Forside

Eksamensinformation
AØKK08001E - Specialeafhandling - Kontrakt:138175
(Julius Løve Fischer)

Besvarelsen afleveres af
Julius Løve Fischer
nbm205@alumni.ku.dk

Eksamensadministratorer
Daniel Sune Sørensen
dss@samf.ku.dk
+4535328466

Bedømmere
Anders Munk-Nielsen
Eksaminator
amn@econ.ku.dk
+4535324426

Cédric Schneider
Censor
csc.eco@cbs.dk

Besvarelsesinformationer

Titel: Algorithmic Pricing Collusion using Reinforcement Learning
Titel, engelsk: Algorithmic Pricing Collusion using Reinforcement Learning
Antal normalsider: 40
Tro og love-erklæring: Ja
Indeholder besvarelsen fortroligt materiale: Nej
Må besvarelsen gøres til genstand for udlån: Ja
Må besvarelsen bruges til undervisning: Ja

UNIVERSITY OF COPENHAGEN
FACULTY OF SOCIAL SCIENCES

Master's Thesis

Algorithmic Pricing Collusion using Reinforcement Learning

Author: Julius Løve Fischer (nbm205)

Advisor: Anders Munk-Nielsen

Keystrokes: Approximately 96.000

Submission date: 31-05-2023

Algorithmic Pricing Collusion using Reinforcement Learning[*]

Julius Løve Fischer

Department of Economics

University of Copenhagen

Abstract

Firms increasingly use Artificial Intelligence (AI) algorithms to price their products. This has raised concerns that AI algorithms may learn to collude on prices above the competitive level. By means of computer simulations, I compare the collusive capabilities of using multi-agent algorithms as opposed to single-agent algorithms. I show that when firms employ multi-agent algorithms to set prices in a sequential duopoly model, they learn to collude on supracompetitive prices close to the joint-profit maximizing level more frequently than single-agent algorithms. By performing various robustness checks, I demonstrate that the results are robust and can be amplified even further.

[*]The associated Python code is available at GitHub.

Contents

1 Introduction

Artificial Intelligence (AI) is in rapid development. With its ability to simulate human intelligence, AI holds unprecedented potential to revolutionize many areas of society, for better or worse. In relation to economics, empirical evidence suggests that firms are increasingly using AI algorithms to price their products across various industries[1]. In light of these findings, concerns that AI algorithms may engage in collusive pricing behaviour above the competitive level have been raised by economists and the European Commission among others[2]. Effectively, AI algorithms may form price cartels without any explicit form of communication. From a welfare perspective, such an outcome is undesirable, and it would therefore warrant legislation for AI algorithms used to price products. This could either be through regulation, or through refinement of current antitrust laws.

The most common way to implement AI algorithms is by means of reinforcement learning. This is the approach taken in most of the literature on algorithmic pricing collusion, and it is also the approach taken in this thesis. In reinforcement learning, algorithms learn to select perceived best actions through trial and error. In the context of algorithmic pricing collusion, firms employ algorithms that compete against each other in a market where they repeatedly explore and select actions in the form of prices. As the algorithms gather more experience over time by selecting different prices, they learn which prices are perceived as being optimal. The literature on algorithmic pricing collusion has to my knowledge only considered Q-learning, which is a simple reinforcement learning algorithm originally designed for single-agent settings, however Q-learning can also be applied to multi-agent settings. Works that have studied the emergence of algorithmic pricing collusion using Q-learning include Waltman and Kaymak (2008), Calvano et al. (2020), Asker et al. (2021), and Klein (2021). These works have used computer simulations to demonstrate that Q-

[1]See e.g. Chen et al. (2016) and Assad et al. (2020).

[2]See e.g. Calvano et al. (2020), European Commission (2020), and Danish Competition and Consumer Authority (2021).

learning can learn to collude on prices above the competitive level in economic models. Waltman and Kaymak (2008), Calvano et al. (2020), and Asker et al. (2021) focus on economic models with a simultaneous-move framework, in which the algorithms set prices simultaneously. On the contrary, Klein (2021) focuses on an economic model with a sequential-move framework, in which the algorithms take turns setting prices. Klein (2021) motivates this framework by noting that it is unlikely that competing algorithms in reality update their prices at exactly the same time. In this thesis, I will focus on the economic model considered by Klein (2021).

My contribution to the literature is to investigate the emergence of algorithmic pricing collusion using algorithms that are inherently designed for multi-agent settings. To this end, I will use algorithms within the multi-agent reinforcement learning (MARL) literature from the field of computer science. The main motivation for considering multi-agent algorithms is due to the findings in Klein (2021) that, while Q-learning can learn to collude, it often does not learn to collude on prices that are joint-profit maximizing. Furthermore, when the complexity of the economic model is increased by increasing the set of prices that the algorithms can choose from, Q-learning colludes on fixed prices far less frequently. Instead, Q-learning tends to converge to deterministic Edgeworth price cycles. As a next step, I find it natural to investigate whether the same implications hold when algorithms that are inherently designed for multi-agent settings are being employed by firms.

My main finding is that when multi-agent algorithms are applied to the economic model in Klein (2021), the convergence to deterministic Edgeworth price cycles becomes the exception rather than the rule. Instead, the multi-agent algorithms tend to collude on fixed prices that are close to the joint-profit maximizing level. Further, the same result holds when the complexity of the model is increased. These findings enhance the concerns that AI algorithms used to price products may learn to collude in practice.

The rest of the thesis is organized as follows. Section 2 introduces the theory of reinforcement learning. Here, the basic concepts of reinforcement learning are first introduced informally, and subsequently formalized mathematically. In section 3, I present the considered economic model and relate it to the concepts of reinforcement learning. In section 4, I describe the algorithms that will be put to use in the economic model. In section 5, I first describe the setup and performance metrics used in the simulations. This is followed by a presentation and analysis of the results of the simulations. In section 6, I perform various robustness checks, and in section 7 I provide a discussion of the results and the considered MARL algorithms. Lastly, section 8 sums up with some concluding remarks.

2 Reinforcement Learning Theory

In this section, I will explain the theory behind reinforcement learning from a computer science perspective and try to relate it to economics. First, I will present the underlying idea of reinforcement learning and the components that describe a reinforcement learning system. Next, the concept of a Markov decision process is introduced which formally defines the dynamics of a reinforcement learning system in the single-agent case. Lastly, the concept of a Markov decision process is extended to the multi-agent case which is called a Markov game. The concept of a Markov game is important, because the economic model to be presented in the next section fits within this framework. The exposition of this section is partly based on Sutton and Barto (2018) and partly based on Zhang et al. (2021) and Buşoniu et al. (2010).

