Q-Learning algorithms in a Hotelling model

Lucila Porto*

Universidad de San Andrés

2022-08-29

Abstract

What if Q-Learning algorithms set not only prices but also the degree of differentiation between them? In this paper, I tackle this question by analyzing the competition between two Q-Learning algorithms in a Hotelling setting. I find that most of the simulations converge to a Nash Equilibrium where the algorithms are playing non-competitive strategies. In most simulations, they optimally learn not to differentiate each other and to set a collusive price. An underlying deviation and punishment scheme sustains this implicit agreement. The results are robust to the enlargement of the action space and the introduction of relocalization costs.

Keywords: Algorithmic Collusion, Reinforcement Learning, Q-Learning, Hotelling

JEL Codes: L12, L41, D43, L13, D83, C73

*e-mail: *lporto@udesa.edu.ar*

I am grateful for the valuable discussions with Lucía Quesada. I would also like to thank my family for all the help and support. Special thanks are to my brother Manuel for all the useful comments, advice and for lending his computer when mine was not working. Finally, I would also like to thank Gonzalo Ballestero for his helpful comments. All errors and omissions are my own.

1. INTRODUCTION

More and more people are leaving their decisions in the "hands" of algorithms. What movie should we watch? Let us choose the one suggested by Netflix. What is the best route to the supermarket? Let us follow the directions of Google Maps. However, this paradigm change is not only reduced to consumers' decision-making. Firms' decision-making is also experiencing the same change. What price should I set for the product I am selling on Amazon? Let us set the one suggested by the algorithm. What variety of products to offer? Again, let us let the algorithm determine that. The economic consequences of these last questions motivate this paper. In particular, how competition changes in a market where algorithms set prices and choose the degree of differentiation and whether independent algorithms can reach collusive outcomes.

Algorithmic collusion is one of the main concerns on the antitrust agenda nowadays (CMA, 2018; OECD, 2017; Ohlhausen, 2017). Many studies show that two independent algorithms can reach supra-competitive outcomes in several settings. This paper adds another model setup to this list. Understanding under which market conditions algorithmic collusion could arise is the first step towards designing an algorithmic antitrust regulation. The debate is still open about whether there should be a particular algorithmic antitrust regulation, different respect from the traditional one, and whether the regulation should be ex-ante or ex-post (Calvano et al., 2019; Harrington, 2018; Mehra, 2016). However, this debate is beyond the scope of this paper.

Empirical evidence about pricing algorithms is still scarce. The main reason is that there is no way to identify the sellers that outsource their decision to algorithms from those that made their choices manually. Chen et al., 2016 develops a methodology to identify algorithmic sellers. They found that 2.4% of the sellers from their Amazon dataset[1] delegate their decision to algorithms. Also, the authors compare the cumulative distribution of algorithmic and non-algorithmic sellers and find some differences. For example, algorithmic sellers have a higher probability of winning the Buy Box at all levels of rank sellers except for the first one. This is a significant advantage since the majority of the sales on Amazon go through it. Wieting and Sapi, 2021 implement Chen et al.'s methodology on another platform: *Bol.com*. They also found that a significant proportion of sellers outsource their decisions to algorithms. The autorhs run an econometric analisys and found that algorithmic sellers are more likley to win the Buy Box.

Respect theoretical papers, there is a growing literature that accounts for the competition between algorithms, and the market implications. In Asker et al., 2021; Ballestero, 2022; Calvano et al., 2020; Klein, 2019, the algorithms choose prices. In Calzolari et al., 2021; Kimbrough and Murphy, 2009; Waltman and Kaymak, 2008 the algorithms choose quantities. In Brown and MacKay, 2021 the firms choose between algorithms that set prices with different frequencies. As can be seen, most of them are limited to one variable decision problem. As far as I am aware, this is the first economic paper in this literature where the algorithms choose two variables. In particular, this paper aims to tackle the question: *what if two algorithms not only set prices but also the degree of differentiation between them?* To address this question, I analyze the competition between two algorithms in a Hotelling setting.

Broadly speaking there are two decision-making algorithms generations: the adaptive ones and the learning ones. The first of those has a model in the back-end, and the algorithms try to find their best strategy considering demand estimations and previous actions. Contrary, the learning algorithms are mostly model-free, and the algorithms learn by experience. This paper focuses on learning algorithm, particularly Q-Learning algorithms. The question about the emergence of collusion from the competition between two learning algorithms is interesting. Mainly because those algorithms are designed independently, and they are not even aware of the existence of another algorithm, but they could learn to play collusive strategies.

[1] Their dataset was restricted to bestseller products only. The proportion of algorithmic sellers depends on some threshold values.

In most of the simulations, I found that two Q-Learning algorithms that repeatedly compete in a Hotelling scenario choose not to differentiate and set a price greater than the competitive one. Moreover, the strategy that each algorithm follows is, in most cases, the best response to the strategy of its rival. In other words, they are playing a Nash Equilibrium. Finally, I probe the existence of a deviation-punishment scheme. So, if one algorithm breaks the implicit agreement by undercutting its rival's price, the other algorithm punishes the cheater. However, the punishment is finite in time. So, after some iterations, both algorithms return to set a similar supra-competitive price to the one set before the deviation. Summing up, the repeated competition and the existence of a deviation and punishment scheme reduce the price competition in the Hotelling one-shot game and lead to an equilibrium with minimum differentiation and supra-competitive prices. All the results are robust to the enlargement of the action space (the algorithms have more locations or prices from which to choose) and the introduction of relocation costs.

Previous papers show that two Q-Learning algorithms can play collusive strategies when they have one decision variable (Calvano et al., 2020; Klein, 2019). This result also holds under uncertainty (Ballestero, 2022; Calzolari et al., 2021). This paper adds a two decision variable setting to this collusive result list. In conclusion, two Q-Learning algorithms can play collusive strategies besides facing a more complex environment. When the setting gets more and more complex, the Q-Learning algorithm faces computational difficulties because of the problem's dimension. In consequence, it is unlikely that actual firms use this type of learning algorithm. However, Q-Learning's transparency and simple parameter interpretation give valuable insights. The paper results have to be seen as a modest step toward a better understanding of algorithmic collusion.

The rest of this paper is organized as follows. Section 2 describes the state of the literature. Section 3 presents the economic environment. Section 4 introduces the Q-Learning algorithm and the pseudocode used in the simulations. Section 5 outlines the baseline parameter configuration. Section 6 describes some theoretical benchmarks equilibriums that can arise in a Hotelling model and the metrics used to evaluate the performance of the algorithms. Section 7 and 8 present the simulation's results and robustness checks. Finally, section 9 concludes.

2. Literature

The possibility of sustaining collusive equilibriums in a Hotelling model is a topic well investigated in the economic literature. The firm's possibility of differentiating each other reduces the severity of the price competition. Also, with flexible product choice or the chance to relocate in the space line, the punishment after a deviation is reduced (Chang, 1992). Also, there is a growing literature on algorithmic decision-making and collusion. However, no paper set a bridge between both literatures. So, this is the first paper that tackles the question of algorithmic collusion in a Hotelling setting.

The present work contributes to the growing literature on algorithmic collusion. In this field, Calvano et al., 2020 find that two Q-Learning algorithms can sustain collusive outcomes in an infinitely repeated Bertrand game with Logit demand. In Klein, 2019 the author finds that two Q-Learning algorithms can also sustain collusive outcomes in an infinitely repeated Maskin and Tirole setting with linear demand. In that sense, collusion is also achievable when the price decision is sequential instead of simultaneous.

Collusion is also robust to the introduction of uncertainty and changes in the decision variable. First, the work of Ballestero, 2022 extended Klein's work and found that the collusion can be sustained even when the firms have stochastic costs. Second, in Calzolari et al., 2021 the authors found that two Q-Learning algorithms can sustain collusive outcomes in an infinitely repeated Green and Porter setting. Here, the algorithms choose quantities simultaneously in an environment with uncertain demand and imperfect observability.

There are also some works with a Hotelling setting. For example, in Sanchez-Cartas and Katsamakas, 2022 the authors find that a Q-Learning algorithm competes with a Particle Swarm Optimization (PSO)[2] algorithm can sustain supra-competitive prices in a Hotelling setting. However, the decision variable of both algorithms is only the price. They are already located over the segment. In particular, they are fully differentiated. Another example is Vainer and Kukacka, 2021. The authors find that two Nash-Q-Learning algorithms learn to play Nash Equilibrium strategy in a Hotelling setting. Like the previous case, the algorithms are already located at the ends of the segments, and they only choose the price. In the robustness checks, the authors find that if the learning algorithm is Q-Learning instead of Nash-Q-Learning, both algorithms learn to choose supra-competitive prices. Nevertheless neither in Sanchez-Cartas and Katsamakas, 2022 and in Vainer and Kukacka, 2021, the authors test the presence of a reward-punishment scheme. Therefore, it is impossible to tell whether the supra-competitive prices are the result of an implicit collusion agreement or pure luck. This work goes beyond both papers introducing the location choice problem, together with the price choice problem, and testing for the existence of a deviation-punishment scheme.

The other difference is the transportation cost. In Sanchez-Cartas and Katsamakas, 2022 and in Vainer and Kukacka, 2021, the function that the authors use is lineal. I use a quadratic function instead. The Hotelling model with lineal transportation costs presents discontinuities in the demand and the profit function. Also, a Subgame Perfect Nash Equilibrium (SPNE) does not exist for all configurations of location and price (d'Aspremont et al., 1979). Q-Learning is a model-free algorithm; consequently, the discontinuities do not necessarily generate a problem for the learning of the algorithms. However, the discontinuities introduce difficulties when comparing the simulations and theoretical results.

The two variable extension in learning algorithms is not new in the economic literature (Dogan & Güner, 2015; Park & Ryu, 2022; Takahashi et al., 2018; Xie & Chen, 2004). However, there is not a unified methodology followed in multi-action reinforcement learning (RL) problems. In some works, each variable decision is considered as an independent RL problem (Wang & Yu, 2016; Xie & Chen, 2004). In this approach, the algorithm aims to find an optimal action rule for each RL problem. For example, Xie and Chen, 2004 proposes a vertical supply chain with a supplier and two horizontal differentiated retailers. Both retailers are modeled as two Q-Learning algorithms that simultaneously choose the retail price and quantity to order from the supplier. An alternative methodology consists in considering all the possible combinations between the actions and treats each one as a primitive action. In this case, the algorithm only needs to find a single optimal action rule (Li et al., 2012; Park & Ryu, 2022). For example, Park and Ryu, 2022 modeled a differentiated duopoly market where two suppliers compete in ethical and transparency levels of the supply chain. Each supplier is a Q-Learning algorithm. In each game stage, the suppliers simultaneously choose a tuple of the ethical and the transparency level.

The present work follows the first approach in which each variable decision is considered by itself. In consequence, the objective of each algorithm is to find one policy rule for the location problem and another for the pricing problem. However, since each decision is not entirely independent, both RL problems are not completely isolated. It is important to note that following the second approach, where the algorithms' actions are the combinations of locations and prices, makes the setup different with respect to the Hotelling model. This is because, by merging both decision moments in only one, there is no update in the information. So, when choosing the price, the algorithms do not know the current locations.

[2]The PSO is an Evolutionary Algorithm originally from the biology and sociology field. The finding of the optimal action reproduces the behavior of a school of fish or a flock of birds where each member helps and provides information to the rest of the members. In coding language, each member is a particle. And in a Hotelling setting, each action that is tested is a particle. Thus, a pair {location, price} is a particle. The idea is that in the first stage, particles are randomly drawn from the action space. From each action, the player gets a reward. In the following iterations, the particles will move toward the optima. After several iterations, the particles will converge to the best action (Tam, 2021)

If it is not explicitly modeled, the algorithms ignore the relationship between the actions. This relationship could give the algorithms valuable information to optimize the exploration or the learning process. Some works exploit this possibility mainly because, in a multi-action environment, the action space's cardinality grows exponentially, so the computation demand is higher (Chandak et al., 2019; Moodley et al., 2019; Wang & Yu, 2016). For example, in Chandak et al., 2019, the authors exploit the similarity across the feedback received after playing an action and reduce the action space into a representative action space. Other works introduce prior knowledge to discard some actions and reduce the cardinality of the action set. In this work, I do not exploit any action relationship. This is because my goal is focused on the model and results interpretation rather than on the efficiency of the algorithm decision-making. So, I want to make the model as transparent as possible.

The present work contributes to the mentioned literature because the application in competition policy of the two variable decision problem is fairly new. Indeed, as far as I am aware, this is the first paper to model an algorithmic decision-making problem with two variables under the lens of competition policy and antitrust.

3. ECONOMIC ENVIRONMENT

Consider a model of spatial competition where firms and consumers are distributed over some geographic space. There are two firms indexed by $i = \{1, 2\}$. The firms compete in an infinitely repeated location-price game. In every stage game, at first, they simultaneously choose a location, and then, knowing both locations, they simultaneously choose a price. The location dimension can be interpreted geographically or as representing some more general characteristics of the good[3].

A mass of consumers is uniformly distributed over a $[0, 1]$ segment. Each consumer will buy at most one unit of the product (he may not consume). He will choose to buy from the firm that generates the highest consumer surplus (CS). In particular, if the consumer is at the point $x \in [0, 1]$ and buys from the firm i, which is located at l_i, he obtains a surplus equal to:

$$CS = u - p_i - f(l_i, x), \tag{1}$$

where u is the gross surplus of consuming one unit of the good, p_i is the price paid, and $f(l_i, x)$ is the disutility of purchasing a product that is not the most preferred. I assume a quadratic utility cost $f(l_i, x) = \tau * (l_i - x)^2$. So, the disutility is increasing in the distance between the firm i and the consumer location x.

The firm's posible locations are restricted to the same $[0, 1]$ segment where the consumers are located. So, in each stage game, the firm i can choose $l_i \in [0, 1]$ and $p_i \in [c_i, u]$[4], where c_i is the firms' marginal cost.

Given the location and price of both firms, there is a consumer who is indifferent about purchasing from either firm. The location of this indifferent consumer is given by:

$$x^* = \frac{l_{right} + l_{left}}{2} + \frac{p_{right} - p_{left}}{2\tau(l_{right} - l_{left})}, \tag{2}$$

where the subindex *right* and *left* refer to the firm on the right and left respectively. x^* divides the segment between consumers at the left, who prefer the leftmost firm, and consumers at the right, who prefer the rightmost firm.

[3]The location-price game can be thaugh as an abstraction of a model of horizontal product differentation. In that game, each location over the segment represent a different variety of a product. The firms, when choosing location, are choosing which variety to offer. The distribution of the consumers over the segment represents the heterogenous preferences over the varieties. In a model of horizontal product differentation, each consumer has a different ranking over the varieties. So, the most prefered variety is not the same for everyone. Also, if two varieties have the same price, then a consumer will buy the variety closer to his adress because it is the more prefered one.

[4]Note that the setting a price below c_i is a dominated strategy by setting any $p_i \in [c_i, u]$ because it will gives you a non-positive payoff. Setting a price above u is also a dominated strategy by setting any $p_i \in [c_i, u]$ because the payoff is always null. That is why the price space is restricted to that particular segment.

It is important to note that, besides both firms charging different prices, x^* does not necessarily lie between the two firms. Moreover, for certain combinations of location and price, the location of the indifferent consumer could fall outside the $[0, 1]$ interval.

Since consumers may prefer not to consume at all, the demand obtained by each firm is not always the entire segment to the right or left of the indifferent consumer. There exist another type of indifferent consumer who is indifferent between consuming and not consuming. This consumer is the one who arises from equating the consumer surplus to zero $CS(x) = 0$. The roots of this polynomial of degree 2 are the locations of this indifferent consumer. The demand for each firm is then defined as:

$$[\text{Correct side of } x^*] \cap [\text{area between the roots of the polynomial } CS(x) = 0]. \tag{3}$$

Figures 2, 3, and 4 display the different possible cases of the firms' demand. If the firms choose the same location, they compete *á la Bertrand*, where the potential demand of the lower-priced firm is the entire segment, and the effective demand of the higher-priced firm is zero. The effective demand of the lower-priced firm is the area between the locations of the consumer indifferent between consuming or not consuming. If the firms also set the same price, they divide the demand evenly.

Based on the location of both firms and the indifferent consumer, there exist three possible cases:

Insert Figure 1 here.

The following figures are examples for each case. In each plot, the blue (red) curve represents the consumer surplus when the consumer buys from firm 1(2).

Case 1.

Insert Figure 2 here.

Case 2. In the left panel, firm 1's demand corresponds to the segment $[0, x^*]$, and firm 2's demand, to the segment $[x^*, 1]$. In the right panel, there is no intersection between both effective demands.

Insert Figure 3 here.

Case 3. In the left panel, $x^* > 1$, so all the consumers purchase from firm 1. The right panel represents the opposite case with $x^* < 0$. There, all the consumers purchase from firm 2.

Insert Figure 4 here.

The demand in each case is defined by:

- **Case 1:**

$$D_{i,t} = \begin{cases} \min\{1, \max\{roots\}\} - \max\{0, \min\{roots\}\}, & p_{it} < p_{jt} \\ 0.5 * [\min\{1, \max\{roots\}\} - \max\{0, \min\{roots\}\}], & p_{it} = p_{jt} \\ 0, & p_{it} > p_{jt} \end{cases}$$

- **Case 2:**

$$D_{i,t} = \begin{cases} \max\{\min\{x^*; \max\{roots\}\} - \max\{0; \min\{roots\}\}; 0\}, & l_i < l_j \\ \max\{\min\{1; \max\{roots\}\} - \max\{x^*; \min\{roots\}\}; 0\}, & l_i > l_j \end{cases}$$

- **Case 3:**

$$D_{i,t} = \begin{cases} \min\{1, \max\{roots\}\} - \max\{0, \min\{roots\}\}, & |l_i - x^*| > |l_j - x^*| \\ 0, & |l_i - x^*| < |l_j - x^*| \end{cases}$$

In both case 2 and case 3, it could not happen that $l_i = l_j$ nor that $|l_i - x^*| = |l_j - x^*|$ because that would imply that they are in the same location (case 1).

The profit earned by each firm is given by:

$$\pi_i(p_{i,t}, l_{i,t}, p_{j,t}, l_{j,t}) = (p_{i,t} - c_i) * D_{i,t}. \tag{4}$$

4. Q-Learning

In his thesis, Watkins proposes Q-Learning as a tool to tackle Markovian decision processes (MDP) (C. Watkins, 1989). A MDP is defined as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, R, \delta, \Omega)$ where $\mathcal{S}$ and $\mathcal{A}$ are the finite state and action spaces, respectivly. $T : \mathcal{A} \times \mathcal{A} \to \mathcal{S}$ and $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ are the transition and reward functions, respectivly. $\delta \in [0, 1)$ is the common discount factor to both algorithms. The initial state s_0 is drawn from the distribution function Ω.

The algorithm aims to find an optimal policy rule to maximize the discount sum of rewards. A policy rule is a mapping from the state space to the action space $g : \mathcal{S} \to \mathcal{A}$ that tells the algorithm which is the optimal action that has to be taken in every state.

After each algorithm plays an action $a_i \in \mathcal{A}$ in state $s \in \mathcal{S}$, they receive a reward $r(a_i|s) \in \mathbb{R}$ and the environment moves to another state $s' \in \mathcal{S}$. The transition probability from one state to another is unknown to both algorithms. This is because each algorithm when playing a_i does not know the action taken by the other algorithm. Also, the true expected reward r^* for playing one action is unknown and unobservable to the algorithm. Indeed, the objective of each algorithm is to estimate r^* for each state so she can play optimally.

Each algorithm is completely unaware of the existence of the other algorithm. For each one, it is as if the model has some noise. In that sense, the reward that the algorithm i sees after playing the action a_i in state s is a function of the true reward and a noise σ_i:

$$r(a_i|s) = r^*(a_i|s) + \sigma_i, \tag{5}$$

σ_i captures the influence of action a_j in the reward that algorithm i receives. Because the effect of playing action a is unknown for each algorithm (transition to another state and reward they receive), the model is stochastic. However, the environment is fully observable because the algorithms know in which state they are in every moment.

