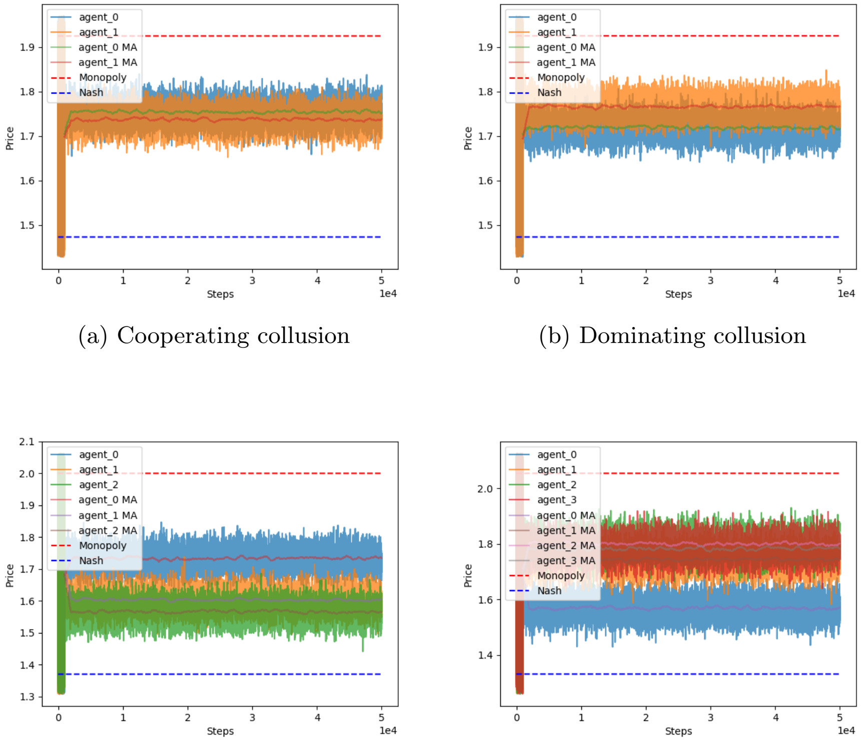
The two on-policy algorithms behave markedly different from one another; this may be due to its difference in asynchronous training or the fact that PPO is more state-of-the-art than A3C, but either way, both on-policy algorithms perform much worse than our off-policy algorithms.

4.3.2 Continuous

In Bertrand competition described by Calvano et al., the price space is discrete for illustrative purposes. However, prices in the real market are continuous, and algorithms used in practice must reflect this. Now we will explore a few continuous algorithms to see if collusion arises.

DDPG

For DDPG, we see significant collusion, one which is as good or better than Q-learning and DQN in the discrete setting. Even with a much more complex observation space, DDPG colludes reliably for several agent and environment settings. This is due to its similarity to DQN, and the fact that it is an off-policy algorithm.



(a) Cooperating collusion

(b) Dominating collusion

(c) Three agents ($n = 3$)

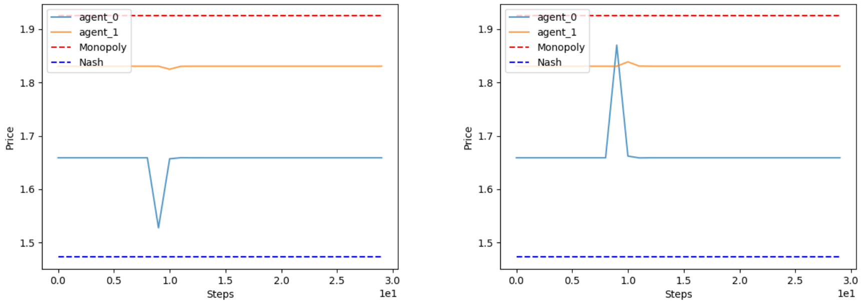
(d) Four agents ($n = 4$)

Figure 4.8: DDPG training for multiple agents. In (a) and (b), we show a cooperating and dominating collusion; note that the dominating collusion is much less divergent than with DQN or even Q-learning. In (c) and (d) we introduce more agents and the collusive prices decrease slightly, but much less than in Q-learning.

In Figures 4.8a and 4.8b, we have two agents $n = 2$ for our environment. The cooperating and dominating equilibrium have similar, high collusive outcomes. We can also see in Figure 4.8a that the actions played are never less than 1.6 or more than 1.8 after an initial exploration period, which is a departure from what we observed in the discrete case. We also observe high equilibrium after only about 2k iterations, which is remarkably fast to coordinate collusion. Runs with two agents observe 70-90% Δ across experiments, whereas runs with more agents show a wide variety of profit depending on the specific agent.

The settings with more than two agents still observe a high collusive equilibrium with relatively more dominating strategies at play; however, this is not always the case. Sometimes, agents collude as cooperatively as Figure 4.8a. An interesting note is that adding more agents does not significantly decrease the collusive price, although a larger range of long-term prices for each agent exists. This could be a product of the continuous price setting allowing more expression in each agent, and that more fine-grained prices allow for agents to communicate more effectively, even if the complexity of the space has become larger when moving from discrete to continuous.

DDPG also quickly returns to its original collusive equilibrium once it observes a deviating agent, in both the downward and upward deviation setting. It only takes one extra round to return to the original in both scenarios.