2.1 The Underlying Idea of Reinforcement Learning

In traditional machine learning problems, learning is assumed to be in the form of supervised learning. In supervised learning, the learner has access to a collected labeled data set[3]. Each collected data point consists of an input (possibly a vector of

[3]In machine learning, the learner refers to the algorithm that is being used to learn from the data.

inputs) and a corresponding output (label). In econometrics, the input corresponds to the independent variables (regressors), and the output corresponds to the dependent variable (regressand). It is called a *labeled* data set because the output (label) in supervised machine learning is known, unlike in unsupervised machine learning, where the collected data set is *unlabeled* as the output (label) is unknown. The goal in supervised machine learning is concerned with generalization. Specifically, the collected input-output pairs are used to train a machine learning algorithm with the intent to predict labels given new unseen input points. If the algorithm is capable of providing good predictions on new unseen data points, the algorithm is said to generalize well. The way in which data is collected in the supervised learning framework is similar to regression analysis within econometrics, however economists are usually interested in causal inference rather than generalization.

In reinforcement learning, the learner does not have access to a collected data set. Instead, the learner (agent) learns by experience as it repeatedly interacts with an environment. This is called the agent-environment interaction. Further, the goal in reinforcement learning is inherently different than in supervised learning. Rather than being concerned with generalization, the objective in reinforcement learning is to maximize some notion of reward. The fundamental learning principle that underlies most reinforcement learning algorithms is trial and error. This way of learning naturally resembles how human beings learn. Consider, for instance, a child learning to ride a bicycle. At first, an inexperienced child may try different actions resulting in the child falling off the bicycle. However, by means of trial and error, the child eventually gathers enough experience to learn which actions to take in order to not fall off the bicycle. In the terminology of reinforcement learning, we could model the above example as an agent-environment reinforcement learning system containing the following components: states of the environment, available actions of the agent, a reward function, and a transition probability function determining the dynamics of the system, that is, how states transition from one state to the next state. The state of the environment contains the relevant information that the agent bases its actions on. In the bicycle example the state could be a vector containing the current speed of the bicycle, the current position of the handlebar, and so on. The agent observes the current state from the environment and then chooses an action, e.g. to use the breaks or change the position of the handlebar. As a consequence of the current state and action, the environment will determine a new state according to the transition probability function which would be characterized by relevant laws of physics. Moreover, as the new state is determined, the environment simultaneously outputs a reward. If the agent has not fallen of the bicycle in the new state it receives a positive reward. Conversely, if the agent has fallen of the bicycle it receives a negative reward. The goal of the agent is to maximize the sum of received rewards by choosing appropriate actions. Thus, the received reward acts as a signal to the agent so that the agent can learn to steer its decision making towards optimal actions.

The bicycle example illustrates that the idea of reinforcement learning can resemble how human beings learn. It might therefore seem natural to extend the idea of reinforcement learning to economic models in which agents take the form of consumers or firms. In this case, the notion of reward that agents wish to maximize could be utility or profits.

2.2 Markov Decision Processes

The ideas described in the previous section can for a single-agent model mathematically be formalized as a Markov decision process (MDP). A MDP is a discrete-time stochastic control process that builds on the concept of a Markov chain.

Definition 1 *A Markov chain (process) is defined as a tuple, $MC = (\mathcal{S}, p)$, where $\mathcal{S}$ denotes the state space and $p : \mathcal{S} \times \mathcal{S} \rightarrow [0, 1]$ denotes the transition probability* *function adhering to the Markov property.*[4]

In Markov chains (processes) with discrete state spaces p is a transition matrix, and in Markov chains (processes) with continuous state spaces p is a kernel[5]. The function $p(s'|s)$ is the probability (density if a kernel) of the next state, s', conditional on the current state, s. A Markov chain (process) with (possibly stochastic) rewards associated to each state is called a Markov reward process (MRP).

Definition 2 *A Markov reward process is defined as a tuple, MRP* $= (\mathcal{S}, p, r, \gamma)$*, where* $\mathcal{S}$ *and* p *are defined as in definition 1,* $r : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ *denotes the reward function adhering to the Markov property, and* γ *is a discount factor.*

In a MRP, the reward function, $r(s, s')$, outputs the reward obtained in the next state, r', as a function of the current state, s, and next state, s'. Finally, a MDP is an extension of a discrete-time Markov reward process with actions. The implication of this setup is that an agent can now take actions to influence the trajectory of the Markov decision process. The precise formulation of a MDP depends on the setting. Here, we will consider the classical infinite-horizon discounted MDP setting. As most reinforcement learning problems are formulated in this setting, this is not always stated explicitly when formulating the MDP. In a similar vein, I will use MDP to refer to what is formally an infinite-horizon discounted MDP.

Definition 3 *An infinite-horizon discounted MDP is defined as a tuple,* *MDP* $= (\mathcal{S}, \mathcal{A}, p, r, \gamma)$*, where* $\mathcal{S}$ *denotes the state space of the environment,* $\mathcal{A}$ *denotes the action space of the agent,* $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ *denotes the transition probability function that determines the dynamics of the environment,* $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ *denotes the reward function, and* γ *denotes the discount factor*[6]*. Both* p *and* r *adhere to the Markov property.*

[4]When time is discrete it is typically called a Markov chain, and when time is continuous it is typically called a Markov process. A Markov chain can therefore be viewed as a special case of a Markov process.

[5]For a definition of a kernel in the context of Markov chains (processes), I refer to definition 6.2 in Casella and Robert (2004).

[6]In general, the action space can be a function of the state, $\mathcal{A}(s)$, however we will only focus on settings in which the set of possible actions is the same for all states.

The state space and action space in a MDP can be finite, countably infinite or continuous. For our purposes we will consider finite state and action spaces. Further, we can distinguish between a known MDP and an unknown MDP. A MDP is said to be known if the agent has complete information about the behaviour of $p(s'|s,a)$ and $r(s,a,s')$ for each state-action-next-state triple. If this does not hold the MDP is said to be unknown.