In this particular Hotteling setting, the objective of each algorithm is to choose a sequence of prices and locations $\{p_{it}, l_{it}\}_{t=0}^{\infty}$ in such a way as to maximize the discounted sum of profits:

$$\max_{\{p_{it} \in \mathcal{P},\ l_{i,t} \in \mathcal{L}\}_{t=0}^{\infty}} \mathbb{E}\left[\sum_{t=0}^{\infty} \delta^t \pi_{it}\left(p_{it}, p_{jt}, l_{it}, l_{jt}\right)\right], \tag{6}$$

where $\mathcal{P}$ and $\mathcal{L}$ are the discrete spaces of possible prices and locations, respectively. Each space is defined as:

$$\mathcal{P} = \{c, \frac{(N_p - 1)c + u}{N_p}, \frac{(N_p - 2)c + 2u}{N_p}, \ldots, \frac{2c + (N_p - 2)u}{N_p}, \frac{c + (N_p - 1)u}{N_p}, u\}, \tag{7}$$

$$\mathcal{L} = \{0, \frac{1}{N_l}, \frac{2}{N_l}, \ldots, 1\}, \tag{8}$$

where N_p and N_l are parameters to be defined. None algorithm will choose a price below their marginal cost c, nor set a price above the maximum price a consumer is willing to pay u. Note that if $c = 0$, the price grid will be

defined as:

$$\mathcal{P} = \{0, \frac{u}{N_p}, \frac{2u}{N_p}, \dots, u\}. \tag{9}$$

Rewriting the problem in its equivalent recursive form and in terms of the elements of $\mathcal{A}$ and $\mathcal{S}$ to eliminate the time reference:

$$V_i^*(s) = \max_{\{a_i \in A\}} \left\{ \mathbb{E}[\pi_i \mid s, a_i, a_{-i}] + \delta \mathbb{E}\left[V_i^*(s') \mid s, a_i, a_{-i} \right] \right\}, \tag{10}$$

where $V_i^*(.)$ is the value function of the problem, a_i is the action played by the algorithm i, and a_{-i} are the actions played by the rest of the algorithms, s is the current state, and s' is the state of tomorrow. In what follows, the prime ($'$) over a variable will be used as a shorthand for the variable's time-notation. So, if $s_t = s$, then $s_{t+1} = s'$. The same applies for a.

Notice, that the action space has two dimensions instead of one. In consequence, this problem falls inside the multi-action MDP field. A multi-action MDP is a MDP with a multi-dimensional action space. $\mathcal{A}$ is now defined as the product of sub-action spaces: $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times ... \times \mathcal{A}_{N_A}$ where N_A is the number of actions the algorithm must take. In this problem, $N_A = 2$ and $\mathcal{A} = \mathcal{A}_l \times \mathcal{A}_p$, where $\mathcal{A}_l$ and $\mathcal{A}_p$ are the action set when the algorithms choose location and price, respectively. The agent can handle the problem as N_A independent problems where in each one, she has to estimate a value function and construct an optimal policy function (Wang & Yu, 2016). However, when algorithms choose the price in this setting, they have to know both locations. In that sense, each problem can not be treated as totally independent from each other.

Both Q-Learning algorithms came into the world without knowledge of each action's rewards. In fact, by playing, they will be learning by trial-and-error. Their primitive objective is to estimate the long-run reward of each action in each state to be able to play optimally. In that sense, it is worthy to state the subjacent problem of the algorithms:

$$Q_i(s, a_i) = \mathbb{E}[\pi_i \mid s, a_i, a_{-i}] + \delta \mathbb{E}\left[\max_{a_i' \in A} Q_i(s', a_i') \mid s, a_i, a_{-i} \right], \tag{11}$$

where $Q_i(.)$ is the $\mathcal{S} \times \mathcal{A}$ reward matrix. The element $(s, a) \in Q_i$ is the expected return of playing action a in the state s.

Considering that the algorithms learn by trial-and-error and that they must *see* a reward after playing one action, it is necessary to have one reward matrix Q_i for each decision variable that the algorithm has. So, each algorithm will have one Q_i for the location problem and another one for the price problem. It is worth introducing some notation. The following table describes the notation used in each problem:

Insert Table 1 here.

In what follows, Q_i will be used as a synonym of $Q_{l,i}$ and $Q_{p,i}$ when it is indistinct to which particular problem I am referring. The same applies to a_i and s.

An action of the algorithm i in the location choice problem is a tuple $(l_{i,t}, p_{i,t})$. Note that besides choosing only the location, the action includes a price. This is because, when choosing the location, the algorithm internalizes the subsequent price decision. The way she takes into account the price is the following:

> For every location $l^{\#}$, pick the price p^* that is $\arg\max$ of $Q_{l,i}(s_l, a_l = (l^{\#}, p))$. Compute the reward of the action $a_l = (l^{\#}, p^*)$. The algorithm will have as many rewards as the number of possible locations. Then compare them all and keep with the maximum. The optimal location will be the one corresponding to this maximum reward.

An action in the price choice problem is a price p_i.

Under the one-period memory assumption, a state s_t contains information of the current period (price choice problem) or one-period behind (location choice problem). The one-period memory assumption is necessary for the state space to be finite. A state in the location choice problem is a pair of tuples $\{(l_{i,t-1}, p_{i,t-1}), (l_{j,t-1}, p_{j,t-1})\}$. A state in the price choice problem is a pair $\{l_{i,t}, l_{j,t}\}$.

Note that the last price played by both algorithms is not part of the state space of the $Q_{p,i}$ matrix. The reason behind this is that if not, there will be three-time moments in the updating of the $Q_{p,i}$ matrix ($s_p = \{$past prices, current locations$\}$ and $a_p = \{$current price$\}$, $s'_p = \{$current price, future locations$\}$). So, adding the past price as a state variable would break the one-time memory assumption.

After playing the action $a_i \in \mathcal{A}$ in state $s \in \mathcal{S}$ in the time moment t, the element (s, a_i) of the $Q_{i,t}$ matrix is updated according to the following equation:

$$Q_{i,t+1}(s, a_i) = (1 - \alpha) \cdot Q_{i,t}(s, a_i) + \alpha \cdot \left[\pi_i(s, a_i, a_{-i}) + \delta \max_{a'_i \in \mathcal{A}} Q_{i,t}(s', a'_i) \right]. \tag{12}$$

All the rest $Q_{i,t}$ matrix's elements $\neq (s, a_i)$ remain the same $Q_{i,t+1}(.) = Q_{i,t}(.)$. The parameter $\alpha \in [0, 1]$ is called the learning rate and affects the weight that the initial Q values have on the final Q values estimated and the speed of the learning process. The process is too slow for low values of α because it gives a small weight to the new rewards. In contrast, if $\alpha \to 1$, the algorithm forgets too fast what she has learned.

The algorithms must be exposed repeatedly to every state and play a priori non-optimal actions to ensure learning. For this purpose, the algorithms will follow an $\epsilon-$greedy action in each stage game. With probability ϵ, the action played by the algorithm will be uniformly picked from the action space $\mathcal{A} = \mathcal{A}_l \times \mathcal{A}_p$. With probability $1 - \epsilon$, the algorithm plays the optimal action for the current $Q_i(s, a)$ matrix.

$$\begin{cases} \sim U(\mathcal{A}_l \times \mathcal{A}_p) & \text{with probability } \epsilon \\ \arg\max Q_i(s, a) & \text{with probability } 1 - \epsilon \end{cases} \tag{13}$$

Note that if the algorithm choose optimally her action, then $a_{l,i}$ and $a_{p,i}$ are the argmax of $Q_{l,i}(s, a)$ and $Q_{p,i}(s, a)$, respectively. Contrary, if she acts randomly, then a tuple $(a_{l,i}, a_{p,i})$ is uniformly picked from all the possible location and price combinations.

The exploration rate ϵ_t is decreasing over time:

$$\epsilon_t = e^{-\beta t}, \tag{14}$$

where $\beta > 0$. Given β, for greater t, smaller ϵ_t and so, fewer actions are chosen randomly, and more are chosen optimally. So, at the beginning of the learning process, the algorithms randomize more frequently than later in the process. Also, ceteris paribus, greater β implies less exploration rate for all t. At one extreme, if $\beta \to \infty$, the algorithm always chooses optimally. However, the algorithm never passes through a learning process where she is forced to try a priori, suboptimal actions. Consequently, she could be stuck in a local optimum and never learn the best strategy. At the other extreme, if $\beta = 0$, the algorithm always randomizes and never chooses optimally. β also affects the rate at which the exploration rate varies. At greater β, the rate at which ϵ_t reduces itself is greater.

4.1 Q_i matrix updating

Figure 5 displays the timing and structure of the first two iterations of the game. The goal of this timeline is to clarify the updating of the two $Q_{l,i}$ and $Q_{p,i}$ matrices. In the figure 5, the current state is highlighted in blue, and the current action in green. The reward obtained by playing that action is highlighted in orange. The next state is highlighted in red. Finally, the moment of updating each $Q_{l,i}$ and $Q_{p,i}$ matrices is highlighted in violet.

Insert Figure 5 here.

The following equations make explicit the updating of both $Q_{l,i}$ and $Q_{p,i}$ matrices after the first action is played.

- $Q_{l,i}$ matrix:

$$Q_{l,i}(\ s_l = \{(l_{i,t-1}, p_{i,t-1}), (l_{j,t-1}, p_{j,t-1})\}\ ,\ a_l = \{(l_{i,t}, p_{i,t})\}\ ) = (1-\alpha)\cdot Q_{l,i}(s_l, a_l) + \alpha \left[\ \pi_i\ (s'_l)\ + \delta \max_{a'_l \in \mathcal{A}_l} Q_{l,i}(\ s'_l\ , a'_l)\right] \tag{15}$$

- $Q_{p,i}$ matrix:

$$Q_{p,i}(\ s_p = \{l_{i,t}, l_{j,t}\}\ ,\ a_p = \{p_{i,t}\}\ ) = (1-\alpha)\cdot Q_{p,i}(s_p, a_p) + \alpha \left[\ \pi_i\ (s_p\ ,\ a_p)\ + \delta \max_{a'_p \in \mathcal{A}_p} Q_{p,i}(\ s'_p\ , a'_p)\right] \tag{16}$$

Notice that $\pi_i(s_p, a_p) = \pi_i(s'_l)$. Both profits are the reward the algorithm recives after playing $\{l_i, p_i\}$ and when the rival is playing $\{l_j, p_j\}$.

Notice that all the information needed for updating the $Q_{l,i}$ matrix is inside one iteration. However, for updating the $Q_{p,i}$ matrix, it is necessary to use the information of two iterations. In particular, the updating requires the new location.

4.2 Pseudocode

The following code block is the pseudocode of the Q-Learning algorithm used in the simulations.

Insert Algorithm 1 here.

4.3 Convergence

There is a theoretical proof that guarantees the convergence of a single-agent Q-Learning problem with finite action and state-space and a deterministic environment (C. J. Watkins & Dayan, 1992). However, in a multi-agent problem, where the problem is stochastic, the convergence result does not necessarily hold. Consequently, there is no guarantee that the elements of the Q_i matrix will converge, nor that the algorithms find the optimal policy rule.

However, convergence can be empirically verified. Following Calvano et al., 2020, an experiment converges if the argmax of the Q_i matrix for each state remains stable over 100,000 consecutive periods. Since I have two Q_i matrices, an experiment converges if that condition holds for both $Q_{l,i}$ and $Q_{p,i}$. I call each of theese post-convergence matrix the *long-run matrix*.

5. BASELINE PARAMETER CONFIGURATION

Economic environment. I set $n = 2$ (duopoly), the marginal cost equal to cero $c = 0$ and no relocalization costs. For the demand side, I set the gross surplus (u) equal to 2.5. I define $\tau = 1$ for the transportation cost. The discount rate δ is set equal to 0.95.

Action and state space. I define $N_p = 7$ and $N_l = 3$. The cardinality of the action space depends on the decision variable of the algorithm. When she is choosing the location, the cardinality of the action space is $|A_l| = N_p \times N_l = 21$. In contrast, when choosing the price, the cardinality is $|A_p| = N_p = 7$.

The algorithms have one-period memory. In consecuence, the state space is $|S_l| = |A_l| \times |A_l| = 21 \times 21 = 441$, and $|S_p| = |N_l| \times |N_l| = 3 \times 3 = 9$, when choosing location and price, repectivly.

I inicializate all the elements of $Q_{p,i}$ and $Q_{l,i}$ equal to zeros. The dimension of each Q_i matrix is: $|Q_{l,i}| = |S_l| \times |A_l| = 441 \times 21 = 9,261$ and $|Q_{p,i}| = |S_p| \times |A_p| = 9 \times 7 = 63$. At the begining of each experiment, the initial state $s_l(t = 0)$ and $s_p(t = 0)$ are randomly choosen from a uniform distribution over each action space.

Convergence. The stage game is repeated for $T = 20,000,000$ periods or until the two $Q_{l,i}$ and $Q_{p,i}$ matrices converge, what happens first. Each repeated game consists of an experiment.

Experiments and simulation. Since the learning process depends on some randomness (initial state and the flipping coin that determines if the algorithms choose optimally or randomly their actions), and so, to reduce the stochastic noise, I run $N_{Exp} = 100$ experiments for each set of parameter configurations. This set of experiments is one simulation.

Representative simulation. I define the representative simulation as the one with $\alpha = 0.16$ and $\beta = 1.6 \times 10^{-5}$.

Parameter grid. I run several simulations, each one for diferent learning and experimentation parameters' values. In this way, I probe that the results do not depend on a particular parameter configuration.

I define the same learning and exploration parameter grid as Calvano et al., 2020. Then, the α grid and the β grid are define as the intervals $[0.025, 0.25]$ and $[0.0, 2 \times 10^{-5}]$. However, I take only six equal-spaced values from those intervals because of computational limitations. So, for this economic environment and action and state space configuration, I run 36 simulations. Each simulation is a different combination of α and β. Also, each simulation includes 100 experiments, and each experiment is ran for $T = 20,000,000$ periods or until convergence.

6. Benchmark equilibriums and performance metrics

6.1 Benchmark equilibriums

As a benchmark, I state some common equilibriums in a Hotelling setting.

Bertrand Equilibrium. In a one-shot game, and if the firms are located at the same location ($l_i = l_j$), in the unique Nash Equilibrium both firms set a price equal to the marginal cost $p^B = c$. Replacing the baseline parameter values of section 5, the Bertrand price is $p^B = 0$. Each firm's profit is $\pi^B = 0$.

Subgame Perfect Nash Equilibrium. If the location-price game is played only once, the firms choose to differentiate as much as possible ($l_i = 0$ and $l_j = 1$) and set a price equal to the transportation cost plus their marginal cost $p^{SPNE} = \tau + c$. Each firm's profit is $\pi^{SPNE} = \tau/2$. For the baseline parameter values, $p^{SPNE} = 1$ and, $\pi^{SPNE} = 0.5$.

Social optimum. Since demand is fully inelastic, a higher price does not generate a deadweight loss if the segment is fully covered. The price will only redistribute the total surplus between the firms and the consumers. The social optimum equilibrium will be the one that minimizes the total transportation cost. If there are three possible locations over the segment (baseline simulation), the pair of locations that minimizes the total transportation cost are the ones in which the distance between the firms is 0.5. If there are five possible locations over the segment (robustness check), the social optimum pair of locations is $\{0.25; 0.75\}$.

Table 2 shows the total transportation cost for each pair of locations. Note that the pair of locations when $N_l = 5$ include also the ones with $N_l = 3$. So, table 2b includes only the ones that are not present in 2a. Without loss of generality, I assume that both firms set the same price.

Insert Table 2 here.

Collusion. I consider the collusive outcome benchmark as the one where the two firms are active in the market and set the same price. It is not always true that the firms acting as a monopoly would prefer to serve the whole segment. According to the price they set, there will be a consumer that is indifferent between consuming and not consuming. Such consumer's location will determine if the whole segment is covered or not. Equating the consumer surplus to cero and solving for d:

$$CS = 0 = u - p_i - \tau d^2, \tag{17}$$

$$d(p_i) = \sqrt{\frac{u - p_i}{\tau}}, \tag{18}$$

where d is the distance of the indifferent consumer to firm i. Note that the distance of the indifferent consumer varies negatively with respect to the price.

If both firms coordinate their locations to maximize the joint profit, they will minimize the distance between them and the consumer further apart. So, if $N_l = 3$, they will never locate simultaneously at one extreme of the segment because the consumer further apart will be at one unit of distance. In all the other cases, there are 0.5 units of distance. In $N_l = 5$, the firms will locate in the same location of the social optimum equilibrium. This is the only pair of locations where the consumer further apart is at 0.25 units of distance.

The aggregated demand $D(p)$ for the monopoly firm when $N_l = 3$ and both firms locate at the middle or one at each extreme of the segment is:

$$D(P^C) = \begin{cases} 2\sqrt{\frac{u-p^C}{\tau}} * \frac{1}{1-0} & \text{if } d < 0.5, \\ 1 * \frac{1}{1-0} & \text{if } d \geq 0.5, \end{cases} \tag{19}$$

the first expression corresponds to the case when the segment is not fully covered. If not, all the consumers pucharse. Then, the demand is equal to the entire segment multiplied by the density function ($\frac{1}{1-0}$).

The monopoly firm will choose a price p^C to maximize the joint profit function:

$$\max_{p^C} \pi(p^C) = (p^C - c)D(P^C). \tag{20}$$

There will be an interior solution only in the not-fully covered case. On the contrary, when all the consumers pucharse one unit of good, the optimal price will be the one that solve $d(p) = 0.5$.

Deriving $\pi(p^C)$ and solving for p^C:

$$p^C = \frac{2u + c}{3}. \tag{21}$$

Replacing in the distance expression:

$$d(p^C) = \sqrt{\frac{u-c}{3\tau}}. \tag{22}$$

Replacing the parameters values of the baseline configuration:

$$p^C = \frac{5}{3} \approx 1.67 \qquad\qquad d(p^C) = \sqrt{\frac{5}{6}} \approx 0.83 \tag{23}$$

Due to fact that $d(p^C) > 0.5$, the optimal price is not $p^C = \frac{5}{3}$, but instead the one that solves $d(p^C) = \sqrt{\frac{u-p_i}{\tau}} = 0.5$. Solving for p^C:

$$p^C = \frac{9}{4} = 2.25 \tag{24}$$

The aggregated collusive profit is given by

$$\pi = (p^C - c) = 2.25 \tag{25}$$

When $N_l = 3$ but the firms locate at 0.5 units of distance between them, the aggregated demand is equal to:

$$D(P^C) = \begin{cases} 3\sqrt{\frac{u-p^C}{\tau}} * \frac{1}{1-0} & \text{if } d < 0.25, \\ 0.5 * \frac{1}{1-0} + \sqrt{\frac{u-p^C}{\tau}} * \frac{1}{1-0} & \text{if } 0.25 < d < 0.5, \\ 1 * \frac{1}{1-0} & \text{if } d \geq 0.5, \end{cases} \tag{26}$$

where the firsts two cases, the segment is not fully covered. In the first one, there are consumers, between the firms and between the middle firm and the segment limit, that do not pucharse at all. In the second case, all the consumers between the firms pucharse, but there are some consumers between the middle firm and the segment limit, that do not. Lastly, in the third case, all the consumers pucharse one unit.

Neither of the two first cases' solutions verify the distance restriction. So, the optimal collusive price is the same than the previous case.

Finally, when $N_l = 5$ the optimal pair of locations is $\{0.25, 0.75\}$. Then, the aggregated demand is equal to:

$$D(P^C) = \begin{cases} 4\sqrt{\frac{u-p^C}{\tau}} * \frac{1}{1-0} & \text{if } d < 0.25 \\ 1 * \frac{1}{1-0} & \text{if } d \geq 0.25. \end{cases} \tag{27}$$

Again, the first expression corresponds to the case with no-fully covered segment. If not, the demand is the second one.

Once again, the interior solution do not verify the distance restriction. So, the optimal collusive price is:

$$d(p^C) = \sqrt{\frac{u-p_i}{\tau}} = 0.25 \tag{28}$$

$$p^C = u - \frac{1}{16}\tau = 2.4375 \tag{29}$$

The aggregated collusive profit is given by

$$\pi = (p^C - c) * 1 * \frac{1}{1-0} = 2.4375 \tag{30}$$

Repeted game. In an infinitely repeated game, any supra-competitive outcome is sustainable for any discount factor of future payments greater than a threshold value (Folk-Theorem). Thus, firms can maintain collusive equilibria through a deviation and punishment scheme.

In a one-shot game, the firms will fully differentiate each other. However, an equilibrium with minimum differentiation may arise in a repeated game where the settlement of an implicit agreement reduces the price competition intensity.

6.2 *Performance metrics*

Profitability. I use two metrics to evaluate the performance of the algorithms in terms of profitability.

The first one is the evolution of the average one-period profit. To build this metric, I take the average between experiments (*exp*) of the one-period profit for every t:

$$\bar{\pi}_t = \sum_{exp=1}^{100} \frac{\pi_{t,exp}}{100}. \tag{31}$$

The second metric I use is the extra-profit-gain[5]:

$$\Delta = \frac{\tilde{\pi} - \pi^{PSNE}}{\pi^C - \pi^{PSNE}}, \tag{32}$$

where $\tilde{\pi}$, is the average of the 1,000 periods after convergence. On the other hand, π^{PSNE} and π^C are the Perfect Subgame Nash Equilibrium and the collusion one-period-profit, respectively. If $\tilde{\pi} = \pi^{PSNE} = 0.5$ then $\Delta = 0$. In this case, besides the repeated scenario, the algorithms do not manage to get supra-competitive payoffs. Contrary, if $\tilde{\pi} = \pi^C = 2.25/2 = 1.125$, then $\Delta = 1$. In this case, the algorithms manage to get the collusion profit. Note, however, that Δ could be negative. Negative values arise when the algorithms get a payoff less than the one of PSNE. For example, when they get the Bertrand payoff $\pi^B = 0$.