(a) Down deviation

(b) Up deviation

Figure 4.9: DDPG's response to deviation. Note the slight delay in the return to original collusive equilibrium by the deviating agent; in both, agent 0 gives another price just below (a) or above (b) the original collusive price, to see if this would make for a useful deviation.

This could be because a complex cycle of punishment does not present itself as in Q-learning, since there are no discretized states that lead to a pattern of price increase towards the original collusive equilibrium. Instead, this is a very quick and less aggressive punishing strategy, as shown by the punishing agent's behavior to move slightly in the direction of the deviation after one time step. The deviating agent remains slightly in the direction of the deviation after its initial large deviation, but after viewing the punishing agent's response, decides to return to the original collusive price. DDPG knows that the optimal strategy is what it was doing originally, so it changes accordingly.

We use DDPG as our representative continuous algorithm in later sections. DDPG is an algorithm that can be implemented relatively easily in real-world pricing scenarios, and this is why we must pay attention to its collusion properties.

A3C and PPO

A3C and PPO can also take a continuous action space [7]. However, these algorithms learned to price quite low, which was hard to classify as collusion or just misunderstanding of the environment. Further research could be carried out with A3C and PPO in the continuous setting.

4.3.3 Summary

We have noticed several trends for algorithms that achieve the highest performing collusion. First, we noticed that off-policy algorithms outperform on-policy algorithms, partially because off-policy algorithms utilize an alternative form of exploration that on-policy algorithms lack, but also because our implementation of off-policy algorithms incorporates a decaying exploration rate that coordinates collusion from initialization through evaluation. Our on-policy implementations require separation of exploration (training) and evaluation (testing), which creates a scenario in which algorithms must pick up on collusive practices from random observation and reward, which obscures communication between agents and leads to lower "collusive" prices. We also noticed our advanced off-policy algorithms see high performing collusion in many fewer convergence steps, indicating that Q-learning is only the tip of the iceberg as far as the capabilities of AI collusive practices.

To summarize the results from above, I have provided Table 4.1. The table describes the state space, convergence, collusive price, general collusive strength measured by ability to punish deviations and frequency of collusive outcomes, and profit gain Δ. The convergence for A3C and PPO is not defined since the exploration length is determined by the user.

Algorithm	State Space	Convergence	Price	Strength	Δ
Q-learning	Discrete	0.75-2m	1.65-1.8	High	60-90%
DQN	Discrete	150-250k	1.7-1.8	High	70-90%
A3C	Discrete	N/A	1.55-1.7	Low	30-70%
PPO	Discrete	N/A	1.55-1.7	High	30-70%
DDPG	Continuous	2k	1.7-1.8	High	70-90%
A3C	Continuous	N/A	1.45	Low	0%
PPO	Continuous	N/A	1.45	Low	0%

Table 4.1: Results of each algorithm. Q-learning, DQN, and DDPG have the strongest and highest collusion. DQN and DDPG learn in significantly fewer steps than Q-learning. A3C and PPO perform slightly worse, with A3C exhibiting lower collusive strength.

Chapter 5

Mechanisms to Hinder Collusion

To interrupt the communication between agents that leads to collusive outcomes, we propose a few methods that can disrupt collusive equilibria while still minimizing the effect on market interference (i.e. meeting demand at a reasonable price, etc.).

The first mechanism consists of introducing a supervisor agent that can boost demand to a specific pricing agent, which can interrupt collusion and communication between pricing agents. The supervisor is rewarded for minimizing the product or sum of the prices. We used this on Q-learning, DQN, and DDPG, and found collusion decreases for each, especially DQN.

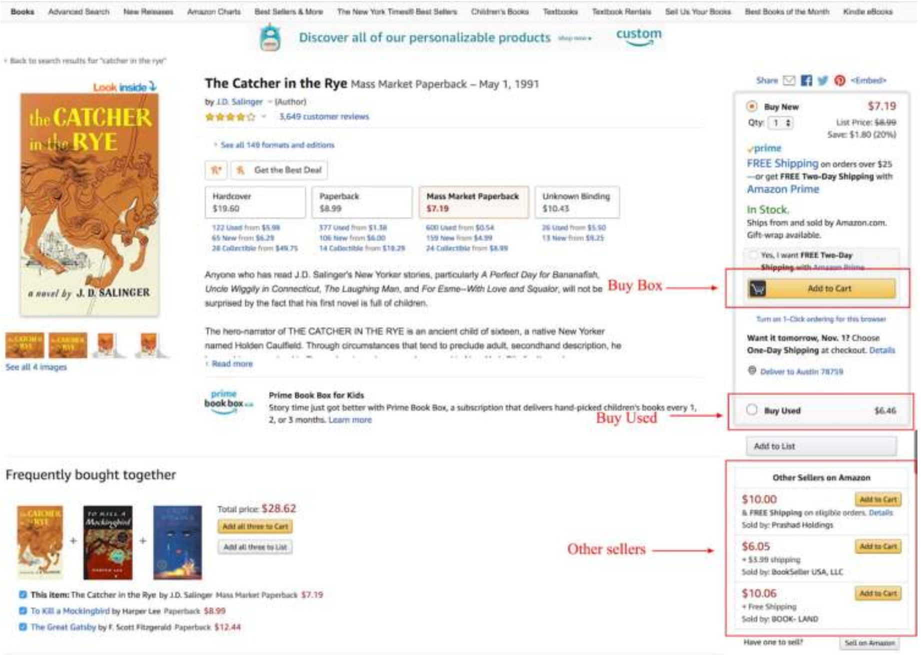
The rest of our mechanisms are environmental changes in which we artificially step prices downward, similar to how an anti-collusive government would interrupt a market. These methods have their own consequences, which we will discuss.