The dynamics in a MDP are as follows. At each time step, t, $s_t \in \mathcal{S}$ describes the state in which the environment is currently in. Based on s_t, the agent selects an action, $a_t \in \mathcal{A}$. As a consequence, the environment outputs a reward, $r_{t+1} \in \mathbb{R}$, according to the reward function, and a new state, $s_{t+1} \in \mathcal{S}$, according to the transition probability function. In the next time step, the agent once again interacts with the environment through the choice of a new action, a_{t+1}, based on the new state, s_{t+1}. Thereby, the agent repeatedly receives new rewards and new states by the environment according to $p(s'|s,a)$ and $r(s,a,s')$. The agent-environment interaction is visualized in figure 1 taken from Sutton and Barto (2018).

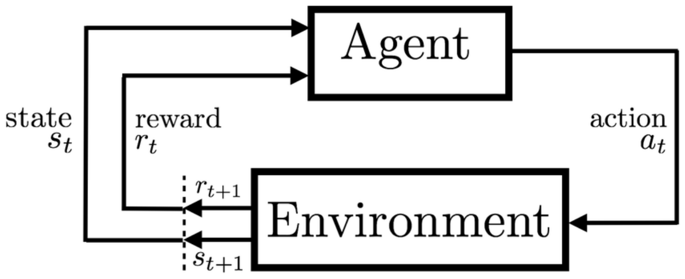


Figure 1: The agent-environment interaction in a MDP. Source: Sutton and Barto (2018).

In the context of an infinite-horizon discounted MDP the goal of the agent is to maximize the expected cumulative future discounted rewards given by:[7]:

$$E\left[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k}, s_{t+k+1})\right] \tag{2.1}$$

[7]In equation (2.1), we use the notation $r(s_{t+k}, a_{t+k}, r_{t+k+1})$ to explicitly show the dependence on a_{t+k}. Henceforth, we will use r_{t+k+1} in place of $r(s_{t+k}, a_{t+k}, r_{t+k+1})$ for brevity.

To this end, the agent must choose a policy function, π, that at each time step determines the action of an agent given the current state. We therefore introduce the notion of the value function of a state s under a policy π, $V_\pi : \mathcal{S} \to \mathbb{R}$, defined as:

$$V_\pi(s) = \mathrm{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right] = \mathrm{E}_\pi \left[G_t | s_t = s \right] \tag{2.2}$$

In equation (2.2), $E_\pi[\cdot]$ denotes the expected value given that the agent follows the policy π. Thus, equation (2.2) is the expected cumulative future discounted rewards when starting in state s and following the policy π. The randomness in equation (2.2) is induced by p, r (if rewards are stochastic), and by π itself (if π is stochastic). For our purposes, we will consider deterministic policies, $\pi^D : \mathcal{S} \to \mathcal{A}$, and stochastic policies, $\pi^S : \mathcal{S} \to \Delta(\mathcal{A})$, where $\Delta(\mathcal{A})$ denotes the set of probability distributions over $\mathcal{A}$, and $\pi^D \subset \pi^S$. Intuitively, the agent wants to find a policy that maximizes equation (2.2). Formally, the agent therefore wishes to solve the following optimization problem[8]:

$$V^*(s) = \sup_{\pi \in \Pi} V_\pi(s), \quad \forall s \in \mathcal{S} \tag{2.3}$$

Where $V^* : \mathcal{S} \to \mathbb{R}$ is called the optimal value function. If there exists a policy π^* such that $V_{\pi^*}(s) = V^*(s)$ for all $s \in \mathcal{S}$, then π^* is called an optimal policy. The problem thus amounts to finding an optimal policy, and this is at the heart of reinforcement learning.

An alternative way to find an optimal policy that maximizes equation (2.2) that will become useful later is to instead consider a state-action value function called the Q-function, $Q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$Q_\pi(s,a) = \mathrm{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right] = \mathrm{E}_\pi \left[G_t | s_t = s, a_t = a \right] \tag{2.4}$$

[8]For discrete state and action spaces, the supremum can be interchanged with the max operator.

Equation (2.4) is the expected cumulative future discounted rewards when starting in state s taking action a and thereafter following the policy π. The optimal Q-function, $Q^* : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, is similarly defined as:

$$Q^*(s,a) = \sup_{\pi \in \Pi} Q_\pi(s,a), \quad \forall s \in \mathcal{S}, a \in \mathcal{A} \tag{2.5}$$

$Q^*(s,a)$ is related to $V^*(s)$ through the expression:

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s,a) \tag{2.6}$$

And hence the optimal policy in s can be obtained as:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s,a) \tag{2.7}$$

2.3 Markov Games

The generalization of a MDP to the multi-agent case is called a Markov game, sometimes also known as a stochastic game. A Markov game can similarly be viewed as an extension of repeated games with states.

Definition 4 *A Markov game is defined as a tuple, $MG = (\mathcal{S}, \mathcal{A}_1, ..., \mathcal{A}_n, p, r_1, ..., r_n, \gamma)$, where n denotes the number of agents, $\mathcal{S}$ denotes the state space observed by all agents, $\mathcal{A}_i$ denotes the action space of agent i, and γ denotes the discount factor. Define the joint action space as $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$, then the transition probability function is given as $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ and the reward function for agent i is given as $r_i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$.*

Thus, agents now jointly interact with the environment simultaneously at each time step, and their joint actions determine the dynamics of the system as illustrated in figure 2 taken from Liu and Jia (2021). As with MDPs, we can distinguish between a known Markov game and an unknown Markov game depending on whether the agent has full knowledge of the transition probability function and the reward function[9]. Markov games are typically divided into three settings depending on the cooperative/competitive nature of the environment. These settings are referred to as fully cooperative, fully competitive, and mixed settings. In a fully cooperative setting, the Markov game is modeled as a team Markov game such that $r_i = ... = r_n = r$. In a fully competitive setting, the Markov game is typically modeled as a zero-sum Markov game such that $\sum_{i \in \mathcal{N}} r_i = 0$. Lastly, the mixed setting imposes no restrictions on the rewards, and it is characterized as being neither fully cooperative nor fully competitive. Each agent acts in its own self-interest, and its reward function may or may not conflict with other agents' reward functions.