Replacing the benchmark profit values in the expression for Δ:

$$\Delta = \frac{\tilde{\pi} - 0.5}{0.625}. \tag{33}$$

The figure 6 presents a grid for Δ with some benchmark values.

Insert Figure 6 here.

Δ can be greater than 1 but can not be less than -0.8. The profits associated with $\Delta > 1$ could be related to a deviation reward. By cheating and breaking the implicit agreement, an algorithm could obtain a reward higher than the collusive one.

Theoretical Q matrix. After verified the convergence of the simulations, I check whether the algorithms are playing a Nash Equilibrium or a Perfect Subgame Nash Equilibrium.

Despite that each simulation converge and, in consecuence, both $Q_{l,i}$ and $Q_{p,i}$ matrices are stable, they are not necessarily optimal. Consequently, the algorithm might not be playing optimally in response to her rival. To check the optimality of the actions taken, I test how different are the long run $Q_{l,i}$ and $Q_{p,i}$ matrices to the $Q_{l,i}$ and $Q_{p,i}$ that arises when playing against a fixed-strategy competitor. I call theese matrices the *theoretical Q-matrices*.

To estimate the theoretical Q-matrix, one algorithm is forced to play her long-run strategy and cannot update her $Q_{l,i}$ and $Q_{p,i}$ matrices. So, it is as if this algorithm has finished the learning process. The other algorithm must pass through all possible states and update her $Q_{l,i}$ and $Q_{p,i}$ matrices. Then, I compare if, given a particular state, the argmax of the long run of both $Q_{l,i}$ and $Q_{p,i}$ matrices matches with the argmax of the theoretical $Q_{l,i}$ and $Q_{p,i}$ matrices. If verifying Nash Equilibrium, the states for comparison are the actions played during the first 1,000 iterations after convergence. Contrary, if checking for Subgame Perfect Nash Equilibrium, the comparison will be made for every possible state, independently if that state is ever reached during those 1,000 iterations.

[5] Calvano et al., 2020 calculate differently this metric. In particular, $\tilde{\pi}$ is a one-period-profit. It is the profit obtained by the algorithm upon convergence.

When the algorithms do not manage to play optimally, they are suffering an opportunity cost for not playing their best response. The Q-loss metric quantifies this cost:

$$Q_i\text{-loss} = \frac{q_i^T(a_i = \arg\max Q_i^T|s) - q_i^{LR}(a_i = \arg\max Q_i^{LR}|s)}{q_i^{LR}(a_i = \arg\max Q_i^{LR}|s)}, \tag{34}$$

where $q_i^T(.)$ is the reward of playing the argmax of the theoretical Q_i matrix given the state s (best response reward), and $q_i^{LR}(.)$ is the reward of playing the argmax action of the long run Q_i matrix given the state s. But each algorithm has two Q_i matrices, one for the location problem and one for the price problem. So, the Q_i-loss metric has to consider the two possible losses of not playing the best response in the location and the price problem. I define the aggregated Q_i-loss metric as the sum of $Q_{p,i}-$loss and $Q_{l,i}-$loss.

7. RESULTS

For the results, I saved the actions played and the profits obtained by each algorithm for the first 1,000 periods after convergence. If the experiment does not converge, I save the same elements for the last 1,000 iterations of the 20,000,000 iterations. Also, I saved the final $Q_{l,i}$ and $Q_{p,i}$ matrices for both algorithms. So, unless otherwise clarified, I used this database.

Convergence All the 100 experiments of the 36 simulations converge.

Long run action Although the argmax of the $Q_{l,i}$ and $Q_{p,i}$ matrices remains stable for every state in all the experiments of each simulation, the algorithms do not necessarily converge to play a single action[6]. That is to say, during the 1,000 periods after convergence, they do not necessarily play the same action. Figure 7a displays the fraction of the simulations that converge to a unit action cycle. Figure 7b displays the average action cycle length in those experiments where the action cycle duration is larger than one.

Insert Figure 7 here.

In the southwest corner of figure 7a, the learning process is slow (small α), and also the algorithms randomize their actions more (small β). Both factors affect the probability of reaching a unit action cycle. Indeed, when both algorithms always randomize their actions ($\beta = 0$), they never reach a unit action cycle. For others parameter configurations, the algorithms, at best, converge in the 70% of the simulation.

The bar reference in figure 7b is centered in very high values because of the large action cycles length in those configurations with $\beta = 0$. In contrast, in the other parameter configurations, the action cycle length barely exceeds 25 actions in the lowest value of α, and 7 actions in the rest of the simulations.

Presenting the results of all the simulations including those with $\beta = 0$ could be seen as erroneous. Also, the results of those simulations could be seen as irrelevant because the algorithms are always randomizing their actions. Moreover, one can argue that including those simulations changes the limits in the heatmaps' reference color bar, making harder the interpretation. However, the benefits of including them are greater than the costs. In particular, the results of those simulations provide an upper or lower limit that is useful in the figures' interpretation. In that sense, comparing the results of every simulation with $\beta \neq 0$ with those with $\beta = 0$ answers the question of how better is the algorithms' performance when they can choose optimally than when they cannot.

[6] In what follows, I will used as synonyms the following terms: converge to play a single action, converge to an action cycle of length one and converge to a unit action cycle.

The convergence to a unit action cycle is not a guarantee of playing a particular action. Figure 8a is a histogram of the action played on those unit action cycles. Over all the experiments of the 36 configurations, in 30.66% of those simulations, the algorithms converge to play a single action. In more than 70% of this 30.66%, each algorithm set a price equal to 2.08, and both are located in the middle of the segment. This is the mode of distribution. The second most repeated unit action cycle is the one where both algorithms set a price equal to 1.66, and they locate at the middle of the segment. So, the two most frequent actions are those where both algorithms are minimally differentiated and set a supra-competitive price.

Note that the equilibrium $\{(0.5, 2.083); (0.5, 2.083)\}$ is virtually the collusive one (see section 6). Indeed, the price grid do not include the collusive price because of the few points inside the grid. So, a price equal to 2.083 is the closest one to the collusive price of 2.25.

Insert Figure 8 here.

Figure 8b shows the fraction of the simulation where the action played in the unit action cycle is the mode distribution of the actions played in cycles of length one (in this case $\{(0.5, 2.083); (0.5, 2.083)\}$. The levels on this heatmap are very similar to the unit action cycle heatmap levels. In that sense, for some configurations of $(\alpha,\ \beta)$, it is not only more probable to reach a unit action cycle but also to play the most frequent action.

Profits Figure 9a plots the evolution of $\bar{\pi}_t$ as a function of time (t) for my representative simulation. For building this figure, I rerun the simulation, saving the algorithms' one-period profits from the very beginning.

Due to the high volatility of the profits, I calculate the simple moving average[7] (SMA) of the series. Figure 9b displays the SMA of the profits as a function of time (t).

Insert Figure 9 here.

It is clear from the figures that by the repetition of the game, the algorithms manage to get higher profits.

Figure 10 displays the boxplot of the extra profit gain for the representative simulation (10a) and the average between experiments of the extra profit gain of one algorithm, as a function of α and β (21a).

Insert Figure 10 here.

The performance of the algorithms in the representative simulation has very little variance. In almost every experiment, Δ is almost 0.9. Only in three experiments each algorithm obtains a profit that seems to be an outlier respect of the Δ distribution.

As expected, the average extra profit gain is negative when the algorithms always randomize their actions ($\beta = 0$). However, when $\beta \neq 0$, the average extra profit gain is greater than 0.32 for the lowest values of α and greater than 0.57 in the rest of the simulations.

Theoretical Q-matrix Figure 11a displays the evolution of the fraction of the simulations in which the algorithms converge to a Nash Equilibrium. Figure 11b plots the evolution of the aggregated Q-loss metric. Both plots are for the representative simulation. For making both figures, I rerun the simulation, saving, from the very beginning, the actions played by both algorithms in each iteration. Due to the significant volatility of both variables, I only plot the simple moving average[8] of each time serie.

[7] I take a 5,000 window period. However, the results are robust to changes in the window length.

[8] I take a window period of 5,000 iterations.

Insert Figure 11 here.

As the iterations go by, the figure shows a clear pattern in the learning process. In particular, the fraction of the simulation that converges to a NE grows, and the aggregated Q-loss decreases.

Figure 12a displays the fraction of the simulations where the algorithms converge to a Nash Equilibrium. Figure 12b plots the aggregated Q-loss for one algorithm.

Insert Figure 12 here.

Keeping aside the configurations with $\beta = 0$, in most of the other configurations, especially those with greater α, the algorithms mostly play a NE. That is to say, the actions chosen are the optimal response to the action played by the competitor. So, the minimal differentiation and the supra-competitive prices are not the results of luck and randomness. On the contrary, by the repeated play, the algorithms were able to learn that by playing that way, both are better.

I also check for the fraction of the simulation that converges to a PSNE. However, that equilibrium is never achieved. The non-convergence into PSNE is not surprising. Perfect Subgame Nash Equilibrium is very demanding. The algorithms must respond optimally in states that maybe are never be reached.

Collusion Before analyzing the existence of a deviation and punishment scheme, it is necessary to verify the non-existence of supra-competitive outcomes in the no-memory scenario and the positive relation between Δ and the level of patience of the algorithms.

No-memory When the algorithms have no memory, there is no possibility of collusion because there is no way they can punish deviations. Following Calvano et al., 2020, I modelled the no-memory scenario as a simulation with $\delta = 0$. Additionally, the $Q_{l,i}$ matrix is reduced only to its action space dimension. Because there is no memory, the algorithms do not remember past actions. However, when choosing a price, they do know the current locations. Consequently, the $Q_{p,i}$ do not suffer any changes. The new cardinality of the $Q_{l,i}$ matrix is $|Q_{l,i}| = |\mathcal{A}_l| = N_l \times N_p = 3 \times 7$ matrix. It remains with two dimensions because when choosing the location, the algorithms continue to internalize the next price choice.

The value of the exploration parameter β remains the same (this implies that the times an action is played randomly is now greater). Figure 13 displays the extra profit gain when the algorithms have no memory.

Insert Figure 13 here.

As expected, every point in the Δ distribution when the algorithms have no memory is below the minimum value of the Δ distribution when the algorithms do have memory (see figure 10a). In more than 75% of the simulation, the extra profit gain is negative. The negative values correspond to profit values less than the one of SPNE. Nevertheless, the extra profit gain distribution does not reach the lowest possible value of -0.8, which corresponds to Bertrand's Nash Equilibrium profit.

Respect figure 13b, the average extra profit gain values are notably less when the algorithms have no memory. In this scenario, there is no possible collusion. Consequently, the algorithms only manage to obtain a profit that is, at best, the one of SPNE.

Average Extra Profit Gain Δ as a function of the discount factor δ Following the Folk-theorem, equilibriums, where the algorithms obtain profits higher than the competitive one, are sustainable for levels of patience greater than a minimum threshold. In that sense, a positive relationship exists between the maximum profit an algorithm can receive and her future payoff valoration[9]. For this analysis, I run the representative simulation for different values of δ. In particular, I take $N_\delta = 20$ equaled spaced values from the grid $\{0.0, ..., 0.99\}$. The figure 14 displays the evolution of each algorithm's average extra profit gain as a function of the discount factor δ.

Insert Figure 14 here.

For both algorithms, the plot confirms the positive relation between the supra-competitive profits and the level of patience.

Deviations and Punishments To check whether the supra-competitive outcomes are part of an implicit collusion agreement between the algorithms and not mere results of luck and randomness, I verify the existence of a deviation and punishment scheme.

Starting from the long-run action played by the algorithms, I force one of them to deviate from her optimal action. In particular, she carries on a price reduction without changing her past location. If the long-run action is not a unique pair of location and price, the deviation is forced at each point of the action cycle. Then, for each time period, I take the average of the responses.

When the action cycle is of length one, the algorithms never choose to play the lowest price ($p_i = 0$). However, it is sometimes played in some of the other action cycles. Playing the lowest price is a problem because there is no possible price reduction in those cases. The figure 15 shows the fraction of the simulation where the minimum price is played during the action cycle.

Insert Figure 15 here.

As was expected, the simulations with $\beta = 0$ are the ones with more quantity of experiments in which the action cycle includes the minimum price. This fraction of the simulation is reduced when the algorithms choose their actions optimally more frequently (bigger β) and when the learning process is faster (bigger α). Nevertheless, for every parameter configuration, there is at least one experiment where, in the action cycle at which the algorithms converge to play, the algorithms set the minimum price. In other words, the number of experiments, where $p = 0.0$ is included in the action cycle, is always grater than cero for every combination of α and β.

There are two options[10] to implement the deviation and punishment analysis in those cases. The first one is to leave those experiments out of the sample. The second one is to start the analysis from the rest of the points inside the action cycle and skip starting from that point. Each of the options have some drawbacks. On one hand, the experiment-exclusion alternative cannot be implemented when the fraction of the simulations with $p = 0.0 \in$ the action cycle is 100%. So, this alternative is impossible when $\beta = 0$. Also, there are some

[9] However, this positive relationship is not necessarily monotone in a repeated hotelling game. For example, see Chang, 1992.

[10] Forcing a location deviation may be a third option. In that case, all deviations would have to be of location type. No analysis is possible if in some experiments the cheating algorithm implements a location deviation and in others a price deviation. However, even forcing all deviations to be of location type, neither the deviation's interpretation nor the algorithms' response is straightforward. Moving away without changing the price is not always profitable. In this sense, an algorithm is forced to implement a location deviation that is sometimes not rational. Finally, a fourth option can be forcing a simultaneous location and price deviation. In other words, the cheating algorithm is forced to undercut its rival's price and also to move in the location grid. However, this alternative faces the same difficulties as the third one. Also, it is less clear the interpretation of the algorithms' response because both variables moved at the same time during the deviation.

parameters configuration where the fraction of the simulations with $p = 0.0 \in$ the action cycle is less that 100%, but rose very high values. In those cases, excluding that fraction of the simulation will leave the simulation with few experiments.

On the other hand, in the point-avoidance alternative I am not cutting off some experiments from the simulations, but rather I am cutting off some played actions within the experiments. Note that in the experiments in which $p = 0.0 \notin$ action cycle both alternatives are the same. However, in all simulations, there is at least one experiment where $p = 0.0 \in$ action cycle. Therefore, when analyzing the mean response of price and location after a deviation, one is averaging the response between experiments that include $p = 0.0$ in the action cycle with experiments that do not.

I decided to implement the first option. In the most interesting simulations, where the values of α and β are far from their lowest grid's values, the number of excluded experiments is not too large. Also, playing $p = 0.0$ is not economically interesting. That action is dominated by setting another price. When an algorithm chooses $p = 0.0$, the reward is always zero. But if she chooses some $p > 0$ the payoff is strictly positive for some equilibriums. Consequently, the fact that the algorithms set $p = 0, 0$ appears to be more of a learning flaw than an optimal response. For all these reasons, I prefer to work only with experiments in which the algorithms do not choose to play $p = 0, 0$. After discarding the experiments where $p_i = 0$ is played, I keep with 93% of the representative simulation.

Figure 16 display the evolution of the average, between experiments, of the price (panel 16a) and location (panel 16b) played by both algorithms after one deviates in the representative simulation.

After some periods of punishment, the algorithms manage to return to the price played before the deviation occurred.

Figures 17 and 18 display the boxplot of the price and location change for both algorithms after the deviation is forced in the representative simulation.

Insert Figure 16 here.

Insert Figure 17 here.

Insert Figure 18 here.

8. Robustness checks

In this section, I repeat the analysis of the section 7 but extending the number of available prices inside the price grid (section 8.1), the number of available locations in the location grid (section 8.2), and introducing relocalization costs (section 8.3). In this way, I probe that the results do not depend on the particular action space of the baseline configuration. Also, that the results are robust to the introduction of relocalization costs.

8.1 More prices

I define $N_p = 9$. The cardinality of the action space when choosing location is now $|A_l| = N_p \times N_l = 9 \times 3 = 27$. On the other hand, the cardinality of the action set when choosing price now is $|A_p| = N_p = 9$. Respect the state

space: $|S_l| = |A_l| \times |A_l| = 27 \times 27 = 729$, and $|S_p| = |N_l| \times |N_l| = 3 \times 3 = 9$. Finally, both $Q_{l,i}$ and $Q_{p,i}$ matrices now have $|Q_{l,i}| = |S_l| \times |A_l| = 729 \times 27 = 19,683$ and $|Q_{p,i}| = |S_p| \times |A_p| = 9 \times 9 = 81$ elements.

All the benchmark situations' price and profit values remain the same. The same is for α and β grid. So, I run 36 simulations again but now with a more dense price grid. Each experiment is ran for $T = 20,000,000$ periods or until convergence.

Results

Convergence All the experiments of the 36 simulations converge.

Long-run action Figure 19 displays the fraction of the simulation that converges to a unit action cycle (panel 19a) and the average action cycle length of those simulations that do not converge to a unit action cycle (panel 19b). Apart from those configurations with $\beta = 0$, the configurations with the lowest value of α never converge to a unit action cycle.

Insert Figure 19 here.

The maximum fraction of simulations that converges to a unit action cycle for a certain parameter configuration is reduced with respect to the baseline configuration. At best, the algorithms converge in 55% of the simulations. Moreover, the action cycle length of those experiments where the algorithms do not converge to a single action play increased, especially when $\beta = 0$. That was to be expected because, in those configurations, the algorithms always randomize their actions. So, if the action space is bigger, the action cycle will also be larger. The average action cycle length is less than 31 for the configuration with the minimum value of α, and less than 10 for the rest of the simulations.

Figure 20a is the action played histogram in those unit action cycle. Figure 20b is the fraction of the simulations where the algorithms converge to a unit action cycle where the tuple of the algorithms' actions is equal to the mode of the action-play distribution.

Insert Figure 20 here.

From all the experiments of the 36 simulations, in only 23.41% of those, the algorithms converge to play a unit action cycle. In almost half of those 23.41%, both algorithms locate at the middle of the segment and set a price of 2.18. Note that this equilibrium is virtually the collusive one (see section 6). Indeed, the price grid do not include the collusive price because of the few points inside the grid. So, a price equal to 2.18 is the closest one to the collusive price of 2.25.

So, similar to the results in section 7, the algorithms choose the same location in the two most frequent actions. Also, they set a supra-competitive price but not the highest one. However, the fraction of the simulation where the algorithm converges to a unit action cycle is reduced from 30.66% to only 23.41%.

Respect figure 20b, from those simulations where the algorithms converge to a unit action cycle, the fraction where the algorithms are playing $\{(2.18, 0.5), (2.18, 0.5)\}$ is sensitive to the value of α. In particular, the probability of converging to a unit action cycle playing the distribution mode is higher for higher values of α. Surprisingly, it is also higher for lower values of β.

Long-run profits Figure 21a displays the distribution over the experiments of the representative simulation of the extra profit gain. As in section 7 (see figure 10a), the hole distribution is above the null value ($\Delta = 0$) corresponding to the profits of PSNE. However, with a larger action space ($N_p = 9$), the distribution has significantly more variance.

Figure 21b is the heatmap of the average, between experiments, of the extra profit gain as a function of α and β. The heatmap is virtually identical to the one in section 7 (see figure 10b). In that sense, besides having a larger action set and, so, a more challenging learning process, the algorithms obtain supra-competitive profits in almost every parameter configuration. As expected in the configuration with $\beta = 0$, the extra profit gain is lower because the algorithms always randomly pick their actions.

Insert Figure 21 here.

Theoretical Q-matrix Figure 22a displays the fraction of simulations converging to a Nash Equilibrium. Figure 22b plots the aggregated Q-loss for an algorithm as a function of α and β. The algorithms never converge to a Subgame Perfect Nash Equilibrium.

Insert Figure 22 here.

The algorithms manage to play optimally besides having a bigger action space. The large fraction of simulations that converge to a NE implies that the minimum differentiation and the supra-competitive prices are an optimal response to the optimal action of the competitor.

As expected, the lowest values of convergence to a NE and the highest values of aggregated Q-loss corresponds to the parameter configurations with $\beta = 0$.

Collusion. Deviation and punishment Figure 23 shows the fraction of the simulations where the minimum price is played during the action cycle.

Insert Figure 23 here.

Figure 24 displays the evolution of the average, between experiments, of the price (panel 24a) and location (panel 24b) played by both algorithms after one deviates in the representative simulation.

Insert Figure 24 here.

After some periods of punishment, the algorithms manage to return to a supra-competitive price as the one played before the deviation occurred.

Figures 25 and 26 display the boxplot of the price and location change for both algorithms after the deviation is forced in the representative simulation.

Insert Figure 25 here.

Insert Figure 26 here.

8.2 More locations

I define $N_l = 5$. With this modification, but with the other parameter values remaining the same as in Section 7, the segment is still fully covered in the collusive benchmark scenario. However, the aggregeated collusive profit is now $\pi = 2.4375$ (see section 6). Consequently, the benchmark values of Δ do not remain the same.

Replacing the new values in the expression for Δ:

$$\Delta = \frac{\tilde{\pi} - 0.5}{0.71875}. \tag{35}$$

The figure 27 presents the new grid for Δ with some benchmark values.

Insert Figure 27 here.

The other benchmark situations' price and profit values remain the same.