5.1 Supervisor Approach

5.1.1 Overview

A first kind of approach is to introduce another agent, which we called the *supervisor agent.* The supervisor works in a way similar to Amazon's "Buy Box", shown below.

The buy box works to highlight certain vendors of a substitutable good, while sidelining other vendors. In Figure 5.1, we see that the "Add to Cart" feature automatically highlights one firm in particular, and that only upon scrolling down do we see other sellers who may have different prices for the same good. This method works to funnel demand towards a certain vendor, perhaps because they paid Amazon more or they are better vendors to work with. In any case,



The Catcher in the Rye Mass Market Paperback – May 1, 1991

by J.D. Salinger (Author)

3,649 customer reviews

› See all 149 formats and editions

Hardcover	Paperback	Mass Market Paperback	Unknown Binding
$19.60	$8.99	$7.19	$10.43
122 Used from $5.98 65 New from $6.29 28 Collectible from $49.75	377 Used from $1.38 106 New from $6.00 14 Collectible from $18.29	600 Used from $0.54 159 New from $4.99 24 Collectible from $8.99	26 Used from $5.50 13 New from $9.25

Anyone who has read J.D. Salinger's New Yorker stories, particularly *A Perfect Day for Bananafish, Uncle Wiggily in Connecticut, The Laughing Man,* and *For Esme--With Love and Squalor,* will not be surprised by the fact that his first novel is full of children.

The hero-narrator of THE CATCHER IN THE RYE is an ancient child of sixteen, a native New Yorker named Holden Caulfield. Through circumstances that tend to preclude adult, secondhand description, he

‹ Read more

Prime Book Box for Kids
Story time just got better with Prime Book Box, a subscription that delivers hand-picked children's books every 1, 2, or 3 months. Learn more

● **Buy New** $7.19
Qty: 1
List Price: ~~$8.99~~
Save: $1.80 (20%)

prime
FREE Shipping on orders over $25 —or get **FREE Two-Day Shipping** with Amazon Prime

In Stock.
Ships from and sold by Amazon.com.
Gift-wrap available.

□ Yes, I want **FREE Two-Day Shipping** with Amazon Prime

Want it tomorrow, Nov. 1? Choose **One-Day Shipping** at checkout. Details

○ **Buy Used** $6.46

Other Sellers on Amazon

$10.00
& **FREE Shipping** on eligible orders. Details
Sold by: Prashad Holdings

$6.05
+ $3.99 shipping
Sold by: BookSeller USA, LLC

$10.06
+ Free Shipping
Sold by: BOOK- LAND

Frequently bought together

Total price: $28.62

☑ **This item:** The Catcher in the Rye by J.D. Salinger Mass Market Paperback $7.19
☑ To Kill a Mockingbird by Harper Lee Paperback $8.99
☑ The Great Gatsby by F. Scott Fitzgerald Paperback $12.44

Figure 5.1: Amazon "Buy Box" with sellers offering other prices below [5]

we will use this idea to direct more demand towards a certain agent, in the hopes that this will disrupt the balance of a colluding equilibrium among agents.

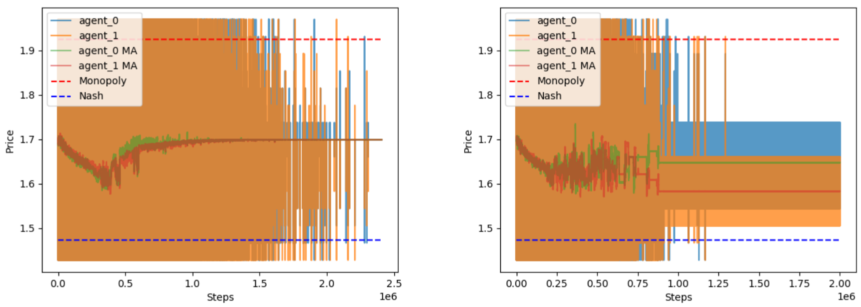
In each experiment, our supervisor agent works with the same reinforcement learning algorithm as the pricing agents. The supervisor agent's action space will be a discrete choice of one of the n agents, and whichever agent is chosen receives a boost to demand by a specific proportional amount, which we call the *proportion boost*, while the other agents lose the corresponding proportion of demand. For example, for a proportion boost of 1.25, if the supervisor agent's action is "agent 0", then agent 0 will receive their demand (calculated from prices of that round and other initialization parameters) multiplied by the proportion boost 1.25, and agent 1 will receive their demand multiplied by 0.75, given by $2 - 1.25 = 0.75$. This method can result in more or less total demand than the demand function dictates, but the difference is negligible.