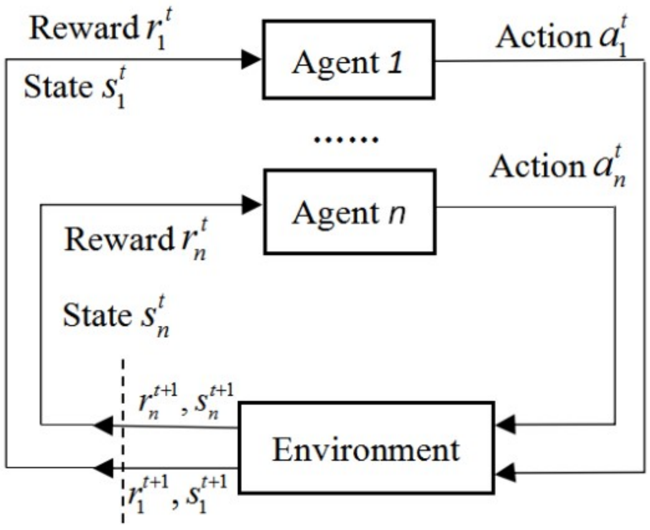


Figure 2: The agent-environment interaction in a Markov game. Source: Liu and Jia (2021).

Similar to the MDP setting, we can define notions of value in Markov games. Letting $\boldsymbol{\pi} \in \boldsymbol{\Pi}$ denote the joint policy, agent i's value function in a Markov game is now given as:

$$V_{\boldsymbol{\pi}}^i(s) = \mathrm{E}_{\boldsymbol{\pi}}\left[\sum_{k=0}^{\infty} \gamma^k r_{i,t+k+1} | s_t = s\right] = \mathrm{E}_{\boldsymbol{\pi}}\left[G_{i,t} | s_t = s\right] \tag{2.8}$$

[9]Generally, the transition probability function and reward function are unknown to the agent in Markov games, as they are functions of other agents' actions, and hence their policies. Thus, full information about other agents' (time-varying) policies are required.

Equation (2.8) is the expected cumulative future discounted rewards when starting in state s and following the joint policy $\boldsymbol{\pi}$. Similarly, letting $\mathbf{a} \in \mathcal{A}$ denote the joint action, the Q-function is now defined as:

$$Q^i_{\boldsymbol{\pi}}(s, \mathbf{a}) = \mathrm{E}_{\boldsymbol{\pi}}\left[\sum_{k=0}^{\infty} \gamma^k r_{i,t+k+1} | s_t = s, \mathbf{a}_t = \mathbf{a}\right] = \mathrm{E}_{\boldsymbol{\pi}}\left[G_{i,t} | s_t = s, \mathbf{a}_t = \mathbf{a}\right] \tag{2.9}$$

Equation (2.9) is the expected cumulative future discounted rewards when starting in state s taking the joint action $\mathbf{a}$ and thereafter following the joint policy $\boldsymbol{\pi}$. In the context of Markov games, Nash equilibria are typically introduced as a solution concept since we are dealing with multiple policies.

Definition 5 *A Nash equilibrium of the Markov game is defined as a joint policy,* $\boldsymbol{\pi} = (\pi^*_1, ..., \pi^*_n)$*, such that for any* $s \in \mathcal{S}$ *and* $i \in \mathcal{N}$[10]*:*

$$V^i_{\pi^*_i, \pi^*_{-i}}(s) \geq V^i_{\pi_i, \pi^*_{-i}}(s), \quad \forall \pi_i \in \Pi_i \tag{2.10}$$

In definition 5, the notation π^*_{-i} refers to the joint policy used in the Nash equilibrium of all agents excluding agent i's policy. In general, there might be multiple Nash equilibria, only some of which are optimal. By optimal, we mean that a Nash equilibrium provides the highest sum of expected discounted rewards out of all possible Nash equilibria.

Ideally, agent i wishes to maximize equation (2.8) or equation (2.9) with respect to its own policy taking into account other agents' (optimal) policies, but this is challenging, as it requires that agent i knows the other agents' policies which may change over time. If the other agents' policies are stationary, the Markov game essentially reduces to a stationary MDP in which agent i solves the problem in equation (2.3). However, as agents often learn simultaneously, this is rarely the case. When agents learn simultaneously, each agent is faced with a moving-target

[10]In definition 5, we could similarly have defined the Nash equilibrium concept in terms of Q-functions as in e.g. Hu and Wellman (2003).

problem: the best policy changes as other agents' policies change (Busoniu et al. (2008)). Algorithms within the MARL literature that try to deal with this issue can be classified into two fields: a joint action learner approach and an independent learner approach. The joint action learner approach tries to estimate (or assumes) other agents' policies in order to learn Q-functions with optimal joint policies, but methods within this field are often restrictive in terms of the assumptions they impose on the agents. For instance, joint action learner algorithms often require homogeneity of the agents' learning algorithms or experience sharing between agents. These assumptions are likely not in accordance with how firms use reinforcement learning to price their products in the real world, and arguably do not fit well within the algorithmic pricing collusion literature. Further, learning in the joint action space has severe scalability issues as the dimensionality of the Q-function in the joint action space increases exponentially with the number of agents. Joint action learners typically maintain estimated Q-functions of each agent which accelerates the scalability issue. For these reasons, we focus on independent learners. Independent learners do not maintain Q-functions in the joint action space, but instead maintain single Q-functions in the agents' own action space as if it was in the realm of MDPs. Thus, the agent is not faced with the challenge of estimating other agents' policies and dealing with the scalability issue. This approach, however, comes at the cost of the environment appearing non-stationary from the perspective of the agent. In particular, the transition probability function and reward function in a Markov game are not only functions of of agent i's current action, but also functions of other agents' actions, and hence functions of other agents' policies. Thus, if the agent treats the Markov game as a MDP, the transition probability function and reward function defining the MDP are non-stationary, as the functions change over time. This happens because in a MDP, the transition probability function and reward function do not explicitly account for the impact of other agents' time-varying policies which implies that $p(s'|s, a_i, \pi_1, ..., \pi_n) \neq p(s'|s, a_i, \pi'_1, ..., \pi'_n)$ and

$r(s, a_i, \pi_1, ..., \pi_n, s') \neq r(s, a_i, \pi'_1, ..., \pi'_n, s')$ when any $\pi_i \neq \pi'_i$. The challenges of non-stationary environments will turn out to be important when we discuss theoretical convergence guarantees for Q-learning later.

2.3.1 Sequential-Move Markov Games

Although Markov games are simultaneous-move games in which agents choose actions simultaneously at each time step, they also cover settings in which agents take actions sequentially. To be precise, a sequential-move Markov game can be viewed as special case of (simultaneous-move) Markov games where at each state, only one of the agents is "active", and the other agents' actions are kept fixed such that they have no influence on the reward function and transition probability function Xie et al. (2020)[11]. In this view, we define a sequential-move Markov game as follows[12].