This modication changes the cardinality of the action and state sets. Now the cardinality of the action space when choosing location is $|A_l| = N_p \times N_l = 7 \times 5 = 35$. On the other hand, the cardinality of the action set when choosing price remains the same $|A_p| = N_p = 7$. Respect the state space: $|S_l| = |A_l| \times |A_l| = 35 \times 35 = 1,225$, and $|S_p| = |N_l| \times |N_l| = 7 \times 7 = 49$. Finally, both $Q_{l,i}$ and $Q_{p,i}$ matrices now have $|Q_{l,i}| = |S_l| \times |A_l| = 1225 \times 35 = 42,875$ and $|Q_{p,i}| = |S_p| \times |A_p| = 49 \times 7 = 343$ elements.

The α and β grid remain the same as the previous robustness checks. So, I run 36 simulations again but now with a more dense location grid. Each experiment is ran for $T = 20,000,000$ periods or until convergence.

Results

Convergence All the experiments of the 36 simulations converge.

Long-run action Figure 28 display the fraction of the simulations that converges to a unit action cycle (panel 28a) and the average action cycle duration of those simulations that do not converge to a unit action cycle (panel 28b). There is a similarity with the previous robustness check with more prices. In both setups, there are several simulations where the algorithms do not manage to converge to an action cycle of length one in any experiment, apart from the ones with $\beta = 0$. These simulations correspond to low values of α in both setups. So, when the learning process is slow and the action space is expanded, the convergence to a unit action cycle is tougher.

Insert Figure 28 here.

Note that, at best, the algorithms converge to a unit action cycle in less than 30% of a simulation. This value is less than the 70% achieved in section 7 (see figure 7a) and also less than the 50% of section 8.1 (see figure 19a). This is as expected because besides in both robustness checks (section 8.1 and section 8.2) the action space is enlarged; the first one is still smaller than the second one. Also, by looking at figure 28a, there is no clear pattern in the levels. Indeed, the highest value is not achieved in the northeast corner but in the northwest one.

Figure 28b is very similar to the one shown in section 7 (see figure 7b). However, the values in the reference bar are higher. In particular, with a more extensive action space, there are more possible action combinations. So, as expected, the average action cycle length is now greater. When $\beta \neq 0$ and α take its minimal value, the average cycle length is at most equal to 60. For the rest of the simulations, the average length is always less than 17.

Figure 29a is the action played histogram in those unit action cycles. The plot 29b is the fraction of the simulations where the algorithms converge to a unit action cycle and play the mode of the action-played distribution.

Insert Figure 29 here.

In 10.16% of all the experiments of the 36 simulations, the algorithms converge to play a single action. In 16% of this 10.16%, the algorithms choose not to differentate and to set a price above the competitive one. As in section 7, the most frequent action played is $\{(2.083, 0.5); (2.083, 0.5)\}$. However, the fraction of simulations that converge to a unit action cycle is reduced to only 10.16%. In section 7 this value was 30.66% (see figure 8a) and in section 8.1, was 23.41% (see figure 20a). So, by enlarging the action space, the fraction of the simulations where both algorithms converge to an action cycle of a single action is reduced.

In contrast with section 7 and section 8.1, where the most frequent equilibrium in the unit action cycles was the collusive one, in 29a it is not. The collusive equilibrium corresponds to the second and third most frequent equilibrium, but not the first one. However, the reason behind this fact is discretization. Indeed, the algorithms do not lose demand for choosing the locations equal to the middle point of the segment and not the locations $\{0.25, 0.75\}$. Regardless of whether the locations of the algorithms are $\{0.5, 0.5\}$ or $\{0.25, 0.75\}$, the most-near collusive price is the same and at that price the segment is fully covered.

With respect to figure 29b, the fraction of the simulations that converges to a unit action cycle equal to the mode of the distribution is notably reduced. Also, in contrast with the relationship established in the section 7, now there is no clear pattern. There is not a clear pattern where there is more chance for specific configurations to converge to a unit action cycle and to be playing the mode of the distribution.

Long-run profits Figure 30a displays the distribution of the extra profit gain for each algorithm for the representative simulation. The mean of each distribution is significantly below the one displayed in the other sections (see figure 10a in section 7 and figure 21a in section 8.1). Also, the minimum and maximum values of the distribution are now farther apart and the distribution presents a greater variability. However, as in section 7 and 8.1, the hole distribution is above the competitive benchmark value of $\Delta = 0$.

Insert Figure 30 here.

The main difference between figure 30b and the one in section 7 (see figure 10b) is about the maximum value. Here, the algorithms, at best, obtain an average extra profit of 0.6. In section 7, the maximum value is 0.8.

Theoretical Q-matrix Figure 31a displays the fraction of simulations converging to a Nash Equilibrium for a grid of values of α and β. Figure 31b displays the aggregated Q-loss of one algorithm. As in section 7, Subgame Perfect Nash Equilibrium is never achieved.

Insert Figure 31 here.

Although the action space is greater, although less fraction of the simulations converge to a unit action cycle, the algorithms manage to play optimally in most cases.

Collusion. Deviation and punishment Figure 32 shows the fraction of the simulations where the minimum price is played during the action cycle.

Insert Figure 32 here.

Figure 33 display the evolution of the average, between experiments, of the price (panel 33a) and location (panel 33b) played by both algorithms after one deviates in the representative simulation.

Insert Figure 33 here.

After some periods of punishment, the algorithms manage to return to a supra-competitive price as the one played before the deviation occurred.

Figures 34 and 35 display the boxplot of the price and location change for both algorithms after the deviation is forced in the representative simulation.

Insert Figure 34 here.

Insert Figure 35 here.

8.3 Relocalization costs

In this section, I repeat the analysis of section 7 but now introduce relocalization costs. Now, the algorithms will suffer a quadratic payment according to the distance they move apart from their previous location. Instant profit is now defined as:

$$\pi_i(p_{i,t}, l_{i,t}, p_{j,t}, l_{j,t}) = (p_{i,t} - c_i) * D_{i,t} - RC(l_{i,t}, l_{i,t-1}) \tag{36}$$

where $RC(l_{i,t}, l_{i,t-1})$ is the relocation cost function. In particular, it is a quadratic function of the number of places within the location grid that the firm moves apart from its location in the previous period:

$$RC(l_{i,t}, l_{i,t-1}) = 0.25 * (\text{number of locations the algorithm moves})^2 \tag{37}$$

Note that the algorithm suffer this cost when choosing locations, but not when choosing prices. So, the instant profit used in the $Q_{l,i}$ updating will be the one given in equation 36, and the instant profit used in the $Q_{p,i}$ updating will be the one used in the baseline configuration (see equation 4).

The introduction of relocalization costs do not modify the benchmark equilibrium values. They remain the same as in section 7 and in section 8.1. Neither it affects the cardinality of the action and state spaces, nor the dimensions of both $Q_{l,i}$ and $Q_{p,i}$ matrices. The α and β grid remain the same as in the previous robustness checks. So, I rerun 36 simulations. Each experiment is ran for $T = 20,000,000$ periods or until convergence.

Results

Convergence All the experiments of the 36 simulations converge.

Long-run action Figure 36 displays the fraction of the simulation that converges to a unit action cycle (panel 36a) and the average action cycle length of those simulations that do not converge to a unit action cycle (panel 36b).

Insert Figure 36 here.

The presence of relocalization costs reduces the incentives to change location. In consequence, the algorithms prefer changing their price rather than their location. This modification increases the incentives to collude for several reasons. First, the fact that the algorithms prefer not to change location is like an action space reduction because some actions are virtually discarded. The action space reduction simplified the learning process of the algorithms and in consequence, increases the probability of convergence to a supra-competitive outcome. Second, the introduction of relocalization costs reduces the incentives for cheating because it increases the punishment. The possibility of differentiating reduces the price punishment. But now, with less incentives to move across locations, this punishment is more severe.

The relocalization cost introduction increases the fraction of simulations that converge to a unit action cycle respect to the results obtained in section 7 (see figure 7a). In section 7, the algorithms at best converge in 70% of the simulation, now this value grows to almost 90%.

Figure 37a is the action played histogram of the unit action cycles. From all the experiments of the 36 simulations, in 39.61% of those, the algorithms converge to play a single action. This value represents an increase respect to 30.66% achieved in section 7 (see figure 8a), to 23.41% in section 8.1 (see figure 20a), and to 10.16% in section 8.2 (see figure 29a). In more than 80% of these 39.61%, the algorithms choose not to differentiate and to set a price equal to 2.08. As it was mentioned in section 7, this equilibrium is the collusive one.

Figure 37b is the fraction of the simulations that converge to play a unit action cycle equal to the action-played distribution mode. The introduction of the relocalization costs increases the figure's values respect the ones displayed in the baseline configuration figure. In section 7 the highest value was 60% (see figure 8b). Now it is near 80%.

Insert Figure 37 here.

Long-run profits [11] Figure 38a displays the boxplot of the extra profit gain distribution. The figure is very similar to the one in section 7 (see figure 10a). Both distributions are centered in a very high value, both have very little variance and only a few outliers. However, the minimum value in figure 38a (a little bit less than 0.55) is above the minimum value in figure 10a (near 0.3).

Insert Figure 38 here.

Theoretical Q-matrix Figure 39 display the fraction of simulations converging to a Nash Equilibrium (panel 39a) and the aggregated Q-loss (panel 39b) for a grid of values of α and β. Subgame Perfect Nash Equilibrium is never achieved.

Insert Figure 39 here.

[11] To simplify the comparison respect to the other model specifications, the profit series will be the one obtained without subtracting the relocalization costs (see equation 4).

Besides introducing relocalization costs, the algorithms manage to play a NE. In that sense, they avoid being stucked in a non-optimal action-play. The relocalization costs could change the incentives of the algorithms and force them to play a non-optimal action to avoid the cost of changing location. However, this is not the case. Both algorithms play a NE in most of the simulations. Indeed, besides the configurations with $\beta = 0$, the minimum heatmap value is 74. In section 7, the minimum value is 47 (see figure 12a). As expected, the configurations with greater aggregated Q-loss are the ones with $\beta = 0$.

Collusion. Deviation and punishment Figure 40 shows the fraction of the simulations where the minimum price is played during the action cycle.

Insert Figure 40 here.

Figure 41 displays the evolution of the average, between experiments, of the price (panel 41a) and location (panel 41b) played by both algorithms after one deviates in the representative simulation.

Insert Figure 41 here.

The introduction of relocation costs reduces the incentives to punish a deviation. Thus, besides an algorithm breaking the agreement and undercutting the price of its rival, the cheated algorithm does not penalize by lowering its price. The absence of punishment is not a minor fact. Indeed, the equilibrium reached by the algorithms cannot be said to be a collusive equilibrium because there is no underlying deviation and punishment scheme. So, besides the fact that the most frequent outcome matches a collusive equilibrium, technically it is not.

Figures 42 and 43 display the boxplot of the price and location change for both algorithms after the deviation is forced in the representative simulation. As was expected, the introduction of the relocalization cost reduces the variance in the location response after the deviation.

Insert Figure 42 here.

Insert Figure 43 here.

The result in figure 41a is surprising because the algorithms do not suffer a price-change cost but rather a location-change cost. So, after being cheated, the algorithms still have the power to change their price without suffering an extra payment. Indeed, one might think that because of the relocalization cost, the algorithms are sure that a punishment will be in terms of price and not in terms of location. So, one is expecting a big price punishment in figure 41a, few line breaks in the average location response (figure 41b) and low variance in the location response (figure 43). However, only the last two points occurred.

The theory indicates that the introduction of relocalization costs creates a more favorable environment for collusion. Mainly because the relocalization cost reduces the incentives to switch locations after a deviation. And so, it increases the price punishment. However, in figure 41a there is no such price punishment. However, neither is the deviated algorithm undercutting prices during several periods. In the next period after deviation, she returns to set a similar to the one pre-deviation.

One possible explanation of this phenomenon is a biased Q-matrix. Some actions' rewards may be downward biased because of the relocalization cost suffered in previous iterations. A second possible explanation is that the relocalization cost may cause a decrease in the speed of response. In other words, after a deviation, the best

strategy of the cheated algorithm maybe is to *let the storm pass*. Following this idea, a price punishment occurs only if the *storm* lasts too long. Figure 44 displays the evolution of the average response after a long deviation in the representative simulation. For this analysis, the cheater algorithm is forced to undercut the other algotithms' price during ten consecutive periods.

Insert Figure 44 here.

The evolution of the price response of the cheated algorithm is surprisingly slow. Only eight periods after the deviation, its price undercuts the cheater's price level (see figure 44a). Moreover, once the cheater is allowed to change her price, she significantly rises it and the other algorithms follows it. In consequence, the price punishment after a deviation is really short. However, this fact is not particular to this setup. Figure 16a, 33a, and 24a also show a brief price punishment. On the other hand, besides the presence of relocalization costs, the evolution of the location response is, surprisingly, more erratic than the price evolution.

Figures 45 and 46 display the boxplot of the price and location change for both algorithms after a long deviation is forced in the representative simulation.

Insert Figure 45 here.

Insert Figure 46 here.

9. Conclusion

Algorithmic collusion is one of the main concerns on the antitrust agenda nowadays. Previous studies show that two independent Q-Learning algorithms can reach supra-competitive outcomes in several settings (Ballestero, 2022; Calvano et al., 2020; Calzolari et al., 2021; Klein, 2019). However, all of these studies were limited to a one-variable decision problem. The present paper is the first one that analyzes the emergence of algorithmic collusion in a two-variable decision problem.

This paper analyze the competition between two Q-Learning algorithms in a Hotelling setting. In most simulations, I found that two Q-Learning algorithms that repeatedly compete in a location-price game choose not to differentiate and set a price greater than the competitive one. Moreover, the strategy that each algorithm follows is, in most cases, the best response to the rival's strategy. In other words, they are playing a Nash Equilibrium. An underlying deviation and punishment scheme sustains this implicit agreement. So, if one algorithm breaks the implicit agreement by undercutting its rival's price, the other algorithm punishes the cheater. However, the punishment is finite in time. So, after some iterations, both algorithms return to set a similar supra-competitive price to the one set before the deviation. Summing up, the repeated competition and the existence of a deviation and punishment scheme reduce the price competition and lead to equilibrium with minimum differentiation and supra-competitive prices.

When the action space of the algorithms is augmented, the learning process is tougher. The convergence towards playing a single action is less frequent. However, the most frequent equilibriums in those unit action cycles, is still the one with minimum differentiation and supra-competitive prices. Also, the fraction of the simulations where the algorithms converge to play a Nash Equilibrium is still high. Moreover, the deviation and punishment scheme is still present because after one of the algorithms undercuts its rival's price, the other punishes the cheater.

When the algorithms suffer relocation costs when changing location, the performance of the algorithms improved. The experiments are more likely to converge to a unit action cycle. As before, the most frequent equilibrium in those unit action cycles is those with minimum differentiation and collusive prices. Moreover, they play a NE in most simulations. So, they are playing optimally in most situations. However, the deviation and punishment scheme is only present when the price deviation lasts for long periods. In that sense, if the cheating is brief, the cheated algorithm prefers not to punish. But if the price undercutting lasts for many periods, the cheated algorithm progressively lowers its price. Only after sufficient many periods, the cheated price is lower than the cheter's price. Surprisingly, the location response after a deviation is more erratic than the price response.

The real world is far more complex than this stylized model, and firms are unlikely to use these simple algorithms in their decision-making process. However, the transparency and the straightforward parameter interpretation of the Q-Learning algorithms give useful insights. This results sheds light to a field that remains obscure due to numerous black-box algorithms. Nevertheless, each new study has to be seen as a modest step toward a better understanding about algorithmic collusion. However, the field is far from reaching a final word. Indeed, each study raises new questions. For example, if the results also hold under different learning algorithms. If the algorithms can also collude in a vertical differentiated model, under an uncertain environment or in a circular city. All these questions remain open for further research.

References

Asker, J., Fershtman, C., & Pakes, A. (2021). *Artificial Intelligence and Pricing: The Impact of Algorithm Design*, National Bureau of Economic Research. https://www.nber.org/papers/w28535

Ballestero, G. (2022). *Collusion and Artificial Intelligence: A Computational Experiment with Sequential Pricing Algorithms under Stochastic Costs*, RedNIE.

Brown, Z. Y., & MacKay, A. (2021). *Competition in Pricing Algorithms*, National Bureau of Economic Research. http://www.nber.org/papers/w28860

Calvano, E., Calzolari, G., Denicolo, V., & Pastorello, S. (2020). Artificial intelligence, algorithmic pricing, and collusion. *American Economic Review*, *110*(10), 3267–3297. https://doi.org/10.1257/aer.20190623

Calvano, E., Calzolari, G., Denicolò, V., & Pastorello, S. (2019). Algorithmic Pricing What Implications for Competition Policy? *Review of Industrial Organization*, *55*, 155–171.

Calzolari, G., Calvano, E., LastNameDenicolò, V., & Pastorello, S. (2021). Algorithmic collusion with imperfect monitoring. *International journal of industrial organization*, *79*, 102712. https://www.sciencedirect.com/science/article/pii/S0167718721000059

Chandak, Y., Theocharous, G., Kostas, J. E., Jordan, S. M., & Thomas, P. S. (2019). Learning action representations for reinforcement learning. *International conference on machine learning*, 941–950. https://proceedings.mlr.press/v97/chandak19a.html

Chang, M. H. (1992). Intertemporal product choice and its effects on collusive firm behavior. *International Economic Review*, 773–793.

Chen, L., Mislove, A., & Wilson, C. (2016). An empirical analysis of algorithmic pricing on amazon marketplace. *25th international conference on Wolrd Wide Web*, 1339–1349. https://dl.acm.org/doi/abs/10.1145/2872427.2883089

CMA. (2018). *Pricing algorithms*, Competition and Market Authority. https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/746353/Algorithms_econ_report.pdf

d'Aspremont, C., Gabszewicz, J., & J.-F. Thiss. (1979). On Hotelling's "Stability in Competition". *Econometrica*, *47*(5), 1145–1150.

Dogan, I., & Güner, A. R. (2015). A reinforcement learning approach to competitive ordering and pricing problem. *Expert Systems*, *32*, 39–48. https://doi.org/10.1111/exsy.12054

Harrington, J. E. (2018). Developing Competition Law for Collusion by Autonomous Artificial Agents. *Journal of Competition Law & Economics*, *14*(3), 331–363.

Kimbrough, S., & Murphy, F. (2009). Learning to collude tacitly on production levels by oligopolistic agents. *Computational Economics*, *33*(1), 47–78. https://doi.org/10.1007/s10614-008-9150-6

Klein, T. (2019). Autonomous algorithmic collusion: Q-learning under sequential pricing. *Amsterdam Law School Research*. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3195812

Li, X., Wang, J., & Sawhney, R. (2012). Reinforcement learning for joint pricing, lead-time and scheduling decisions in make-to-order systems. *European Journal of Operational Research*, *221*(1), 99–109. https://doi.org/10.1016/J.EJOR.2012.03.020

Mehra, S. K. (2016). Antitrust and the Robo-Seller: Competition in the Time of Algorithms. *Minnesota Law Review*, *204*.