The supervisor's reward is determined by negating the product (or sum if wanted) of the agents' prices; therefore, a lower total is preferable for the supervisor, so it is essentially rewarded for minimizing the prices of the agents. The supervisor's observation is the set of prices of the other agents, with the same length of memory k. For Q-learning, the supervisor chooses its action after the pricing agents, so it may take their current prices as its observation. For DQN and DDPG, the supervisor must choose its action at the same time as the pricing agents, so it must use the previous prices as its observation like the pricing agents themselves; this is because of limitations of our library.

We will now observe how our algorithms behave with a supervisor agent, specifically testing against our best performing algorithms: Q-learning, DQN, and DDPG. Note that for each, the supervisor agent uses the same algorithm as the pricing agents; therefore, a supervisor for Q-learning agents uses Q-learning, etc.

5.1.2 Q-Learning

When using Q-learning with the supervisor agent, we stagger training so it is possible for the supervisor to take current price observations, instead of making all agents choose an action simultaneously based on the observations of the previous round. We believe a supervisor agent in a real market would have oversight of pricing and agent tendencies before choosing their action, so we made this possible.



(a) Without supervisor

(b) Proportion boost 1.25

Figure 5.2: Q-learning with and without a supervisor agent. We see a decreased collusive price throughout our experimentation, which decreases from 60-90% Δ for normal Q-learning to 30-70% Δ for Q-learning with a supervisor.

We first notice that the RL algorithms are much more unstable, as shown in Figure 5.2b; algorithms have a moving average that varies quite dramatically when exploration is decaying at a certain rate, and the algorithm's behavior does not settle down until they stop exploring. This suggests that collusion is not as sustainable at this level.

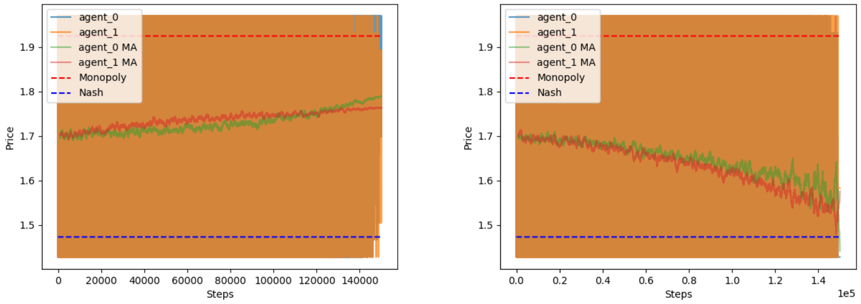
We notice the equilibrium value at the end of training decreases a little between the original and supervisor training runs, so agents are still learning that pricing higher is better than the Nash equilibrium. Whereas we see 60-90% Δ before, we see 30-70% Δ when using the supervisor mechanism.

Overall, this decrease in collusion was relatively successful, with less sustainable collusion and a substantial Δ decrease.

5.1.3 Advanced Algorithms

Because of the limitations of the RLlib library and the difficulty to stagger rewards, for these advanced algorithms we made pricing and supervisor agents choose their actions simultaneously. When using this simultaneous method for Q-learning, the results did not change, so we assume this is a fair simplification to make.

In the case of DQN, we see a clear disruption of collusion, in both the stability of collusion and decrease in equilibrium value.



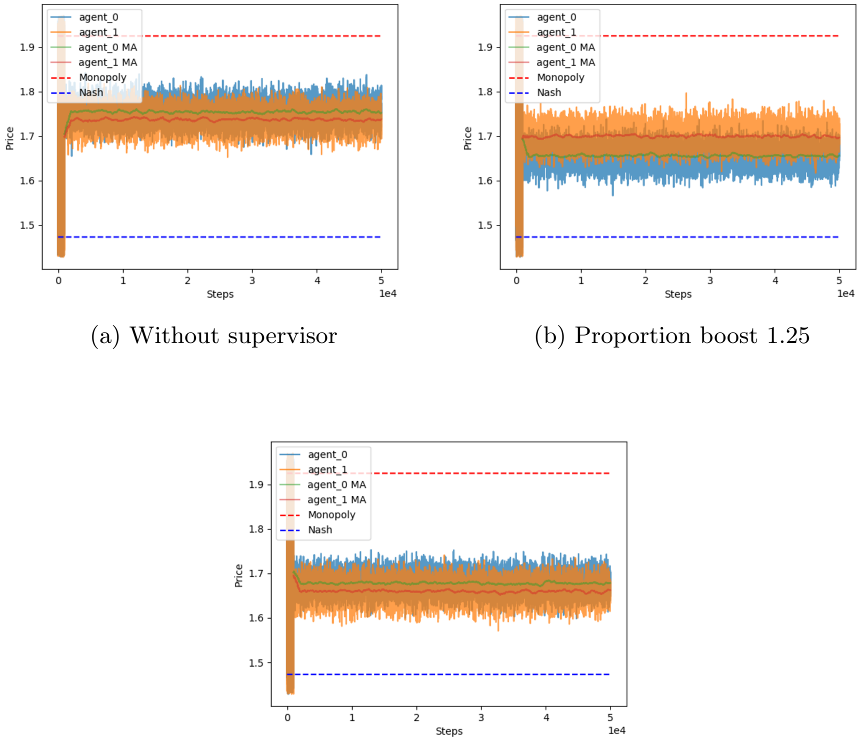
(a) Without supervisor

(b) Proportion boost 1.25

Figure 5.3: DQN with and without a supervisor agent. We observe a significant drop-off in collusive price across experimentation, which decreases from 70-90% Δ for normal DQN to 0-40% Δ for DQN with a supervisor.