Definition 6 *A sequential-move Markov game is defined as a tuple,*
SM-MG $= (\mathcal{S}_1, ..., \mathcal{S}_n, \mathcal{A}_1, ..., \mathcal{A}_n, p, r_1, ..., r_n, \gamma)$*, where* n *denotes the number of agents,* $\mathcal{S}_i$ *denotes the state space of agent* i*, and* $\mathcal{A}_i$ *denotes the action space of agent* i*. Define the joint state space as* $\mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$ *and define the joint action space as* $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$*, then the transition probability function is given as* $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ *and the reward function for agent* i *is given as*
$r_i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$*. Importantly, when the current state is* s_i*, then for each* $s_i \in \mathcal{S}_i$*,* p *and* $r_1, ..., r_n$ *are independent of* $\mathcal{A}_{-i}$*.*

To account for differences in the perceived state, the joint space in definition 6 is now partitioned, $\mathcal{S} = \mathcal{S}_1 \cup, ..., \cup \mathcal{S}_n$, as opposed to the Markov game in definition 4. For instance, if agent 1 moves first and agent 2 moves second, then if the current state is $s_1 \in \mathcal{S}_1$, agent 1 selects an action, $a_1 \in \mathcal{A}_1$, and the state transitions to a new state, $s'_2 \in \mathcal{S}_2$, independently of $a_{-1} \in \mathcal{A}_{-1}$.

[11]Depending on the precise scenario, one can alternatively interpret the non-active player as being able to take an action, rather than being fixed, but still unable to affect the reward function and transition probability function.

[12]Sequential-move Markov games are typically called turn-based Markov games within the MARL literature. I use the term sequential-move to keep it consistent with Klein (2021).

3 Economic Model

In this section, I present the considered economic model and explain how it relates to Markov games. The economic model is a sequential-move model by Maskin and Tirole (1988) in which firms take turns choosing prices. This model was studied by Klein (2021) using Q-learning. In this thesis, the model will be analyzed in the setting of duopolies.

3.1 Sequential Pricing Duopoly Model

In the economic model (environment), competition between two firms (agents) takes place in discrete time with an infinite time horizon. Firm 1 only adjusts its price (action), $a_{1,t} = p_{1,t} \in \mathcal{P}$, at odd-numbered time steps, and firm 2 only adjusts its price (action), $a_{2,t} = p_{2,t} \in \mathcal{P}$ at even-numbered time steps. Prices are discrete and evenly spaced between 0 and 1 according to $\mathcal{P} = \{0, \frac{1}{k}, \frac{2}{k}, ..., 1\}$. Demand for firm i is defined as:

$$D_i(p_{j,t}, p_{i,t}) = \begin{cases} 1 - p_{i,t} & \text{if} \quad p_{i,t} < p_{j,t} \\ \frac{1}{2}(1 - p_{i,t}) & \text{if} \quad p_{i,t} = p_{j,t} \\ 0 & \text{if} \quad p_{i,t} > p_{j,t} \end{cases} \tag{3.1}$$

If we assume no marginal or fixed costs, the profit for firm i is given as[13]:

$$\omega_i(p_{j,t}, p_{i,t}) = p_{i,t} D_i(p_{j,t}, p_{i,t}) \tag{3.2}$$

The monopoly or joint-profit maximizing collusive price is $p^C = 0.5$ giving a per-firm profit of $\omega_i = 0.125$. Future profits are discounted by a discount factor, $\gamma \in [0, 1)$, and the objective for firm i is to maximize the expected cumulative stream of future

[13] Profit is usually denoted as π. In order to avoid confusion between policies and profits I use ω to denote profit.

discounted profits given by:

$$\mathrm{E}\left[\sum_{k=0}^{\infty}\gamma^k\omega_i(p_{j,t+k},p_{i,t+k})\right] \tag{3.3}$$

Thus, the nature of the problem lends itself to sequential-move Markov games in mixed settings. In particular, the action space for firm i, $\mathcal{A}_i = \mathcal{P}$, is simply the prices firm i can choose between, $a_{i,t} = p_{i,t} \in \mathcal{P}$. With regards to the state space, there are several ways to model this. Recall, that the state should contain relevant information to the agent about which action to take. Since profits for firm i are determined by the prices of both firms, one reasonable approach is to simply model the state of firm i as the selected action by firm j in the same period, $s_{i,t} = p_{j,t} \in \mathcal{S}_i$ for $i \neq j$[14]. This is the approach taken in Klein (2021), and we will discuss this matter later. To account for the sequential-move setting, it is reasonable to define rewards for firm i to be firm i's current period profit plus firm i's next period profit discounted as in Klein (2021), where we now use the terminology of states and actions:

$$r_i(s_{i,t},a_{i,t},s_{i,t+1}) = \omega_i(s_{i,t},a_{i,t}) + \gamma\omega_i(s_{i,t+1},a_{i,t}) \tag{3.4}$$

Lastly, note that if independent learners are used to solve the sequential-move Markov game, the issue of non-stationarity arises due to the fact that the transition probability function from the perspective of firm i directly depends on firm j's policy. This is because the new state received by firm i is directly determined by firm j's policy (which may change over time).

4 Reinforcement Learning Algorithms

In this section, I first briefly describe how reinforcement learning algorithms can be classified across different dimensions depending on the solution technique of the

[14]This is feasible due to the sequential nature of the problem. Given that firm i selects an action at time t, the action $p_{j,t}$ is known at time t, since it is fixed from the previous period.

reinforcement learning algorithm. Next, I introduce and motivate the choice of algorithms that will be used for solving the considered economic model. Specifically, I will focus on independent learner algorithms due to the aforementioned restrictive assumptions of most joint learner algorithms that arguably do not fit well within the algorithmic pricing collusion literature.