Moodley, P., Rosman, B., & Hong, X. (2019). Understanding structure of concurrent actions. *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, *11927 LNAI*, 78–90. https://doi.org/10.1007/978-3-030-34885-4{_}6

OECD. (2017). Algorithms and Collusion: Competition Policy in the Digital Age. https://www.oecd.org/daf/competition/Algorithms-and-colllusion-competition-policy-in-the-digital-age.pdf

Ohlhausen, M. (2017). 'Should We Fear the Things That Go Beep in the Night? Some Initial Thoughts on the Intersection of Antitrust Law and Algorithmic Pricing' Remarks from the Concurrences Antitrust in the Financial Sector Conference. https://www.ftc.gov/news-events/news/speeches/should-we-fear-things-go-beep-night-some-initial-thoughts-intersection-antitrust-law-algorithmic

Park, D., & Ryu, D. (2022). Supply Chain Ethics and Transparency: An agent-based model approach with Q-learning agents. *Managerial and Decision Economics*, 1–7. https://doi.org/https://doi.org/10.1002/mde.3597

Sanchez-Cartas, J., & Katsamakas, E. (2022). Artificial Intelligence, Algorithmic Competition and Market Structures. *IEEE Access*. https://ieeexplore.ieee.org/abstract/document/9684893/

Takahashi, H., Nishino, N., & Takenaka, T. (2018). Multi-agent Simulation for the Manufacturer's Decision Making in Sharing Markets. *Procedia CIRP*, *67*, 546–551. https://doi.org/10.1016/J.PROCIR.2017.12.258

Tam, A. (2021). A Gentle Introduction to Particle Swarm Optimization. https://machinelearningmastery.com/a-gentle-introduction-to-particle-swarm-optimization/

Vainer, J., & Kukacka, J. (2021). Nash Q-learning agents in Hotelling's model: Reestablishing equilibrium. *Communications in Nonlinear Science and Numerical Simulation*, *99*, 105805. https://doi.org/10.1016/j.cnsns.2021.105805

Waltman, L., & Kaymak, U. (2008). Q-learning agents in a Cournot oligopoly model. *Journal of Economic Dynamics and Control*, *32*(10), 3275–3293. https://www.sciencedirect.com/science/article/pii/S0165188908000183

Wang, H., & Yu, Y. (2016). Exploring multi-action relationship in reinforcement learning. *Springer*, 574–587. https://doi.org/10.1007/978-3-319-42911-3{_}48

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3-4), 279–292. https://doi.org/10.1007/BF00992698

Watkins, C. (1989). *Learning from delayed rewards* (Doctoral dissertation). King's College. https://www.academia.edu/download/50360235/Learning_from_delayed_rewards_20161116-28282-v2pwvq.pdf

Wieting, M., & Sapi, G. (2021). Algorithms in the Marketplace: An Empirical Analysis of Automated Pricing in E-Commerce. *papers.ssrn.com*. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3945137

Xie, M., & Chen, J. (2004). Studies on horizontal competition among homogenous retailers through agent-based simulation. *Journal of Systems Science and Systems Engineering 2004 13:4*, *13*(4), 490–505. https://doi.org/10.1007/S11518-006-0178-7

ANNEX

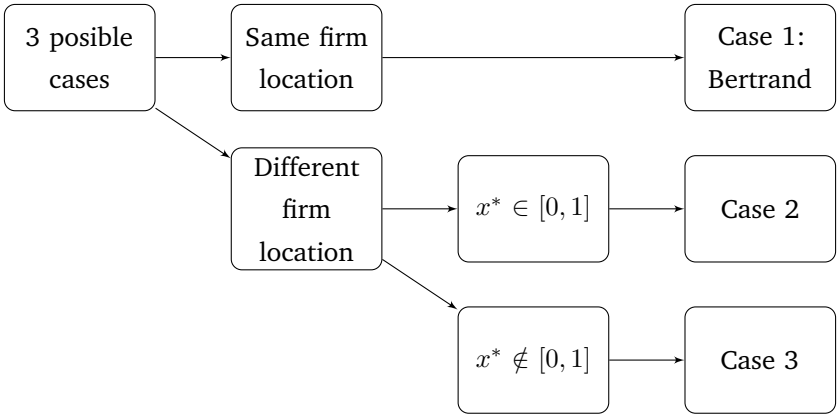


FIGURE 1: DEMAND CASES

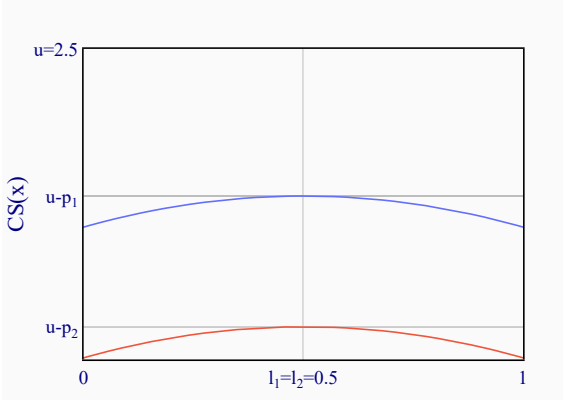


FIGURE 2: DEMAND CASE 1: BERTRAND

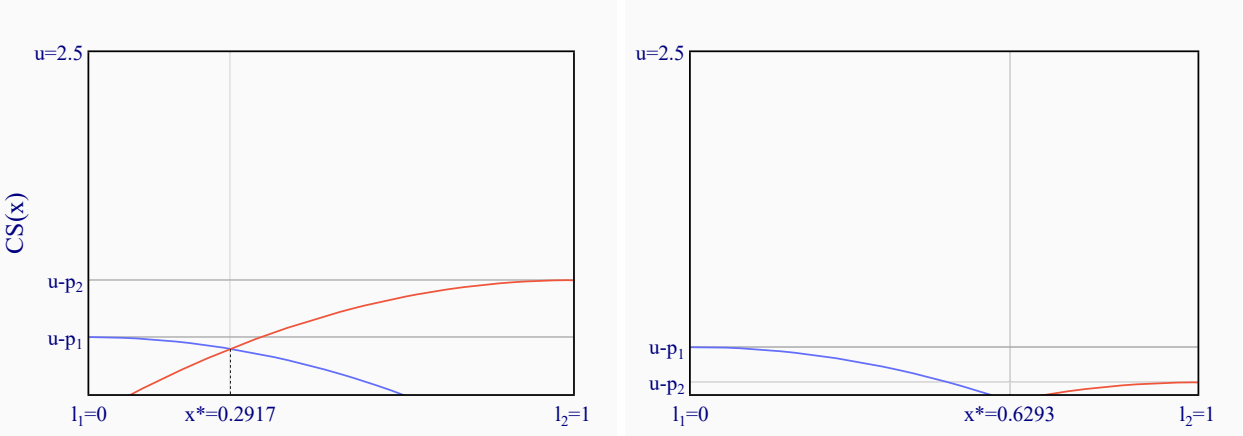


FIGURE 3: DEMAND CASE 2

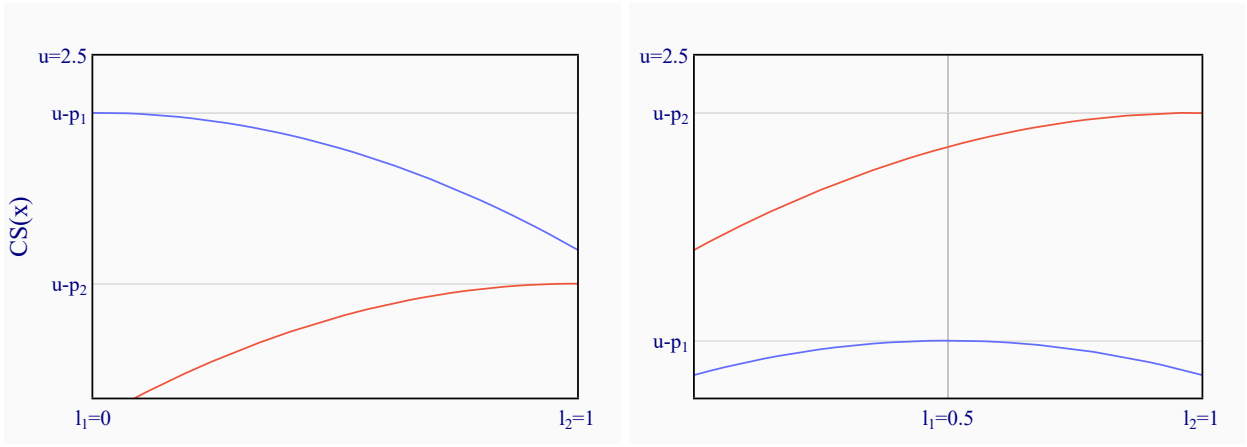


FIGURE 4: DEMAND CASE 3

	Location choice problem	Price choice problem
Q-matrix	$Q_{l,i}$	$Q_{p,i}$
Action space	$\mathcal{A}_l = \mathcal{L} \times \mathcal{P}$	$A_p = \mathcal{P}$
Action	$a_l \in \mathcal{A}_l$	$a_p \in \mathcal{A}_p$
State space	$\mathcal{S}_l = \{(\mathcal{L} \times \mathcal{P}), (\mathcal{L} \times \mathcal{P})\}$	$\mathcal{S}_p = \mathcal{L} \times \mathcal{L}$
State	$s_l \in \mathcal{S}_l$	$s_p \in \mathcal{S}_p$

TABLE 1: NOTATION USED IN EACH PROBLEM

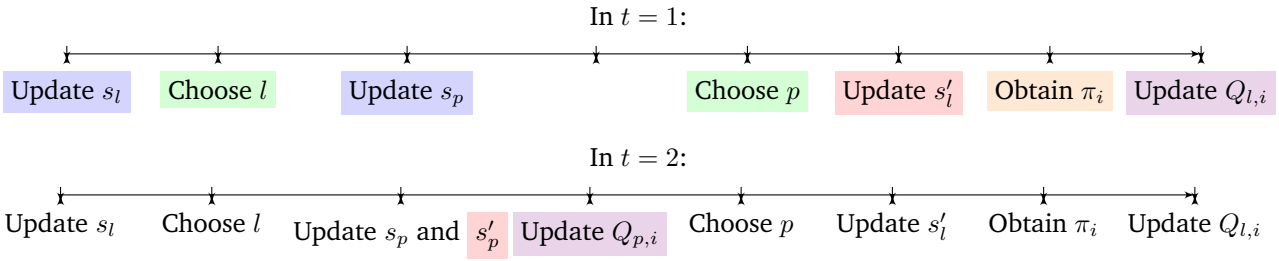


FIGURE 5: Q-MATRIX UPDATING TIMELINE

Algorithm 1 Pseudocode Q-Learning algorithms in a Hotelling model

Data: Setup parameters value

Initialization : Q_l and Q_p matrices

Initialization : Initial state $s_0 = \{(\text{location}_i, \text{price}_i); (\text{location}_j, \text{price}_j)\}$

for $t \leftarrow 1$ **to** T **do**

Set current state $s_l = \{a_{l,i}, a_{l,j}\} = \{(l_i, p_i), (l_j, p_j)\}$

Flip a coin = {random, optimal} with probabilities $\{\epsilon,\ 1-\epsilon\}$

Location choice problem:

if *Coin = random* **then**

Action: $a_i = \{l_i, p_i\}$with uniform distribution

else if *Coin = optimal* **then**

for all l in available locations

Choose p_i such that $a_{l,i} = \{l_i, p_i\} = \arg\max(Q_{l,i}(s_l, a_{l,i})$

Compare all the $\arg\max(Q_{l,i}((s_l, a_{l,i}))$ and choose the l_i with the $\max Q_{l,i}$. Call this l_i as l_i^*

end

end

Update current price state for the current price $s_p = \{l_i, l_j\}$

Update tomorrow price state for the past price $s'_p = \{l_i, l_j\}$

if *t = 1* **then**

continue

else

Update Q_p matrix for action $a_{p,i} = p_i$:

$$Q_{p,i}\left(s_p, a_{p,i}\right) \leftarrow (1-\alpha) \cdot Q_{p,i}\left(s_p, a_{p,i}\right) + \alpha \cdot \left[\pi(s_p, a_{p,i}) + \delta \max_{a'_p \in \mathcal{A}_p} Q_{p,i}\left(s'_p, a'_p\right)\right] \tag{38}$$

end

end

Price choice problem:

if *Coin = random* **then**

You already have the price

else if *Coin = optimal* **then**

Choose p_i such that $a_{p,i} = p_i = \arg\max(Q_{p,i}(s_p, a_{p,i}))$

end

end

Update tomorrow location state $s'_l = \{a_{l,i}, a_{l,j}\} = \{(l_i, p_i), (l_j, p_j)\}$

Obtain the profit $\pi_i(s'_l) = \pi(l_i, p_i, l_j, p_j)$

`/* Remember that this profit is the one that will be used in the` Q_p `matrix updating in the next iteration` $\pi_i(s'_l) = \pi_i(s_p, a_{p,i})$ `*/`

Update $Q_{l,i}$ matrix:

$$Q_{l,i}\left(s_l, a_{l,i}\right) \leftarrow (1-\alpha) \cdot Q_{l,i}\left(s_l, a_{l,i}\right) + \alpha \cdot \left[\pi(s'_l) + \delta \max_{a'_l \in \mathcal{A}_l} Q_{l,i}\left(s'_l, a'_l\right)\right] \tag{39}$$

end

TABLE 2: TRANSPORTATION COST FOR EVERY COMBINATION OF FIRMS' LOCATION

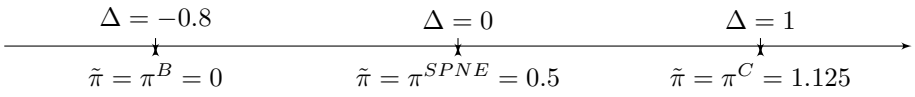
(A) $N_l = 3$

Locations	$TC = f_1(.) + f_2(.)$
$\{0.0, 0.0\}$	$\frac{1}{3} \approx 0.333$
$\{0.0, 0.5\}$	$\frac{5}{96} \approx 0.052$
$\{0.0, 1.0\}$	$\frac{1}{12} \approx 0.083$
$\{0.5, 0.5\}$	$\frac{1}{12} \approx 0.083$
$\{0.5, 1.0\}$	$\frac{5}{96} \approx 0.052$
$\{1.0, 1.0\}$	$\frac{1}{3} \approx 0.333$

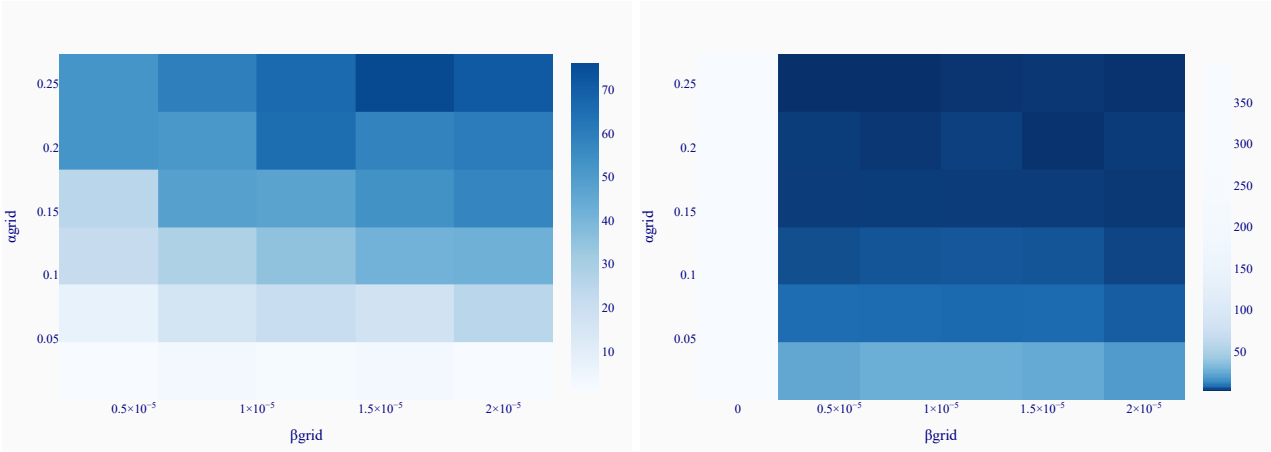
(B) $N_l = 5$

Locations	$TC = f_1(.) + f_2(.)$
$\{0.0, 0.25\}$	$\frac{109}{768} \approx 0.142$
$\{0.0, 0.75\}$	$\frac{31}{768} \approx 0.040$
$\{0.25, 0.25\}$	$\frac{7}{48} \approx 0.146$
$\{0.25, 0.5\}$	$\frac{37}{768} \approx 0.048$
$\{0.25, 0.75\}$	$\frac{1}{48} \approx 0.021$
$\{0.25, 1.0\}$	$\frac{31}{768} \approx 0.040$
$\{0.5, 0.75\}$	$\frac{37}{768} \approx 0.048$
$\{0.75, 0.75\}$	$\frac{7}{48} \approx 0.146$
$\{0.75, 1.0\}$	$\frac{109}{768} \approx 0.142$

Notes: Each of the values of the tables was calculated following these steps. First, determine each firm's demand. The second step is divided into two. To calculate the total transportation cost for all the consumers at the left of firm i, evaluate $\int (l_i - x)^2 dx = -(l_i - x)^3/3$ over the limits of the left-side firm i's demand and multiply it with the consumer density: $1/(1-0) = 1$. For the right-side firm i's demand, evaluate $\int (x - l_i)^2 dx = (x - l_i)^3/3$ over the corresponding right-side limits and multiply it with the consumer density: $1/(1-0) = 1$. Third, the aggregated transportation cost for firm i ($f_i(.)$) is simply the sum of the total transportation cost for the two demand sides. Finally, the total transportation cost (TC) is the sum of each firm's aggregated transportation cost.