DQN performs very well when measuring strength and persistence of collusion, but the supervisor agent disrupts this equilibrium quite effectively, as shown in Figure 5.3b. We record profit gain decrease from 70-90% Δ to 0-40% Δ across runs. When the supervisor intentionally tries to bring down the equilibrium price by choosing which agent gets a demand boost, the agents are not rewarded as much as they anticipated for colluding cooperatively, and continue to search the action space or deviate on other agents more frequently. The balance of collusion here is fragile, and if an agent deems the collusion too unstable or not as rewarding as deviation, they will head towards the Nash equilibrium.

For DDPG, we see that a supervisor agent can decrease equilibrium price, but overall the supervisor does not affect the collusion as much, as shown in Figure 5.4. Because DDPG is a continuous algorithm, and choosing an agent to give a price boost is a discrete action, we give the DDPG supervisor agent the continuous action space between 0 to n, and floor the output to choose the agent. For example, if the DDPG supervisor agent's action is 1.34, we floor this to 1, which applies the proportion boost to agent 1. This may make the relationship between action and reward harder to discern for DDPG, which also contributes to the result below.



(a) Without supervisor

(b) Proportion boost 1.25

(c) Proportion boost 1.75

Figure 5.4: DDPG with and without a supervisor agent. We observe a decrease in collusive price when introducing the supervisor agent, from 70-90% Δ for normal DDPG to 40-70% Δ for DDPG with a supervisor.

DDPG is one of the strongest and most reliably collusive algorithms, so it is somewhat unsurprising that it will still perform well under a supervisor. However, it does drop off decently across runs, from 70-90% Δ to 40-70% Δ. The DDPG mechanism may not rely on fragile or intricate collusive mutual understandings, and instead calculates a new collusive equilibrium price given that the supervisor disrupts demand.

5.1.4 Discussion

It is quite difficult to discern why these algorithms respond differently to supervisor agent integration. We see that DQN's collusive price decreases by more than Q-learning and DDPG, but all decrease notably. One possible explanation is the rate of exploration. DQN explores using the epsilon-greedy approach, but it anneals its ϵ parameter more constantly than the other algorithms. When

Q-learning and DDPG explore at an exponentially decaying rate, they find a collusive equilibrium given the change in demand boosts dictated by the supervisor, and stay on that decreased collusive equilibrium, but this does not decrease further. Meanwhile, with DQN exploring at a more constant rate, its new collusive equilibrium is continually disrupted further, repeating the cycle until it is at or near the Nash price. Essentially, because DQN explores in a different manner than Q-learning and DDPG, it is disrupted by the supervisor agent more constantly, even though it can reach high collusive prices without the supervisor present.

For negligible to little market interference, measured by the loss or excess of demand and the subsequent alteration of the market, we find that collusion can be mitigated quite dramatically by the supervisor agent, utilizing AI working against the collusion of another AI.

Another possible supervisor mechanism could alter the prices themselves, instead of the demand to each agent. This way, the supervisor could learn the Nash price without it being explicitly told, and collusion can be disrupted by the loss of communication through the price history. However, certain limits must be set to stop the supervisor from simply charging the lowest price. Further research could explore this alternative supervisor method.

5.2 Environment Changes

Inspired by different auction tweaks, we have also tested a second set of methods that alter reported prices to impede collusion, which we describe below. Some of these methods are more realistic than others, but observing the algorithms' responses can be helpful hints as to how they adapt and work.

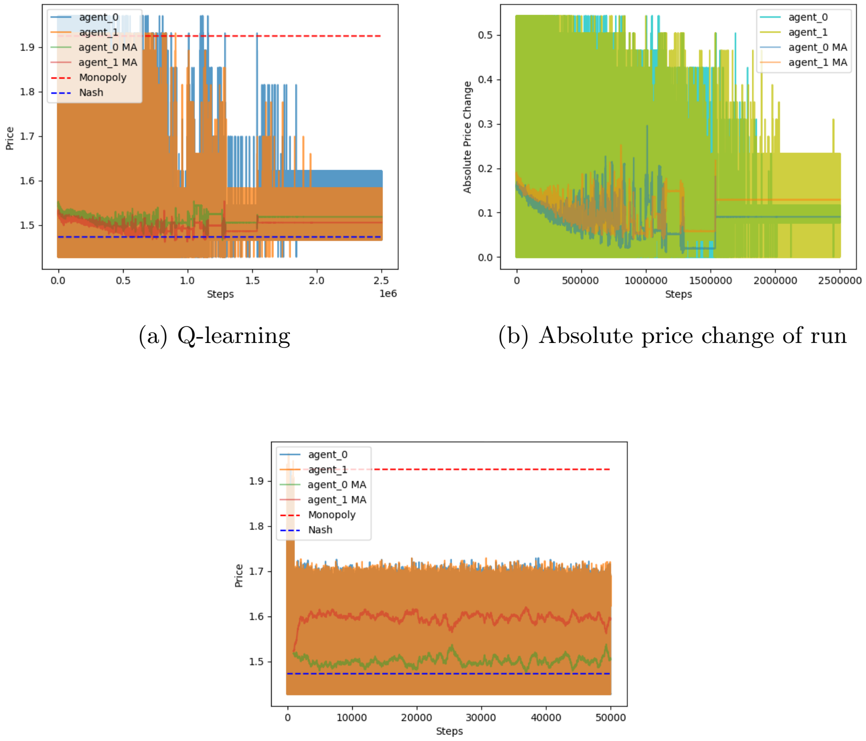
Collusion is already an inefficiency of the market; a market should trade at the competitive price, or in our case the Nash price. This is where demand and supply meet, set by the consumers and firms respectively. However, collusion detracts from fair and free trade, and gives firms more profit while taking from consumers. To measure the extent of market interference by the environmental changes, we define *absolute price change* as the difference between the original price set by each agent and the new price set by the mechanism for each agent. We can measure how much our mechanism is interfering in the market by plotting this absolute price change for each training run.