4.1 Classification of Reinforcement Learning Algorithms

Reinforcement learning algorithms that tries to find an optimal (joint) policy in a MDP or Markov game can be divided into being either model-based or model-free. An algorithm is said to be model-based if it uses the transition probability function and reward function directly to estimate $V^*(s)$ or $Q^*(s, a)$. Thus, if the MDP or Markov game is known, one can apply model-based solution methods from the field of dynamic programming such as value iteration Bellman (1957) and policy iteration Howard (1960) to find an optimal policy based on the value function. These methods are, however, not feasible in our setting as p and r are unknown to the agent, since they are functions of other agents' policies. To find an optimal policy in unknown MDPs or unknown Markov games, one can still use model-based approaches, but in this scenario a model is instead built that, based on experience, tries to estimate p and r, since the agent cannot access the model directly (where the model here refers to p and r)[15]. On the other hand, an algorithm is said to be model-free if it does not require a model of p and r to find an optimal policy. Further, one can distinguish between value-based and policy-based algorithms. An algorithm is said to be value-based if an optimal policy is derived by explicitly estimating the optimal value function or optimal Q-function. On the contrary, a policy-based algorithm searches for an optimal policy in the space of policies directly. Policy-based methods works by learning a parameterized policy, in which the gradients are updated towards directions that will increase the expected sum of future discounted rewards Sutton

[15]These are known as planning methods within the reinforcement learning literature.

and Barto (2018). In this thesis, we will consider model-free value-based algorithms.

4.2 Q-Learning

The first algorithm that will be used for solving the economic model is Q-learning invented by Watkins (1989). Q-learning is an independent learner algorithm that is widely used within the algorithmic pricing collusion literature[16]. It is a model-free value-based algorithm that tries to find an optimal policy by estimating $Q^*(s, a)$. This is accomplished by the means of temporal difference learning. Temporal difference learning is a combination of incremental Monte Carlo methods and bootstrapping methods. Here, I give a brief description of the underlying ideas[17]. For a detailed exposition of temporal difference learning I refer to Sutton and Barto (2018).

Monte Carlo (MC) methods in the context of reinforcement learning are designed for policy evaluation, that is, to estimate the expected cumulative future discounted rewards when starting in state s following a given policy π, $V_\pi(s)$, where π may not necessarily be an optimal policy. As $V_\pi(s)$ involves an expectation the idea is simply to estimate $V_\pi(s)$ by replacing the expectation with an empirical average of the received accumulated rewards, based on samples of trajectories starting in state s that are generated according to the policy π[18]. Notice, however, that the trajectory may not start in state s. One way to deal with this issue is called first-visit MC. In first-visit MC, the first time state s is visited along the trajectory, from this point in time, t, and onwards, we essentially have a "new" trajectory starting in state s as desired. For each trajectory, h, starting in s and generated according to π, the received accumulated rewards along the trajectory, $G_{h,t} = r_{h,t+1} + \gamma r_{h,t+2} + ... + \gamma^{T_h - t - 1} r_{h,T_h}$, serves as *one* Monte Carlo sample, where T_h denotes the final time step of the h'th

[16]See e.g. Calvano et al. (2020), Klein (2021), and Asker et al. (2021).

[17]While Q-learning is a relatively simple algorithm to implement, it is often presented as a black box tool for reinforcement learning. My hopes is that I can provide some intuition for the working mechanisms behind the scenes.

[18]When the expectation is replaced by the empirical average, this is sometimes referred to as a crude Monte Carlo estimator. There exists many other methods within in the field of Monte Carlo methods for estimating expectations.

trajectory. Thus, if we average over these Monte Carlo samples generated by all trajectories starting in s, we have an estimate of $V_\pi(s)$. The first-visit MC is data inefficient, in the sense that we could generate multiple trajectories starting in s every single time state s is visited in one full trajectory. This is the idea behind every-visit MC. Every-visit MC can be defined by the following estimator, $\hat{V}_\pi(s)$[19]:

$$V_\pi(s) = \mathrm{E}_\pi\left[G_t | s_t = s\right] \approx \frac{1}{N(s)} \sum_{h=1}^{H} \sum_{t=0}^{T_h - 1} \mathbb{1}[s_{h,t} = s] G_{h,t} = \hat{V}_\pi(s) \tag{4.1}$$

Where $N(s)$ is a counter for the number of visits to state s. The above estimator can be rewritten using the general idea that a mean can be computed incrementally:

$$\mu_h = \frac{1}{H} \sum_{h=1}^{H} x_h = \mu_{H-1} + \frac{1}{H}(x_H - \mu_{H-1}) \tag{4.2}$$

Using equation (4.2) to rewrite equation (4.1), we arrive at the incremental Monte Carlo algorithm for policy evaluation in algorithm 1. The pseudocode for algorithm 1 is inspired by Brunskill (2019).

Algorithm 1: Incremental Monte Carlo for Policy Evaluation

```
  Input: policy $\pi$ to be evaluated, $\alpha_t$
  Initialize: V(s)=0 $\forall s \in \mathcal{S}$
  Loop forever (for each episode)
1     Generate an episode (trajectory) following $\pi : s_{h,0}, a_{h,0}, r_{h,1}, ..., s_{h,T_h-1}, a_{h,T_h-1}, r_{h,T_h}$
      Loop for each step of episode (trajectory), $t = 0, 1, ..., T_h - 1$
2         $G_{h,t} \leftarrow \sum_{j=t+1}^{T_h} \gamma^{j-t-1} r_{h,j}$
3         $V(s_{h,t}) \leftarrow V(s_{h,t}) + \alpha_t(G_{h,t} - V(s_{h,t}))$
      End
  End
```

In algorithm 1, α_t is a (potentially time-varying) learning rate to account for that fact in some cases it may be sensible to attach a greater weight to more recent updates[20]. If $\alpha_t = \frac{1}{N(s)}$, then algorithm 1 is equivalent to every-visit MC. It is shown in Singh et al. (1996) that $V(s)$ in algorithm 1 is a biased but consistent

[19]In the reinforcement learning literature there is no tradition of using "hats" to explicitly indicate that the quantity is an estimator. Henceforth, we will follow this tradition and omit subscript π to indicate that $V(s)$ is an estimator of $V_\pi(s)$.