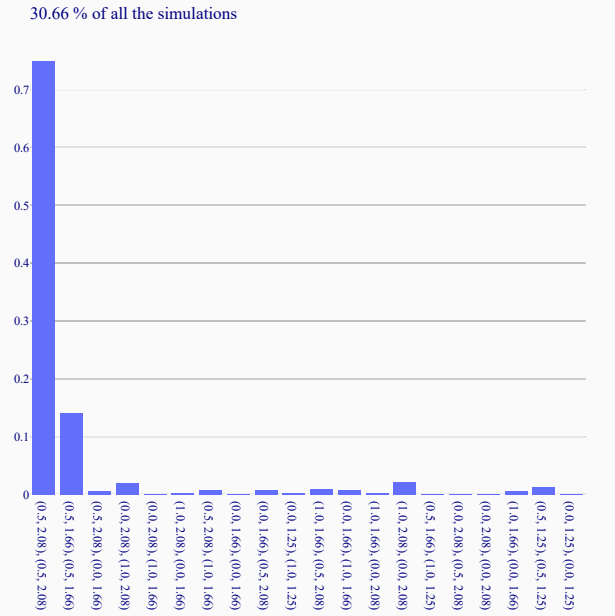
FIGUE 6: Δ GRID



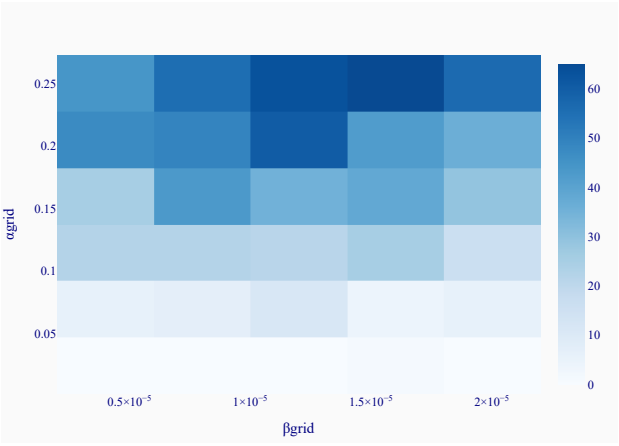
(A) %SIMULATION THAT CONVERGES TO ACTION CYCLE OF LENGTH ONE

(B) AVERAGE ACTION CYCLE LENGTH IN NO-UNIT ACTION CYCLES

FIGURE 7

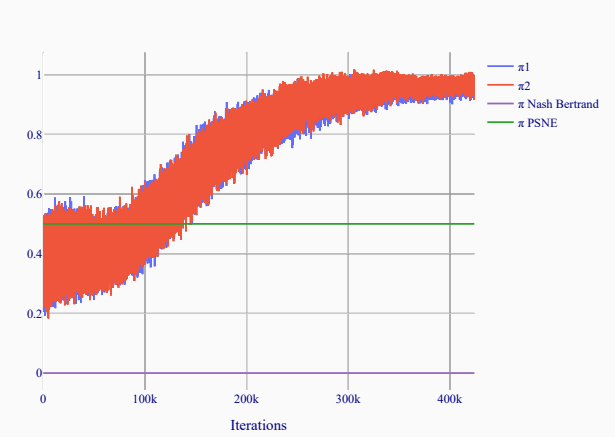


(A) Actions played in cycles of length one

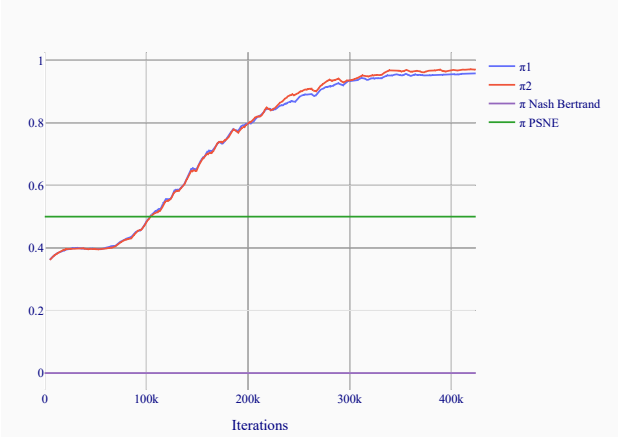


(B) %simulation that converges to distribution's mode

Figure 8

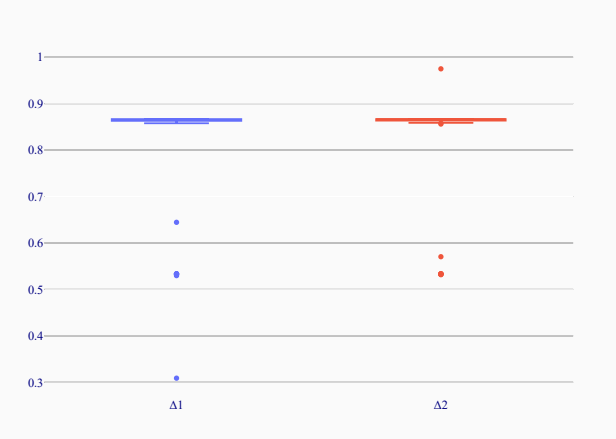


(A) Profit evolution

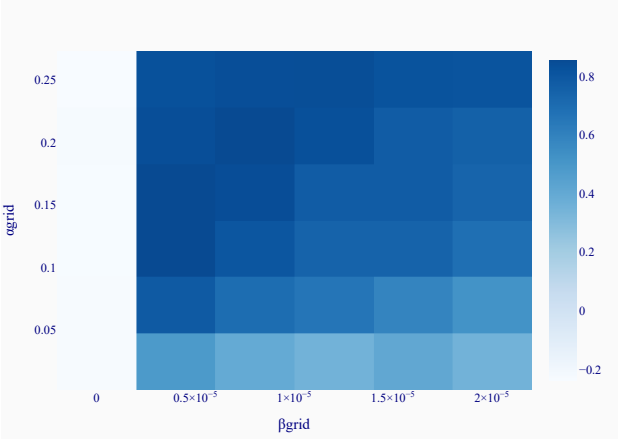


(B) SMA profit evolution

Figure 9: Profit evolution

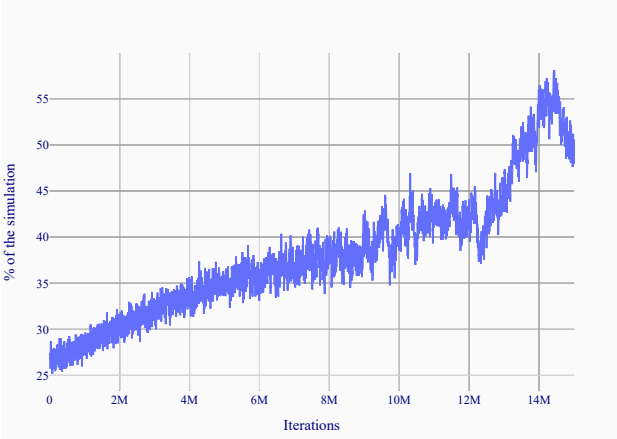


(A) Representative simulation

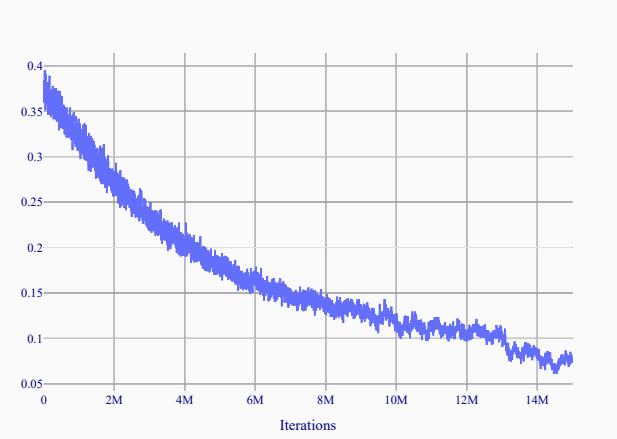


(B) As a function of α and β

Figure 10: Extra profit gain

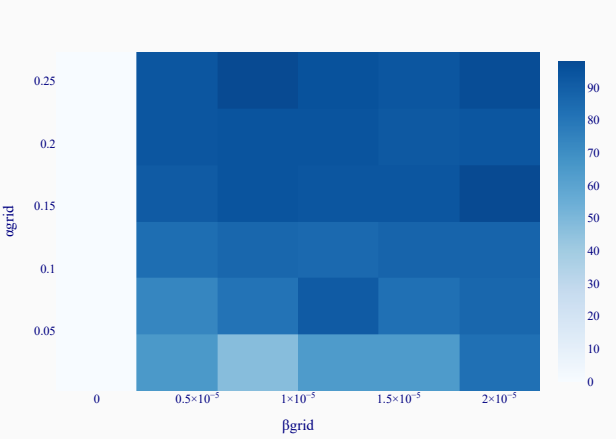


(A) NE evolution

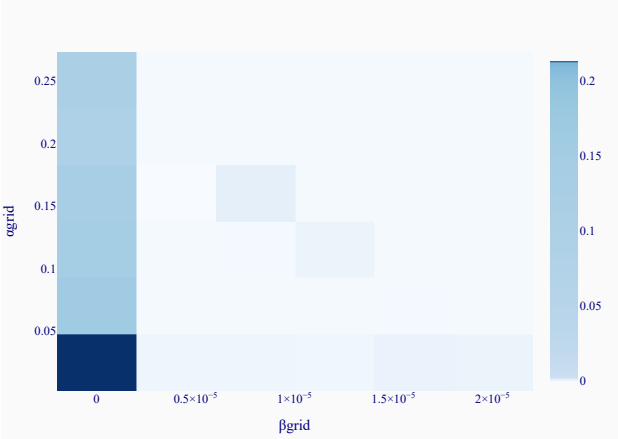


(B) Aggregated Q-loss evolution

Figure 11

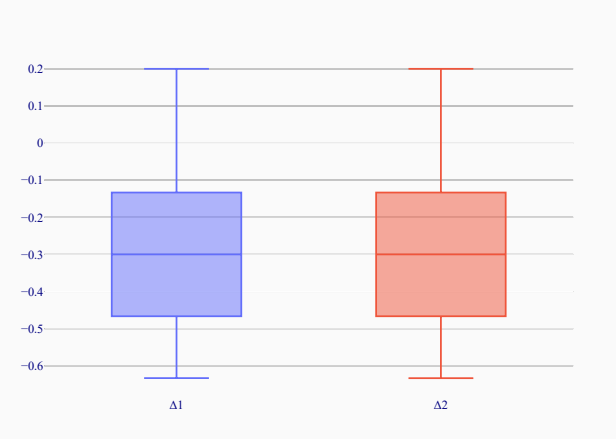


(A) %simulation that converges to a NE

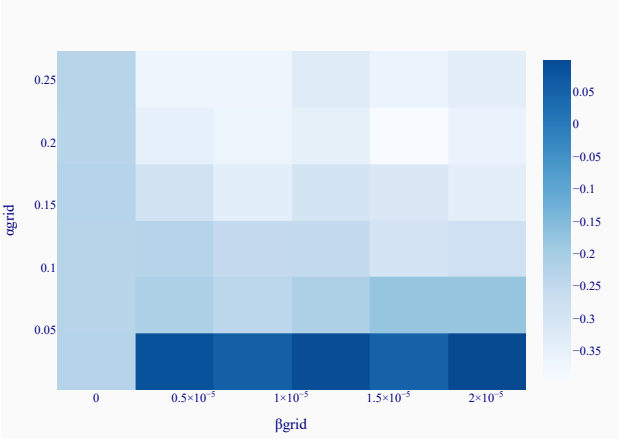


(B) Aggregated Q-loss

Figure 12

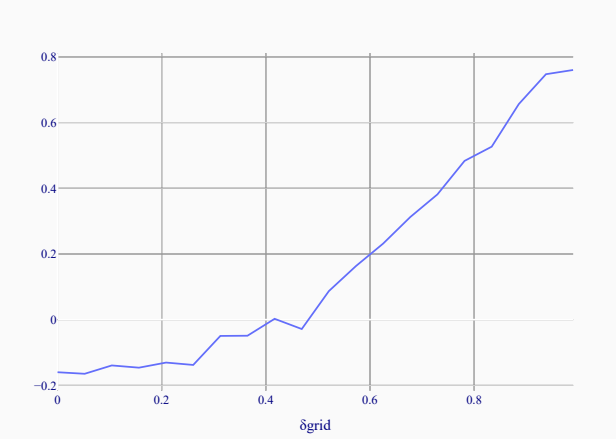


(A) Representative simulation

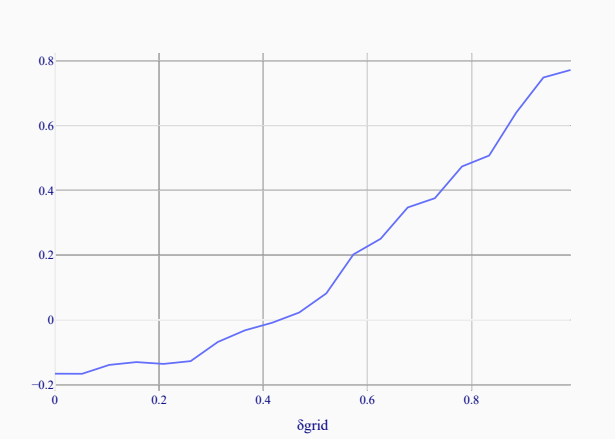


(B) As a function of α and β

Figure 13: No memory - extra profit gain



(A) Firm 1



(B) Firm 2

Figure 14: Average extra profit gain as a function of δ

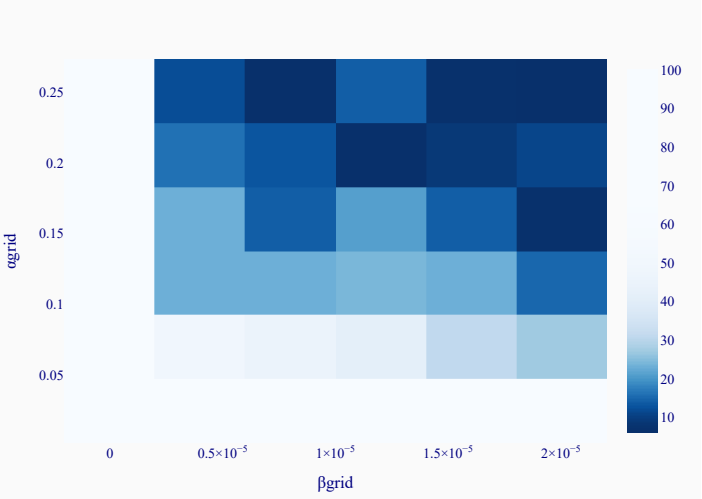
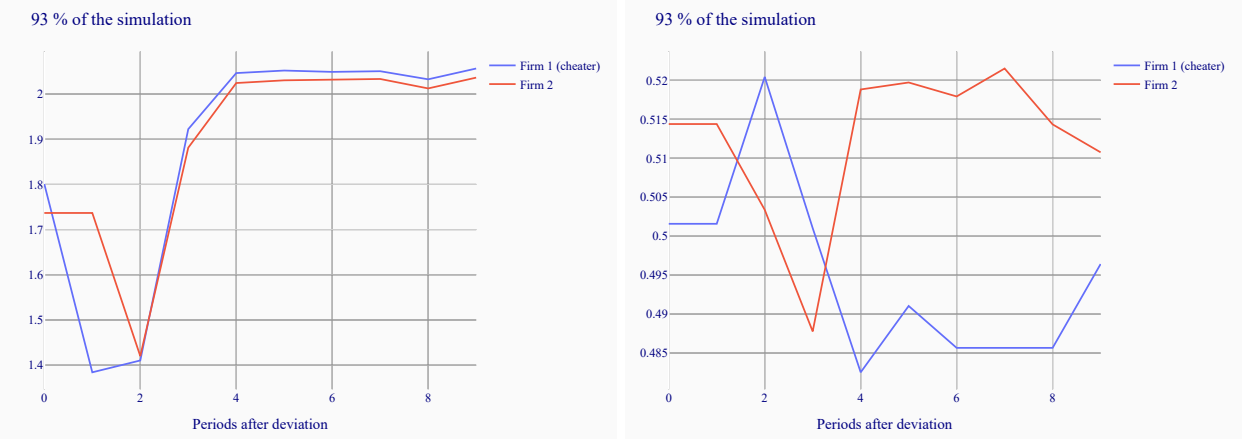
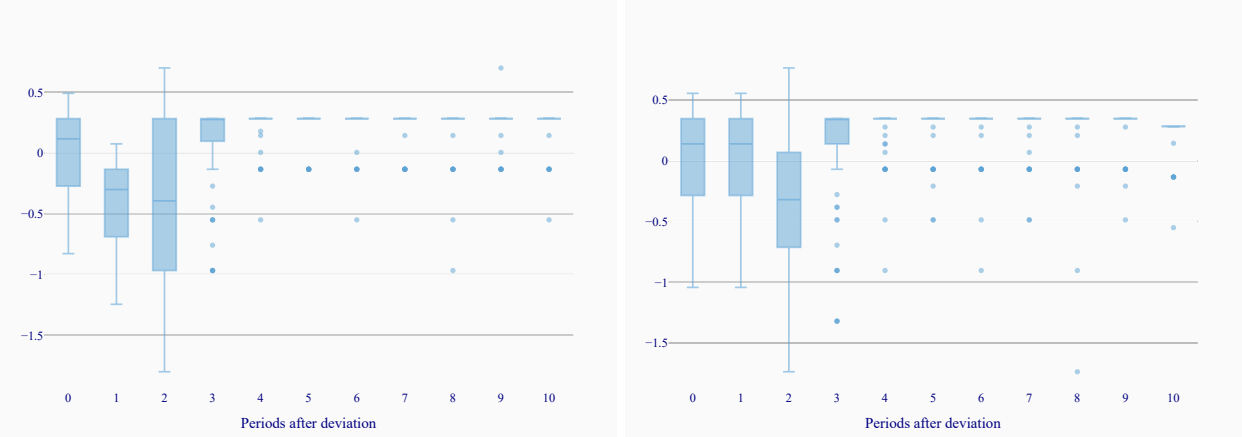


Figure 15: Fraction of the simulations with $p = 0.0 \in$ action cycle



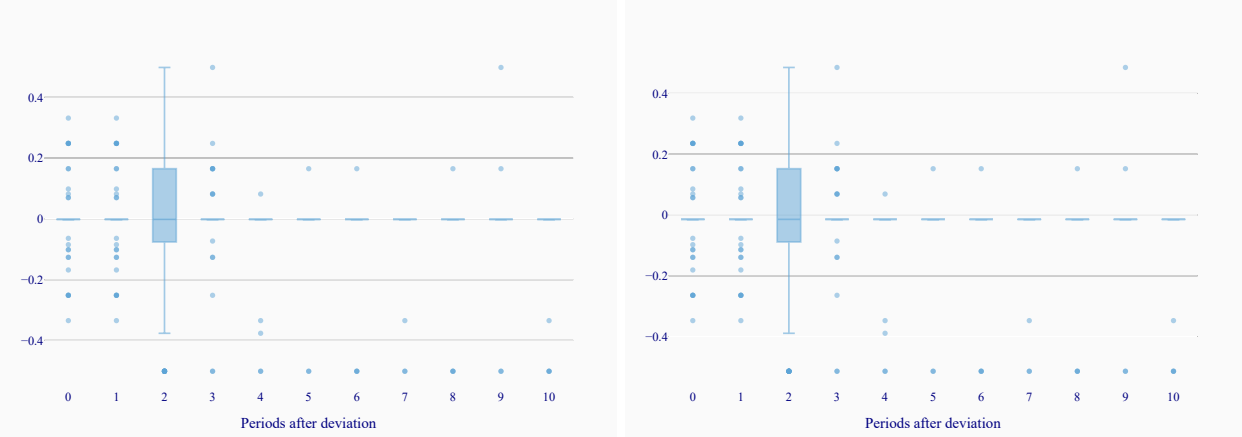
(A) Price (B) Location

Figure 16: Firms response after deviation



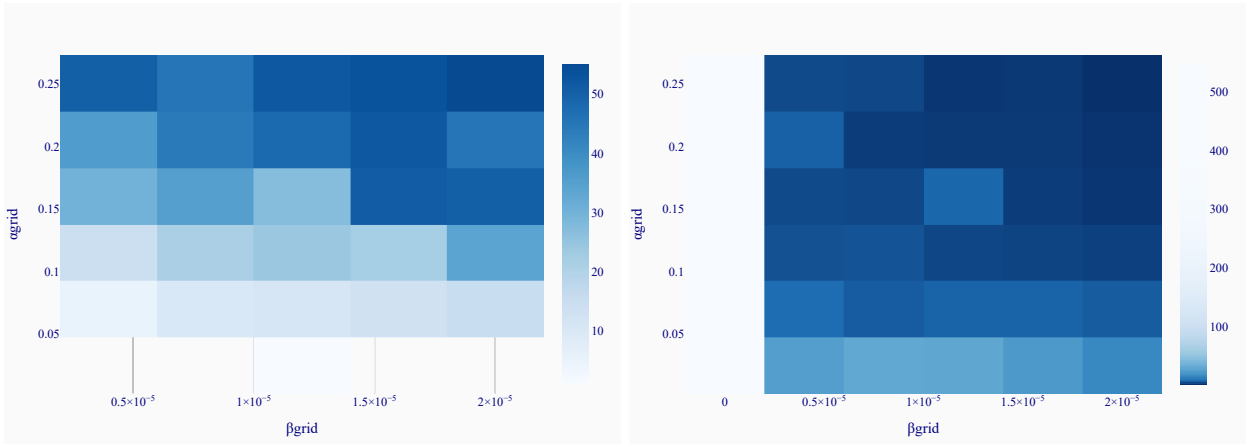
(A) Firm 1 (B) Firm 2

Figure 17: Boxplot price response



(A) Firm 1 (B) Firm 2

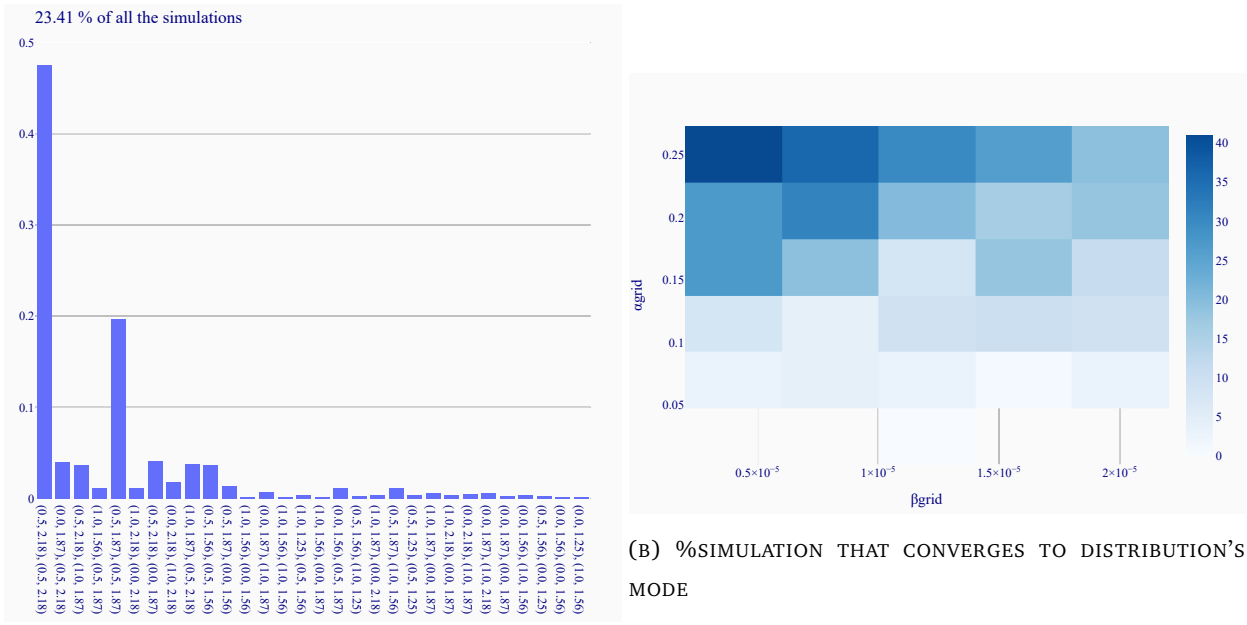
Figure 18: Boxplot location response



(A) %SIMULATION THAT CONVERGES TO ACTION CYCLE OF LENGTH ONE

(B) AVERAGE ACTION CYCLE LENGTH IN NO-UNIT ACTION CYCLES

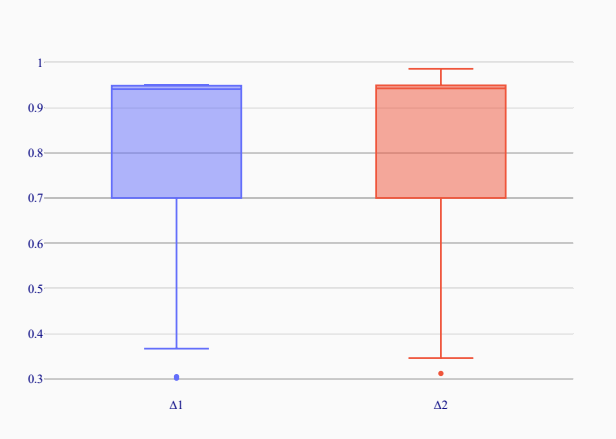
FIGURE 19



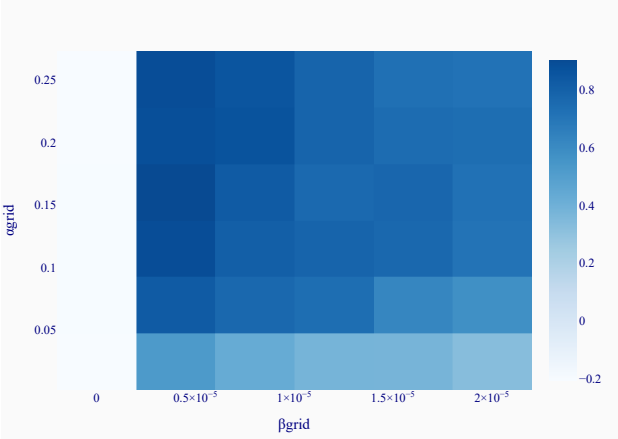
(A) ACTIONS PLAYED IN CYCLES OF LENGTH ONE