5.2.1 Downward Price Step

In one scenario, we make the first, higher price equal to the second, lower price, and force the second price to be just above the marginal cost or Nash price. In the case of a tie, the first agent's price is set to just above the marginal cost.

Let $p_{0,t}$ be the price of agent 0 in period t, and let $p_{1,t}$ be the price of agent 1 in period t. So, if the prices reported are $p_{0,t} = 1.4$ and $p_{1,t} = 1.6$, then for a marginal price or Nash equilibrium price p^N of 1.0, we force the lower price to p^N, and make the higher price equal to the lower price, so $p_{0,t}$ goes from

$1.4 \rightarrow 1.0$ and $p_{1,t}$ goes from $1.6 \rightarrow 1.4$. This scenario mimics interference in a free market by anti-collusive government laws or mandates, which benefits the consumers over the firms by lowering prices to around the marginal cost.



(a) Q-learning

(b) Absolute price change of run

(c) DDPG

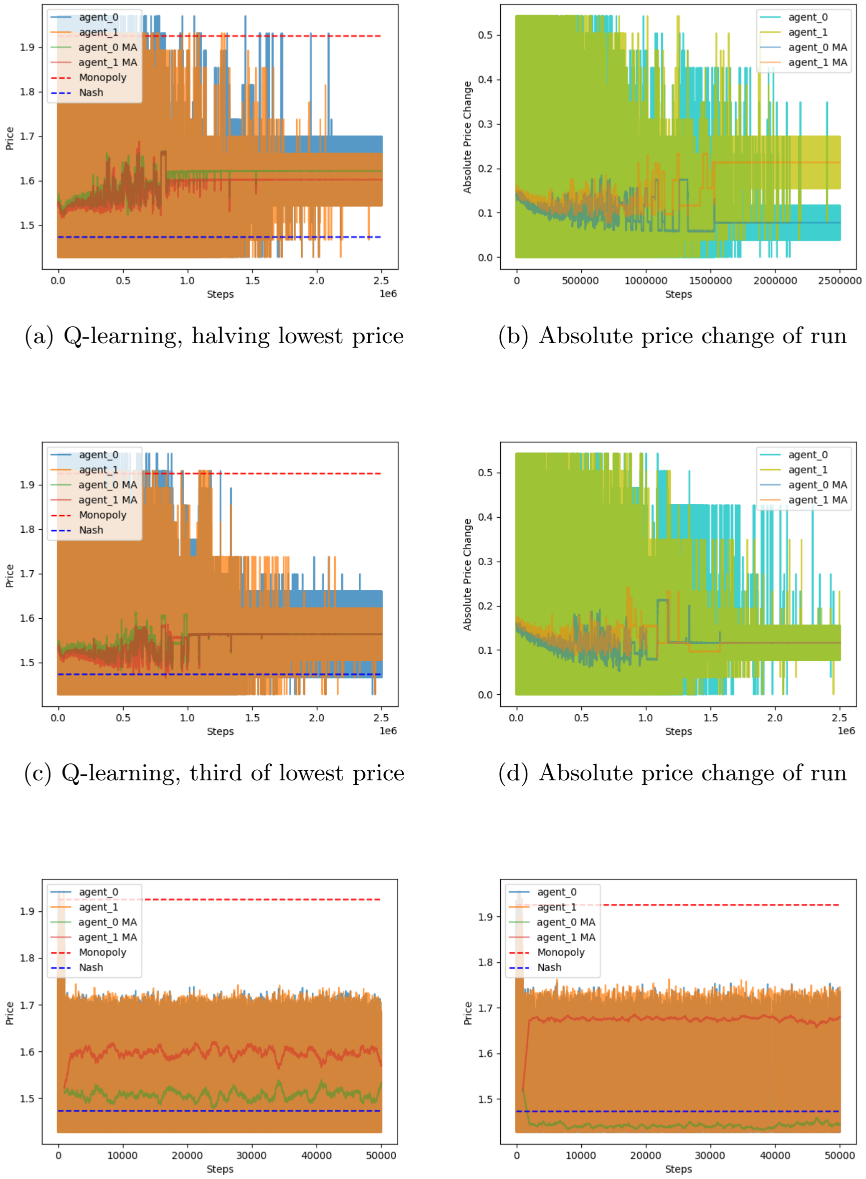
Figure 5.5: Downward price step anti-collusive mechanism. We observe that both agents quickly learn to price near the Nash, since whichever agent has the lower price will receive much more demand if the other agent has a higher price. We include the absolute price change as a measure of interference in the market by showing how much the mechanism alters the prices of both agents.