[20]Note that the update step in step 3 in algorithm 1 is now an exponential average.

estimator of $V_\pi(s)$. Algorithm 1 brings us to another important point: Monte Carlo methods in reinforcement learning are only suitable for episodic MDPs, since we can only compute $G_{h,t}$ once we have reached the terminal state of a trajectory[21]. In infinite-horizon MDPs there is no terminal state, and hence we cannot compute $G_{h,t}$. This is where the idea of bootstrapping comes in. Bootstrapping in the context of reinforcement learning refers to the idea that an estimate is updated on the basis of another estimate. This is the exact same idea used in dynamic programming when estimates of $V_\pi(s)$ (or $V^*(s)$) are updated. In particular, we can rewrite $V_\pi(s)$ as:

$$V_\pi(s) = E_\pi\left[G_t | s_t = s\right] = E_\pi\left[r_{t+1} + \gamma V_\pi(s_{t+1}) | s_t = s\right] \tag{4.3}$$

Thus, we can replace $G_{h,t}$ with $r_{h,t+1} + \gamma V(s_{h,t+1})$ in algorithm 1. Another important aspect of using bootstrapping is that we are no longer required to wait until the end of an episode to compute the *target*, that is to compute $r_{h,t+1} + \gamma V(s_{h,t+1})$. Instead, the target can be computed at every time step. This brings us to the temporal difference learning update TD(0), that Q-learning is based upon:

$$\text{TD}(0): \quad V(s_t) \leftarrow V(s_t) + \alpha_t(\underbrace{r_{t+1} + \gamma V(s_{t+1})}_{target} - V(s_t)) \tag{4.4}$$

In equation (4.4) we suppress notation of the h'th episode, as the TD(0) update does not require the MDP to be episodic in nature (although it could be). Intuitively, $r_{t+1} + \gamma V(s_{t+1})$ provides a better estimate of $V_\pi(s)$ as opposed to $V(s_t)$, since the former progressively loads the information contained in the received reward, r_{t+1}, as time passes. TD(0) is similarly designed for policy evaluation and cannot directly be used for what is called control problems, that is, to find an optimal policy, π^*. In control problems, the idea is to combine policy evaluation with policy improvements in order to find an optimal policy, but this can not be done model-free when working

[21]An episodic MDP is a MDP that runs in episodes with terminal states which are repeated indefinitely.

with value functions, since it would require us to know the transition probability function and reward function in order to perform the one-step look ahead policy improvement, in the same manner that value iteration does policy improvements Silver (2015). To be precise, a greedy policy improvement with value functions would require:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} \sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma V_\pi(s') \right] \tag{4.5}$$

In order to make TD(0) model-free for control problems, we have to make the value of actions explicit by considering Q-functions. In this case, a greedy policy improvement follows directly from the structure of the Q-function, without having to know p and r:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} Q_\pi(s,a) \tag{4.6}$$

The greedy policy in equation (4.6) results in a policy improvement since it satisfies the conditions (4.7) of the policy improvement theorem stated in theorem 1 from Sutton and Barto (2018).

Theorem 1 *Let π and π' be any pair of deterministic policies such that $\forall s \in \mathcal{S}$,*

$$Q_\pi(s, \pi'(s)) \geq V_\pi(s). \tag{4.7}$$

Then the policy π' must be as good as, or better than, π. That is, it must obtain greater or equal expected return from all states $s \in \mathcal{S}$:

$$V_{\pi'}(s) \geq V_\pi(s). \tag{4.8}$$

This is where Q-learning comes into the picture. Q-learning is essentially TD(0) learning with Q-functions and greedy policy improvements designed for control problems. The pseudocode for Q-learning in algorithm 2 is inspired by Sutton and Barto (2018).

Algorithm 2: Q-learning

```
Input: $\pi_b$ (behaviour policy), $\alpha_t$
Initialize: $s_0$, $Q(s,a)$ arbitrarily $\forall s \in \mathcal{S}$, $\forall a \in \mathcal{A}$
for $t = 0, ..., T-1$ do
1       Choose $a_t$ from $s_t$ using $\pi_b$ derived from $Q$ (e.g. $\varepsilon$-greedy)
2       Take action $a_t$, observe $r_{t+1}$, $s_{t+1}$
3       $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \left[ r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$
end
```

Instead of inputting a fixed policy to be evaluated in algorithm 2, we instead input a behaviour policy, π_b (we will soon discuss the role of this behaviour policy). Further, the target now uses $\max_{a'} Q(s_{t+1}, a')$ which implies a greedy policy improvement according to equation (4.6). In Q-learning, the greedy policy used in the target is referred to as the target policy to distinguish it from the behaviour policy. The idea of using a greedy target policy follows from the Bellman optimality equation which is a fundamental property of the optimal Q-function:

$$Q^*(s,a) = E\left[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') | s_t = s, a_t = a \right] \tag{4.9}$$

Thus, the Q-function is updated greedily towards the optimal Q-function independently of the action chosen by the followed behaviour policy. For this reason, Q-learning is called an off-policy algorithm, because the update is performed as if the agent followed the greedy policy. If the action was chosen according to the behaviour policy, the update in step 3 in algorithm 2 would be:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{4.10}$$

Where a_{t+1} is sampled according to the behaviour policy, $a_{t+1} \sim \pi_b(s_{t+1})$. When the update is according to equation (4.10), then algorithm 2 (with slight modifications) is called SARSA (State-Action-Reward-State-Action), which is an on-policy algorithm as it uses the action chosen by π_b to update the Q-function. This is analogous to the update in TD(0) where instead of updating the Q-function towards the optimal

Q-function directly, it instead updates the Q-function towards the currently followed policy, π_b, which itself (if chosen wisely) constantly improves towards an optimal policy. Thus, the target in SARSA is based on the following expression which resembles equation (4.3):

$$Q_{\pi_b}(s,a) = E_{\pi_b}\left[r_{t+1} + \gamma Q_{\pi_b}(s_{t+1}, a_{t+1}) | s_t = s, a_t = a\right] \tag{4.11}$$