(B) %SIMULATION THAT CONVERGES TO DISTRIBUTION'S MODE

FIGURE 20

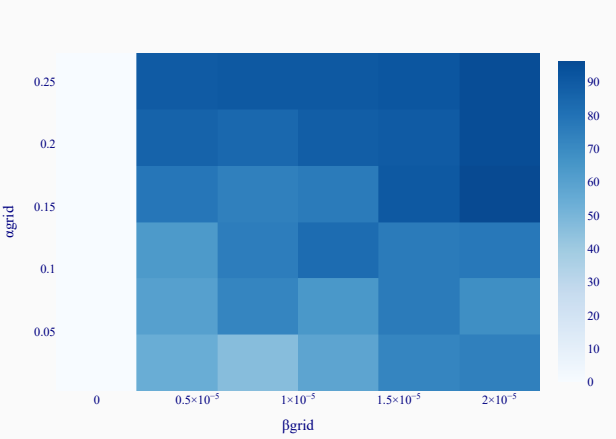


(A) Representative simulation

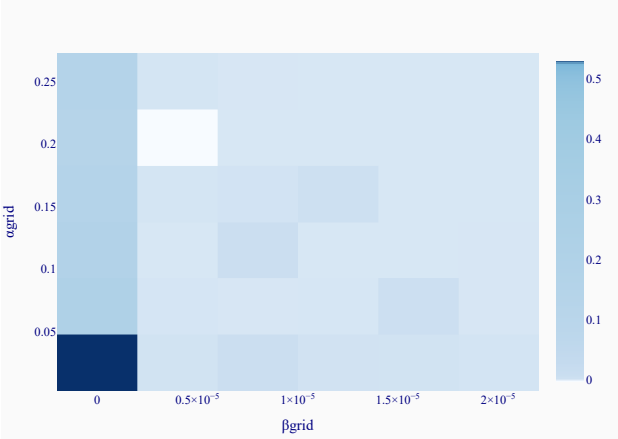


(B) As a function of α and β

Figure 21: Extra profit gain



(A) %simulation that converges to a NE



(B) Aggregated Q-loss

Figure 22

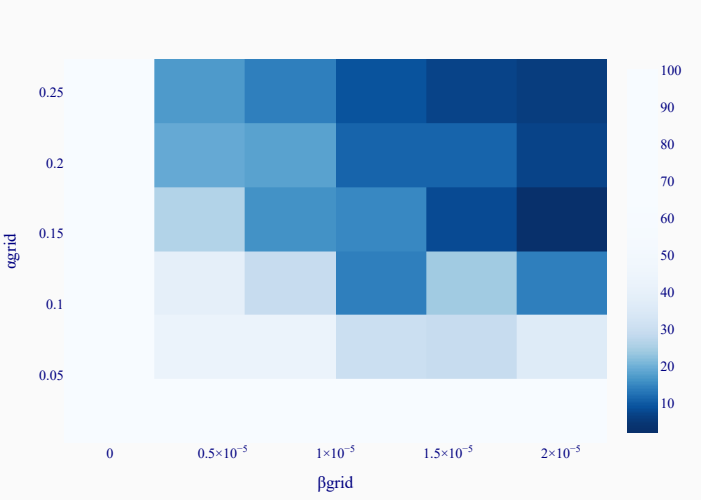
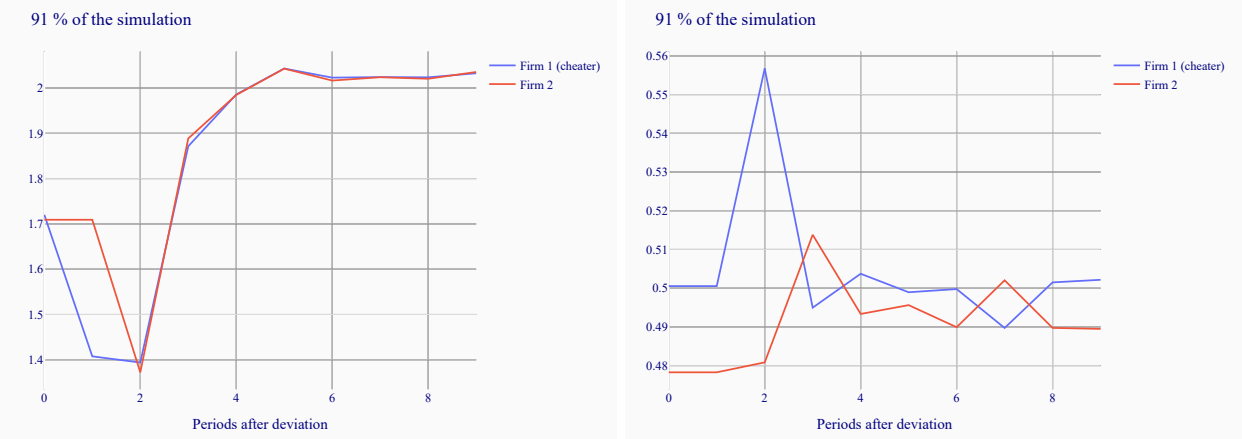
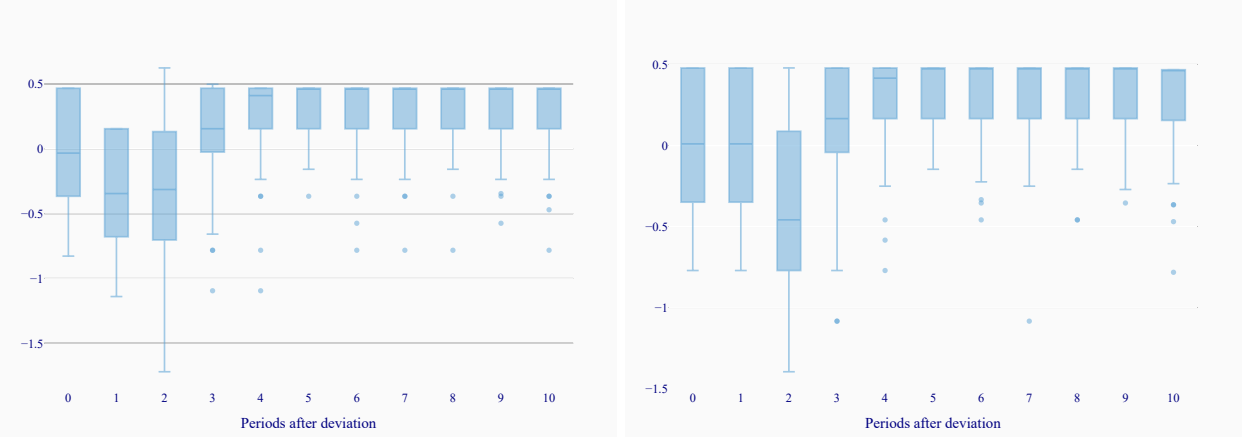


Figure 23: Fraction of the simulations with $p = 0.0 \in$ action cycle



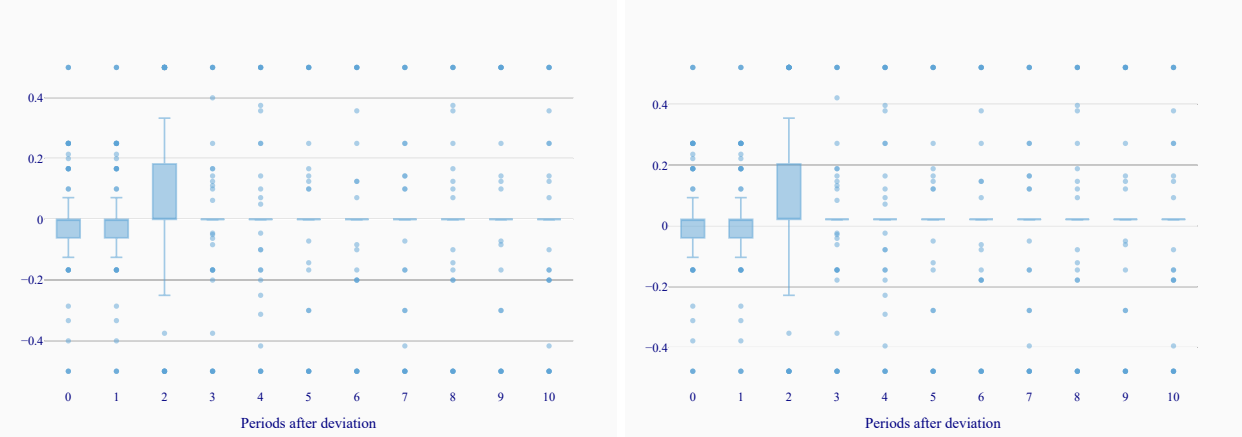
(A) Price (B) Location

Figure 24: Firms response after deviation



(A) Firm 1 (B) Firm 2

Figure 25: Boxplot price response



(A) Firm 1 (B) Firm 2

Figure 26: Boxplot location response

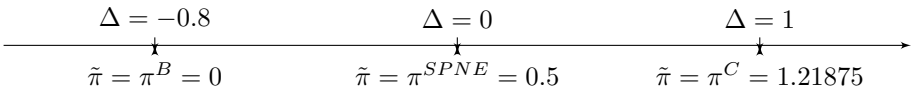
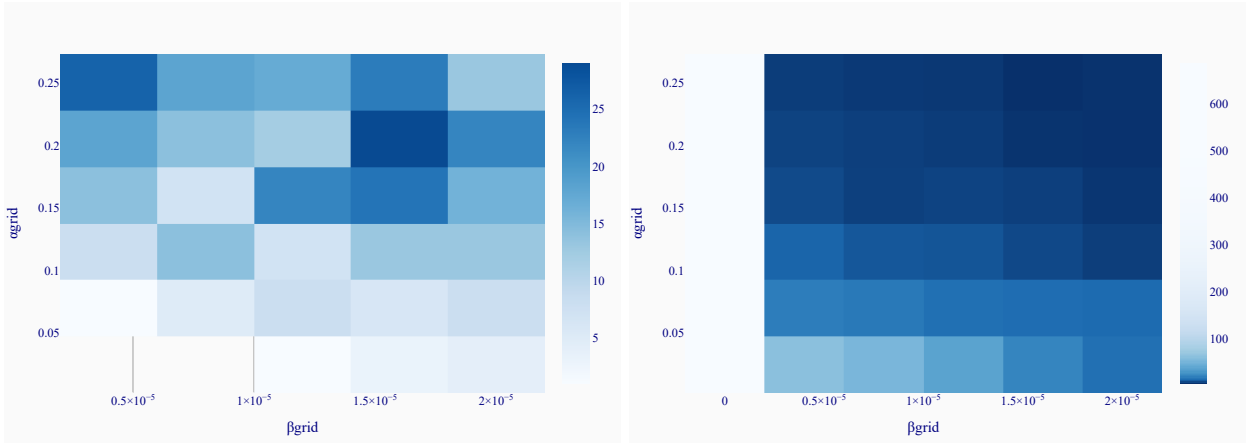


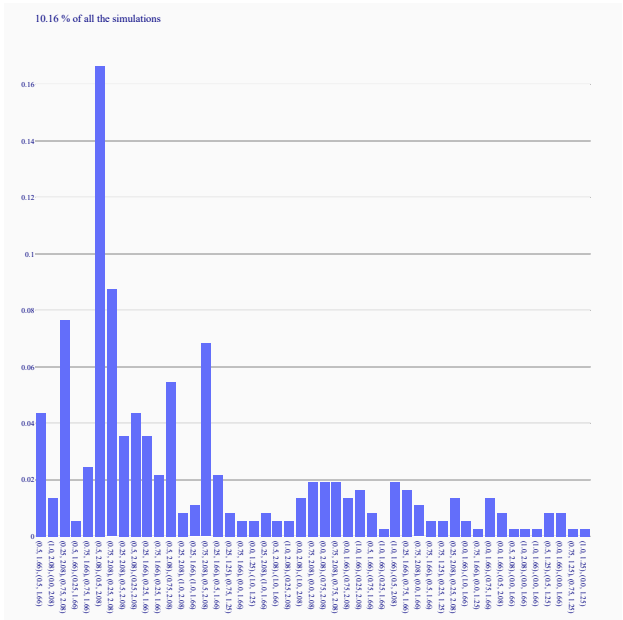
FIGURE 27: Δ GRID WITH $N_l = 5$



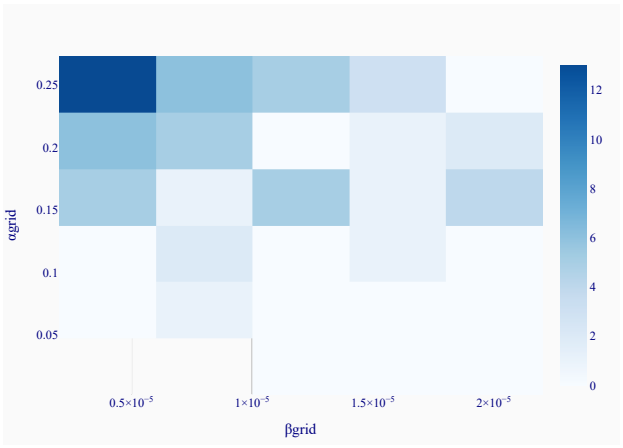
(A) %SIMULATION THAT CONVERGES TO ACTION CYCLE OF LENGTH ONE

(B) AVERAGE ACTION CYCLE LENGTH IN NO-UNIT ACTION CYCLES

FIGURE 28

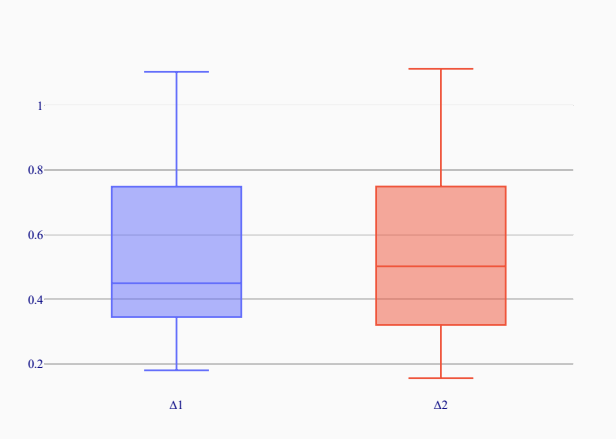


(A) ACTIONS PLAYED IN CYCLES OF LENGTH ONE

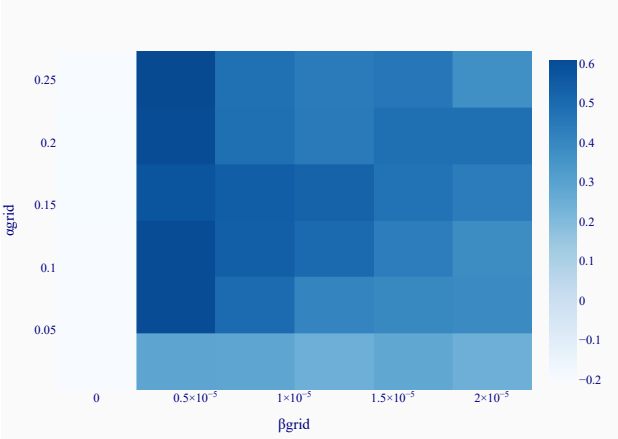


(B) %SIMULATION THAT CONVERGES TO DISTRIBUTION'S MODE

FIGURE 29

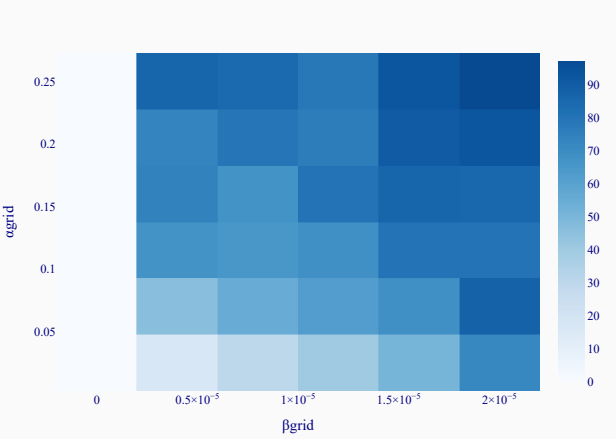


(A) Representative simulation

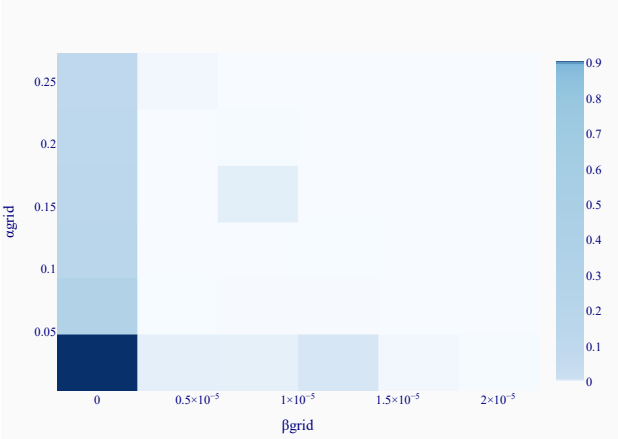


(B) As a function of α and β

Figure 30: Extra profit gain



(A) %simulation that converges to a NE



(B) Aggregated Q-loss

Figure 31

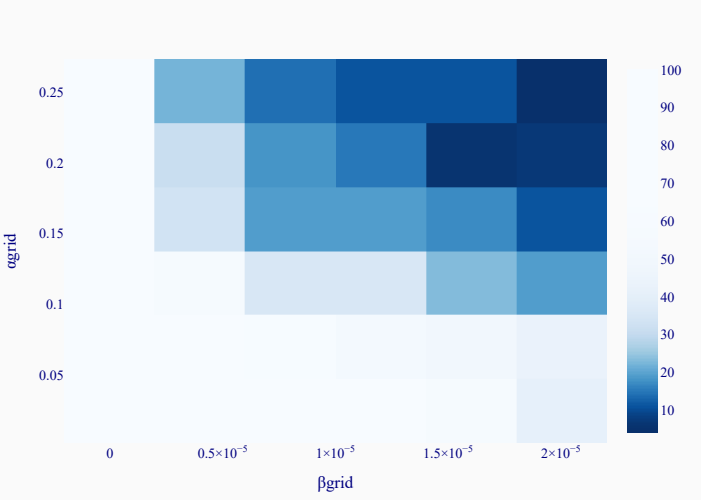
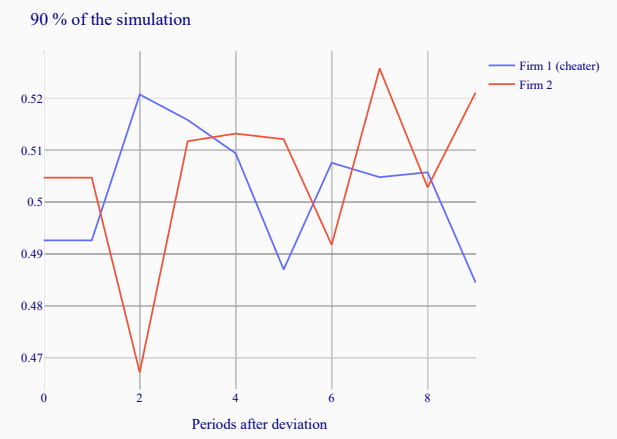


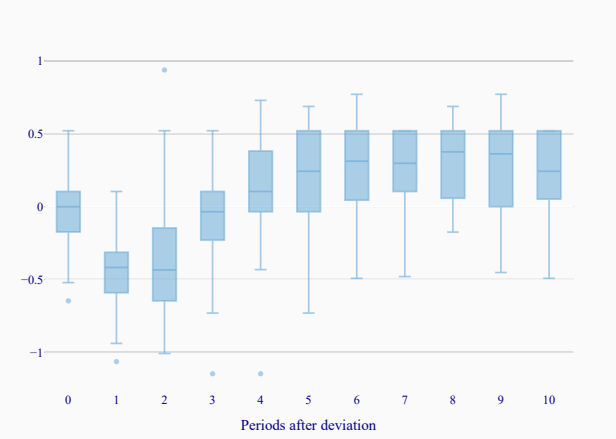
Figure 32: Fraction of the simulations with $p = 0.0 \in$ action cycle