Proceeding with this method would constitute a substantial market interference, as prices are not set as requested and firms are mostly being forced to price low. We measure the level of interference by the plot of absolute price change, which stabilizes at around 0.1 per round near the end of training.

Whichever agent prices higher will lose much of the demand in each round, since the lower agent will receive almost all demand because of its undercutting of the market. Therefore, the agents quickly learn to price low, and they behave by pricing near the Nash equilibrium, as shown with Q-learning in Figure 5.5a. However, in the continuous setting with DDPG in Figure 5.5c, one agent still prefers to price much higher than the other, perhaps because it enjoys an advantageous combination of decent demand for a higher price. The continuous setting allows for agents to find these more fine-grained equilibrium solutions. This DDPG run is a dominating equilibrium, but it is more exaggerated by the price mechanism; for most rounds, agent 0 prices below agent 1, and this becomes stable after some time.

5.2.2 Fractional Decrease

Another similar method is to set the highest price to the second highest price, and then half or third the second price's index in the discrete action space. So, when our agents price $p_{0,t} = 1.6$ and $p_{1,t} = 1.8$, then $p_{1,t}$ goes from $1.8 \to 1.6$ and the other becomes $p_{0,t} = \frac{1.6-1.0}{2} + 1 = 1.3$, or shown by mapping discrete action to price, discrete action 6 is made into discrete action 3 as $\frac{6}{2} = 3 \to 1.3 = p_{0,t}$, assuming the discrete actions were $m = 10$ from \$1 to \$2. This reduces equilibrium value as shown in Figure 5.6, but not by as much, to keep more profit for the firms.



(a) Q-learning, halving lowest price

(b) Absolute price change of run

(c) Q-learning, third of lowest price

(d) Absolute price change of run

(e) DDPG, halving lowest price

(f) DDPG, third of lowest price

Figure 5.6: Fractional decrease anti-collusive mechanism. We see a less dramatic Q-learning collusive price decrease in (a) and (c) when using the fractional decrease instead of the downward price step. We show the absolute price change done by the mechanism over time. DDPG shows similar behavior, but with a more divergent dominating equilibrium.

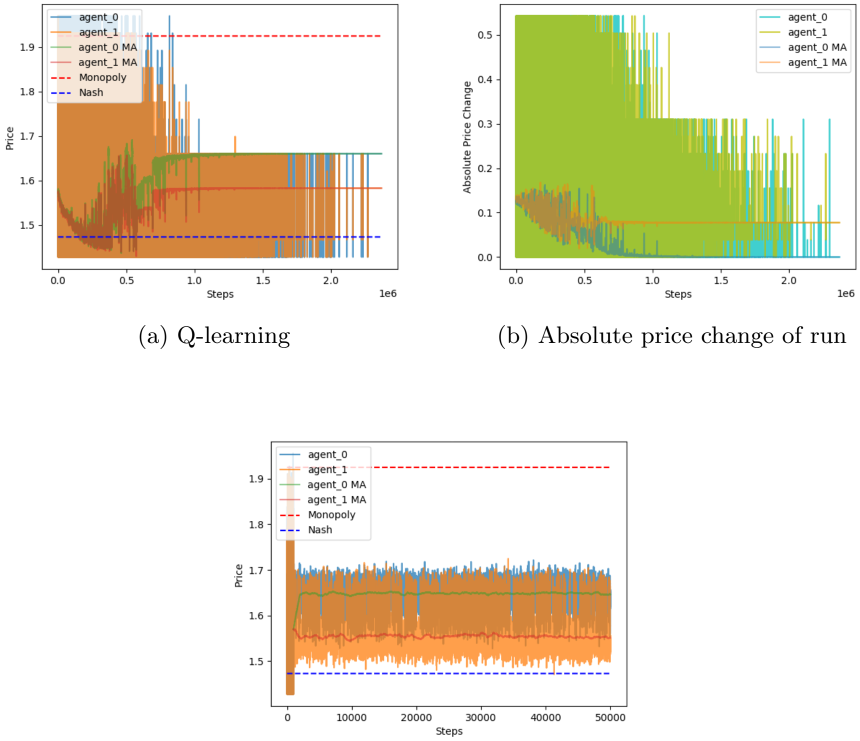
We see that this method has a less dramatic effect on bringing down price as the previous method. The absolute price change by the mechanism stays around 0.1 to 0.2. Cycles arise when agents trade off being the lower pricing agent, and this can be mutually beneficial. For Q-learning, dividing by more (half, third, fourth, etc.) brings the prices down more, but for DDPG, dividing by more seems to exaggerate the dominating equilibrium. This may be due to the fact that if we divide by too much, agent 0 will price below Nash equilibrium, which changes the environment.

5.2.3 Constant Decrease with Original Demand

Interestingly, when we tested making the first price into the second price and the second price into a constant decrease below the second price, such as a 0.1 decrease, we did not observe a lower collusive price, and in fact observed a mid-to-high dominating collusive price. This was due to the mechanism forcing agents to price very close to one another, aiding in collusion and communication.

Therefore, another method is to calculate demand based on the initial quoted prices, and keep the same demand for each agent while we make the first price into the second price and the second price into a constant decrease below the second price. When we changed prices in the above mechanisms, we also changed demand since demand was calculated as a function of each agent's price.

Using our original demand for each agent, for example, with $p_{0,t} = 1.5$ and $p_{1,t} = 1.7$, this would become $p_{1,t} = 1.7 \rightarrow 1.5$ and $p_{0,t} = 1.5 - 0.2 = 1.3$ if we went down the constant 2 discrete actions or the continuous 0.2 price decrease.



(a) Q-learning

(b) Absolute price change of run

(c) DDPG

Figure 5.7: Constant decrease with original demand anti-collusive mechanism. We include absolute price change across iterations as an estimate of interference by the mechanism. Our (a) shows an erratic exploration phase, and a decreased collusive price. Our DDPG shows a decreased collusive price as well.

This makes for a particularly wily training period in Q-learning for Figure 5.7a, and decreases equilibrium price by a decent amount. After a certain point, we see that agent 0 has hardly any price change, while agent 1 has a constant amount just below 0.1. For our experimentation, in the discrete case we decrease the second price by 2 discrete actions, and in the continuous case we decrease by the value of 0.1. Because the Q-learning run shows quite erratic training, it is indicative of a weaker collusion, one that does not punish deviations back toward equilibrium. Again, we see a dominating strategy in DDPG at a low price as well.

5.2.4 Discussion

These three methods yield different results of collusion based on their level of disruption.

The downward price step forces the second price downward, which is the most aggressive but most effective tactic of lowering the collusive price, and this collusive equilibrium is quite stable. Its plot of absolute price change shows the most market interference, but the absolute price change is still comparable to the later mechanisms. The fractional decrease can lower collusive price by more depending on the fractional scale, which can be tuned. This also has a relatively stable collusive equilibrium. The constant decrease with original demand decreased collusive price, but also resulted in a much more erratic exploration phase, and therefore a less sustainable collusive equilibrium, which we also noticed when testing punishment of deviations.

There is no best environmental change mechanism; however, as a general rule, the more invasive methods usually yield lower collusive price. The downward price step shows the most absolute price change, while the constant decrease with original demand shows the least, and this reflects how much the collusive price was lowered by each mechanism. Therefore, markets must choose a method to fit their needs.

Chapter 6

Conclusion

The use of reinforcement learning algorithms on pricing markets has already begun, and it is paramount to make sure these markets do not descend into monopolistic and exploitative practices. We have found that many algorithms can learn to collude, and almost none price the Nash equilibrium or competitive price. However, we have also found that there are ways to disrupt this collusion and high collusive pricing, with some methods exhibiting more market interference than others.

We have verified the Calvano et al. [4] results starting from scratch in Python, and have further tested them with more advanced reinforcement learning algorithms. We have extended Bertrand competition into the continuous setting, and found that many of the same collusive relationships still apply, with collusion potentially even stronger and more exploitative in this setting.

We have seen that there is a discrepancy between on-policy and off-policy RL algorithms in their ability to produce collusive outcomes, and have opened up the possibility to study this phenomenon more thoroughly. On-policy algorithms may lack an essential form of exploration that precludes them from collusion, while off-policy algorithms seem to collude across the board.

We found that mitigating or disrupting collusion is not easy to accomplish, especially without significantly interfering in the market. By using a supervisor agent “buy box” system, we can disrupt collusive behavior. For all algorithms, the supervisor strategy can bring down both collusive behavior and equilibrium price notably. With anti-collusive measures, we can bring agents to price the Nash, but this comes at the expense of free trade and resembles government control of the market.

All areas could be subjects of further research, and each will significantly impact the pricing market of goods for the future. There is still much to be done for this problem, and it is clear to see the massive economic implications of the optimal solution, once it is achieved.

Bibliography

[1] Ibrahim Abada and Xavier Lambin. Artificial intelligence: Can seemingly collusive outcomes be avoided?, February 2020. ENGIE Impact, Grenoble Ecole de Management.

[2] RISE Lab at UC Berkeley. Rllib.

[3] Joseph Louis François Bertrand. Book review of "Theorie mathematique de la richesse sociale and of recherches sur les principles mathematiques de la theorie des richesses". *Journal de Savants*, 67(1):499–508, 1883.

[4] Emilio Calvano, Giacomo Calzolari, Vincenzo Denicolo, and Sergio Pastorello. Artificial intelligence, algorithmic pricing, and collusion. *American Economic Review*, 110(10):3267–3297, 2020.

[5] Amazon.com Inc. Amazon buy box, 2021.

[6] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019. ICLR 2016.

[7] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016. ICML 2016.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. NIPS Deep Learning Workshop 2013.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. OpenAI.

[10] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, Cambridge, Massachusetts, 2017.

[11] Wikipedia contributors. Bertrand competition — Wikipedia, the free encyclopedia, 2020. [Online; accessed 14-March-2021].