(A) Price

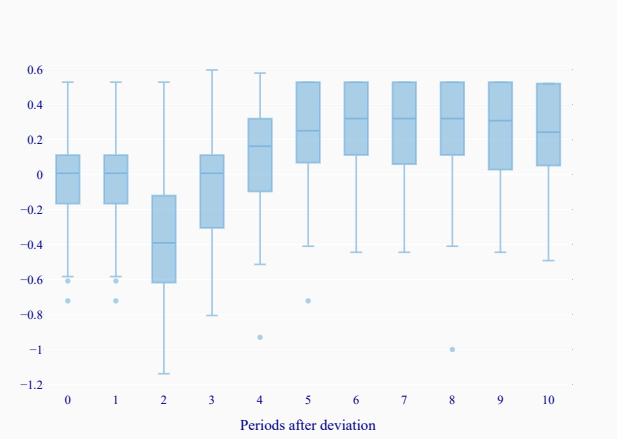


(B) Location

Figure 33: Firms response after deviation

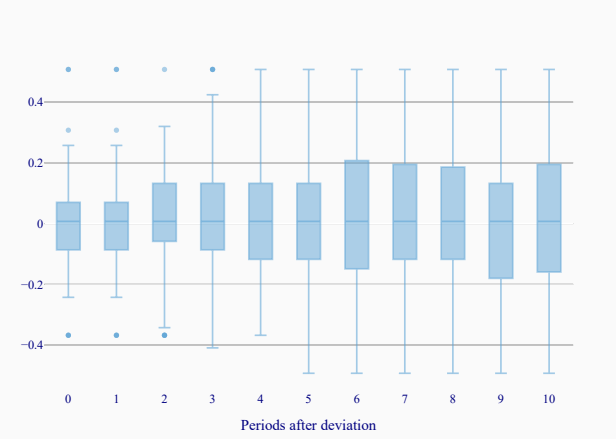


(A) Firm 1

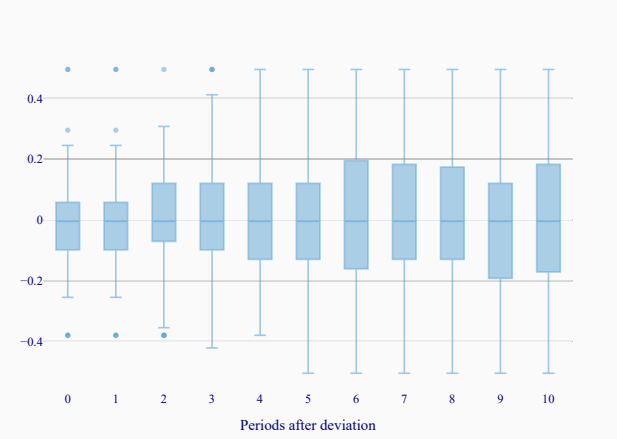


(B) Firm 2

Figure 34: Boxplot price response

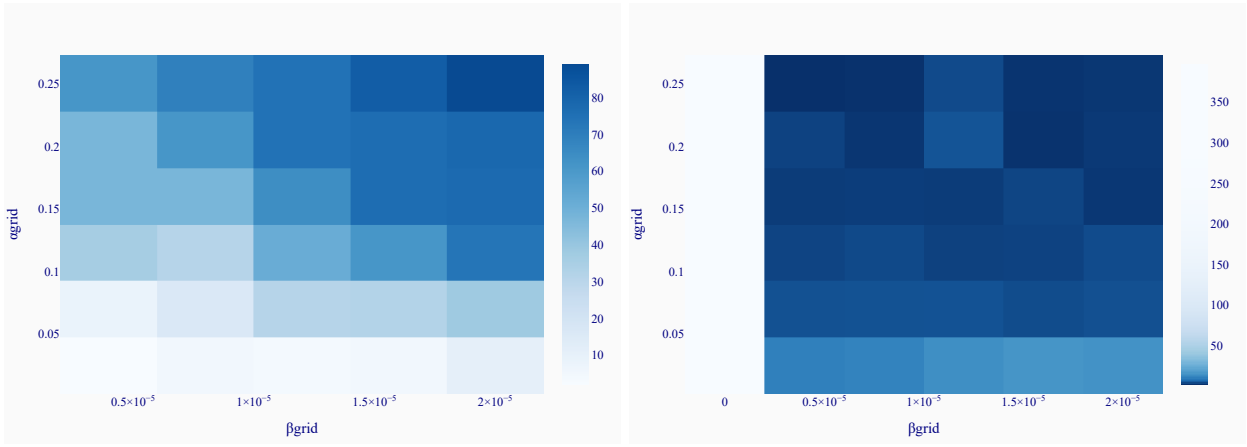


(A) Firm 1



(B) Firm 2

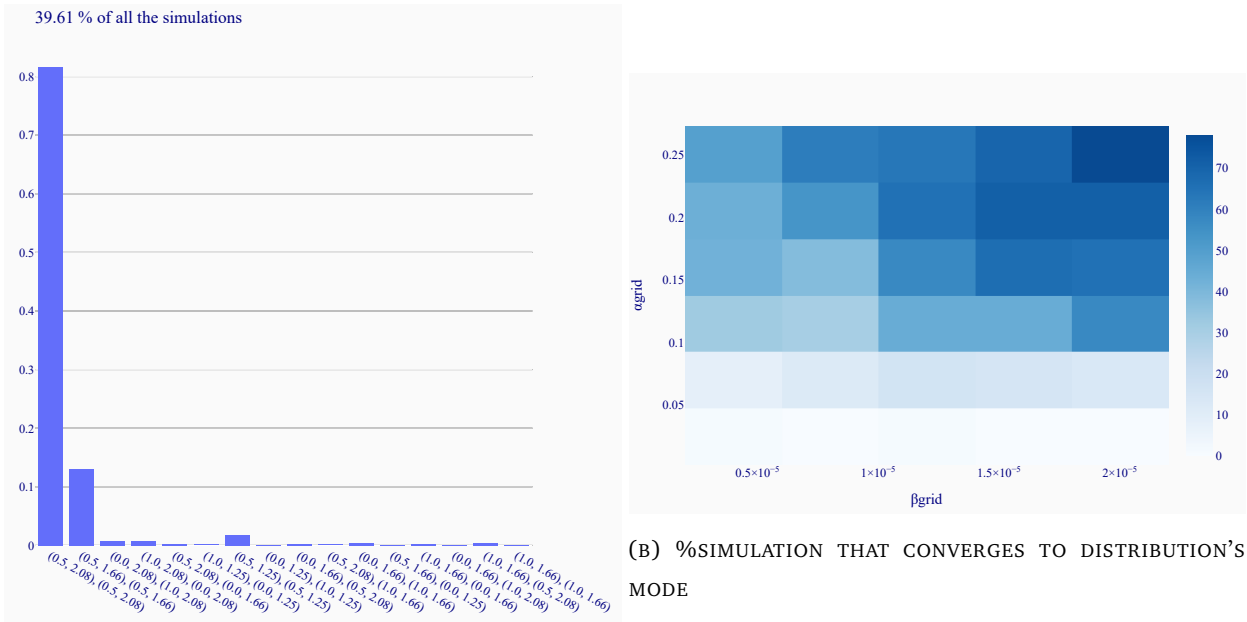
Figure 35: Boxplot location response



(A) %SIMULATION THAT CONVERGES TO ACTION CYCLE OF LENGTH ONE

(B) AVERAGE ACTION CYCLE LENGTH IN NO-UNIT ACTION CYCLES

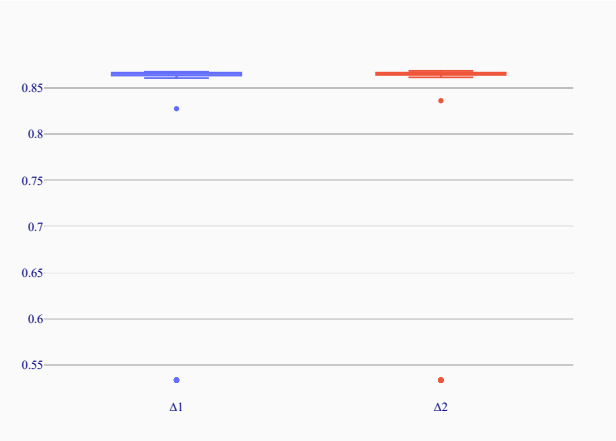
FIGURE 36



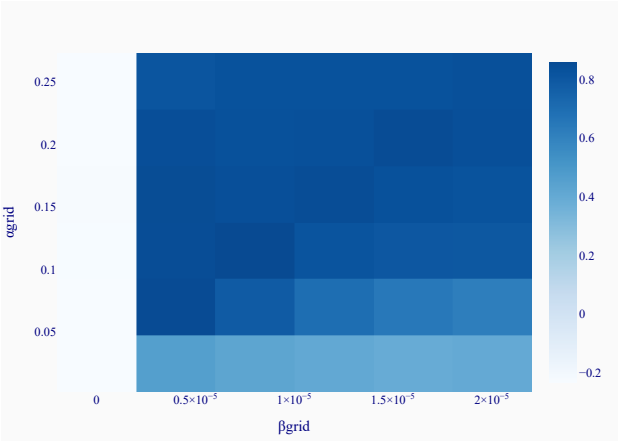
(A) ACTIONS PLAYED IN CYCLES OF LENGTH ONE

(B) %SIMULATION THAT CONVERGES TO DISTRIBUTION'S MODE

FIGURE 37

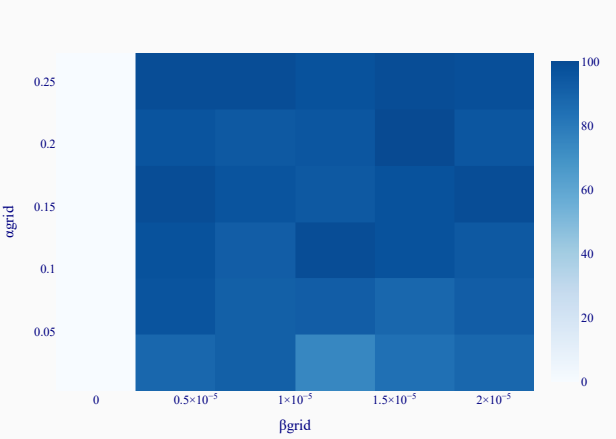


(A) Representative simulation

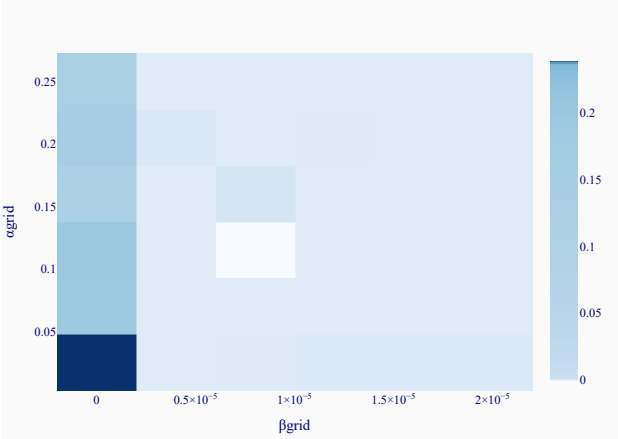


(B) As a function of α and β

Figure 38: Extra profit gain



(A) %simulation that converges to a NE



(B) Aggregated Q-loss

Figure 39

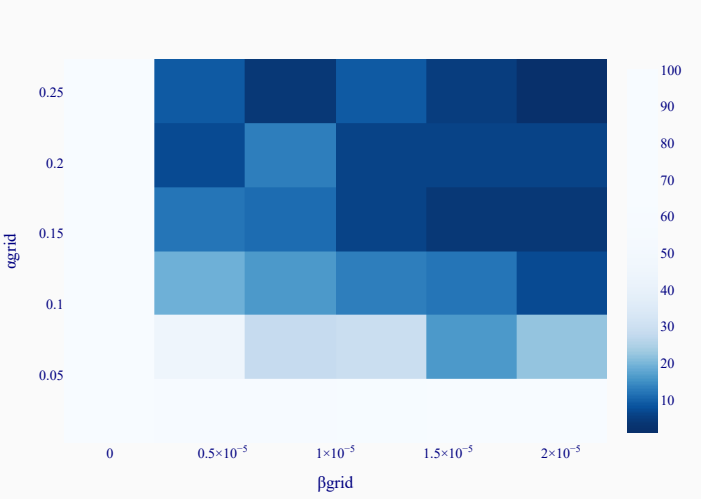
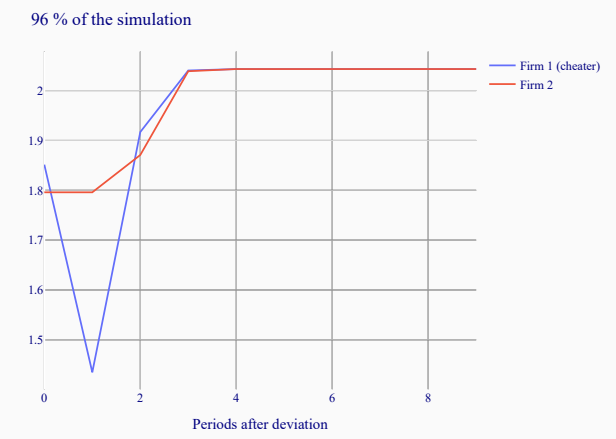
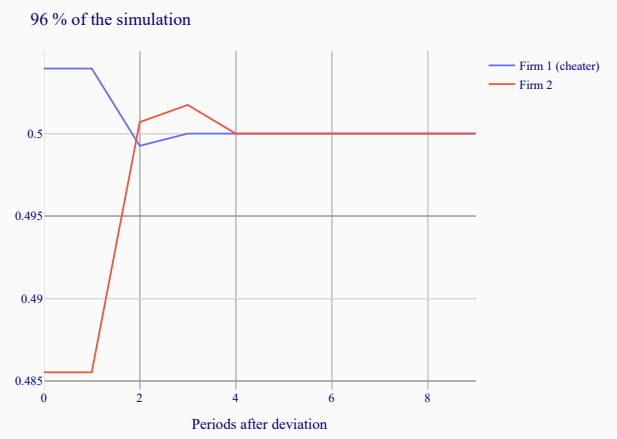


Figure 40: Fraction of the simulations with $p = 0.0 \in$ action cycle

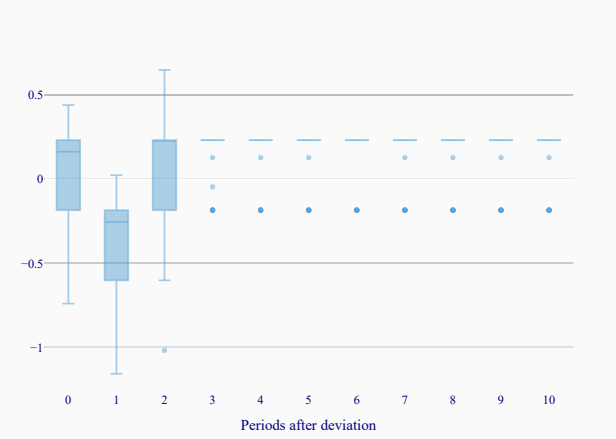


(A) Price

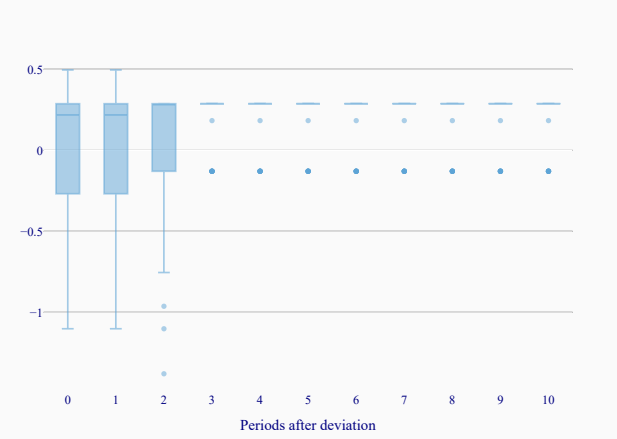


(B) Location

Figure 41: Firms response after deviation

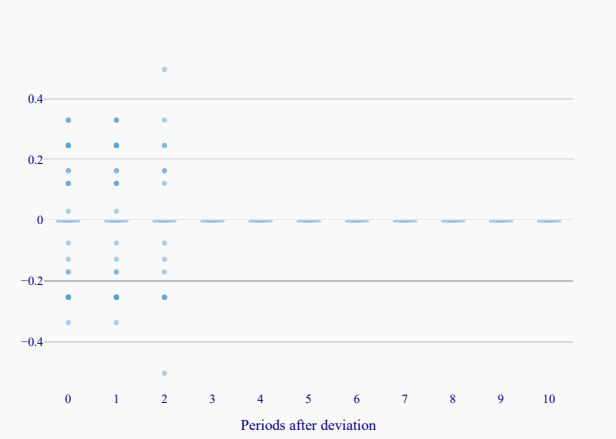


(A) Firm 1

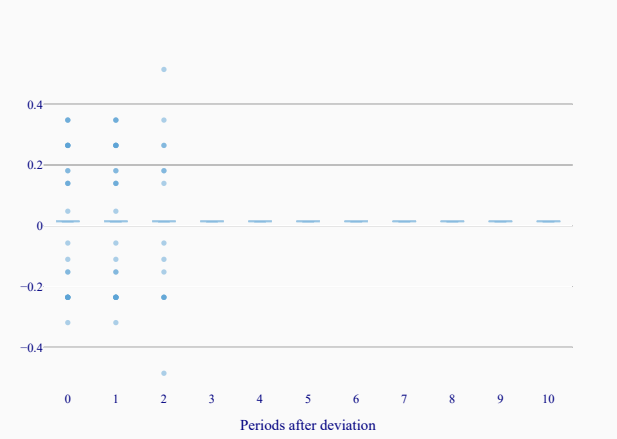


(B) Firm 2

Figure 42: Boxplot price response



(A) Firm 1



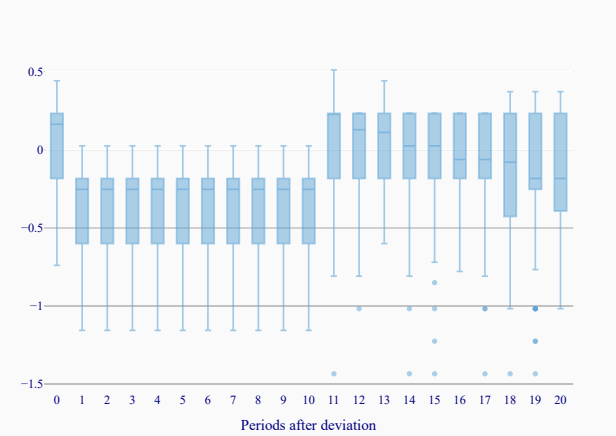
(B) Firm 2

Figure 43: Boxplot location response

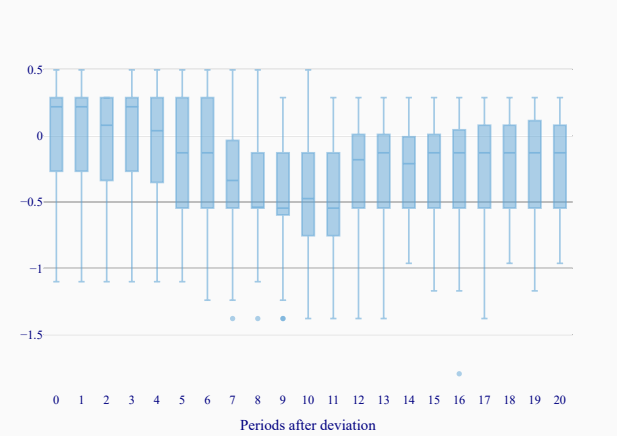
(A) Price

(B) Location

Figure 44: Firms response after deviation

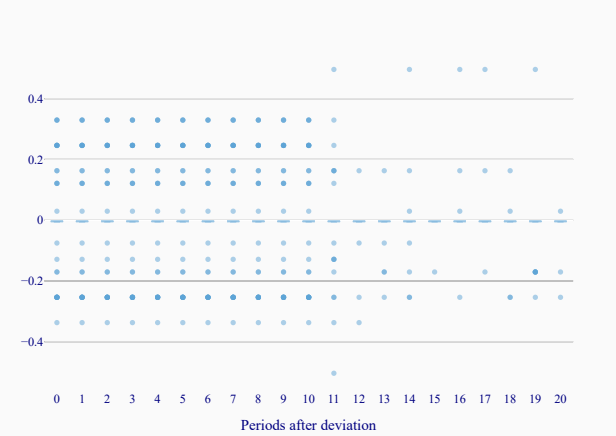


(A) Firm 1

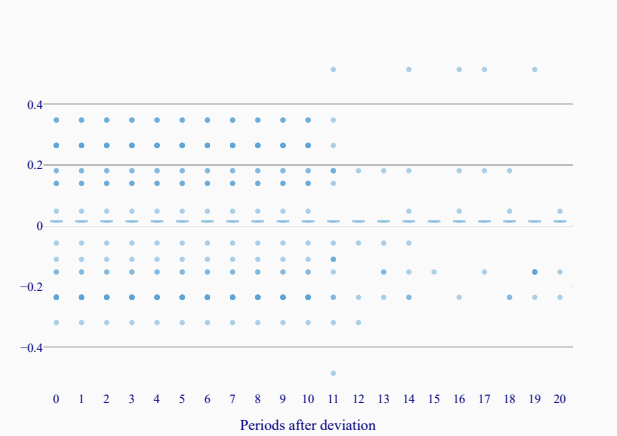


(B) Firm 2

Figure 45: Boxplot price response



(A) Firm 1



(B) Firm 2

Figure 46: Boxplot location response