The question that naturally arises is how we can be sure that algorithm 2 finds an optimal policy. To answer this, we have to understand the concept of generalized policy iteration (GPI) which Q-learning (and SARSA) is based upon. GPI is the general idea of letting policy improvement and policy evaluation processes interact Sutton and Barto (2018). In the policy improvement step in Q-learning, the target policy is greedily improved with respect to the current estimate of the Q-function according to equation (4.6). In the policy evaluation step in Q-learning, the Q-function is then updated to more closely approximate the Q-function for the current target policy rather than some fixed policy. Note that both of these processes are captured simultaneously by the update in step 3, where the process of policy improvement in Q-learning is implicit in the sense that the we do not perform an actual policy improvement step. Instead, the target policy is improved implicitly via the term $\max_{a'} Q(s_{t+1}, a')$. The intuition behind GPI is fairly straightforward. Suppose that the current estimate of the optimal Q-function is a poor estimate such that in some state, s, the greedy target policy currently chooses a suboptimal action, a. Eventually, as the estimate of the optimal Q-function improves according to the current target policy, the agent learns that the action, a, is suboptimal in state s. As a result, the target policy will adjust its greedy action to a different (likely improved) action according to the new improved estimate. Now, the estimate of the optimal Q-function according to the new (improved) target policy is evaluated. Eventually, as the new estimate of the optimal Q-function is improved, the agent learns whether or not the new action is optimal in state s. If the agent learns that the new action is still suboptimal in state s, the target policy is once again improved, and these two interacting processes continue until they stabilize, that is, when no more policy improvements can be made. At this point, one may wonder why we need a behaviour policy, π_b, in Q-learning. To illustrate the necessity of the behaviour policy, consider the following example where instead of sampling actions from π_b in step 1 in algorithm 2, we instead sample actions according to the greedy target policy. Suppose that in some state, s, an action, a, gives a reward of -1 with probability $\frac{3}{4}$ and a reward of +100 with probability $\frac{1}{4}$. If action a is in fact the optimal action in state s, and the agent happened to receive a reward of -1 the first time action a is chosen in state s, then greedy action selection according to a poor estimate of $Q(s,a)$ implies that the agent will never learn the optimal action in state s, provided that some other action in state s happened to give a reward higher than -1 on its first visit. The reason is that greedy action selection under these circumstances will never select the action a in state s again, and hence the poor estimate of $Q(s,a)$ is never updated. This gives rise to what is called the exploration-exploitation trade-off in model-free reinforcement learning, and the behaviour policy is designed to deal with this exact issue. Intuitively, an agent should explore the environment sufficiently in order to discover optimal behaviour, but exploration is costly as the agent may receive low rewards in the process of exploration, which in turn could be avoided by exploiting its current best options. A popular strategy to balance exploration and exploitation is to choose π_b to be an ε-greedy policy defined as:

$$\pi_{\varepsilon-greedy}(s) = \begin{cases} \arg\max_a Q(s,a) & \text{w.p.} \quad 1-\varepsilon_t \\ \text{sample uniformly at random from } \mathcal{A} & \text{w.p.} \quad \varepsilon_t \end{cases} \tag{4.12}$$

The ε-greedy policy selects the greedy action w.p. $1-\varepsilon_t$, and it selects an action uniformly at random w.p. ε_t. It is important to notice that when the behaviour policy in Q-learning is ε-greedy, the policy evaluation step in Q-learning also results in a policy improvement with regards to the the behaviour policy due to the greedy action selection w.p. $1 - \varepsilon_t$[22]. Typically, the value of ε_t is chosen to be high at early time stages, where the agent has little information about the dynamics of the environment, to induce a lot of exploration. As the agent eventually learns the dynamics of the environment through exploration, ε_t is gradually decreased over time to progressively exploit its best actions according to its understanding of the environment. The ε-greedy strategy is widely used within the algorithmic pricing literature[23].

One potential drawback of the ε-greedy strategy is that it chooses actions completely at random when it is in the exploration phase. This might be induce inefficient and costly exploration if actions that empirically appear to be very suboptimal keeps getting explored. Instead, it might be reasonable to explore actions that empirically have shown to be promising. A popular exploration strategy that captures this idea is to use a Boltzmann policy Matignon et al. (2012):

$$\pi_{\text{Boltzmann}}(s, a) = \frac{e^{Q(s,a)/\tau_t}}{\sum\limits_{a' \in \mathcal{A}} e^{Q(s,a')/\tau_t}} \tag{4.13}$$

Where τ_t is called the temperature controlling the randomness of the action selection[24]. When $\tau_t \to 0$, the Boltzmann policy is equivalent to a greedy policy, whereas when $\tau_t \to \infty$, the Boltzmann policy is equivalent to a pure random policy, in which actions are chosen uniformly at random. If $0 < \tau_t < \infty$, the Boltzmann policy will select higher-valued actions with higher probabilities according to the current estimate of the optimal Q-function. Similar to the ε-greedy policy, when the behaviour policy is a Boltzmann policy, the policy evaluation step in Q-learning also implies a

[22]It is desirable to have the behaviour policy improve over time by selecting the greedy action w.p. $1 - \varepsilon_t$ rather than e.g. a pure random policy, as the latter can be very costly.

[23]See e.g. Calvano et al. (2020), Klein (2021), Asker et al. (2021).

[24]Note that the Boltzmann policy is simply a softmax function extended with a temperature parameter.

policy improvement with regards to the behaviour policy. Further, the value of τ_t is typically chosen to be high at early time stages to induce exploration, but gradually decreasing over time to induce exploitation.

We are now ready to present convergence guarantees for Q-learning in the single-agent case.

Theorem 2 *For a finite-state and finite-action infinite-horizon stationary MDP, if the Robbins-Monro conditions,*

$$0 < \alpha_t < 1, \quad \sum_{t=1}^{\infty} \alpha_t = \infty, \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty,$$

are satisfied, and all state-action pairs are visited infinitely often, then the Q-function in Q-learning converges to the optimal Q-function almost surely:

$$P(\forall s \in \mathcal{S}, a \in \mathcal{A} : \lim_{t \longrightarrow \infty} Q(s,a) = Q^*(s,a)) = 1$$

Thus, as long as the learning rate satisfies the Robbins-Monro conditions, both the ε-greedy policy and Boltzmann policy with $\tau > 0$ ensure that all state-action pairs are visited infinitely often such that the Q-function in Q-learning converges to the optimal Q-function almost surely. Thereby, we can compute the optimal policy, π^*, according to equation (2.7)[25]. For practical purposes, the Q-function often converges to a sufficient approximation of the true optimal Q-function within a finite amount of steps in the sense that the implied policy of the estimated Q-function is optimal, i.e., it is the same policy implied by the true optimal Q-function. The proof of Theorem 2 is given in Watkins and Dayan (1992). Importantly, the proof of the theorem relies on the stationarity assumption of the MDP. As we have argued, the stationarity assumption is not satisfied when Q-learning is used as an independent learner to solve a Markov game as if it was a MDP, since we are dealing with a moving-

[25]Note that if ε_t (or τ_t) in π_b decays to zero in the limit, then π_b converges to π^*. This is known as GLIE (Greedy in the Limit with Infinite Exploration).

target problem. Because the agent's learning process in multi-agent systems depends on the behaviour policy used by other agents, different exploration policies may induce different behaviour for independent learner algorithms, and hence different convergence properties. This issue has been raised several times within the MARL literature (see e.g. Buşoniu et al. (2010), Nowé et al. (2012), and Matignon et al. (2012)). Later, we investigate the importance of the choice of exploration strategy for the emergence of collusive equilibria in the sequential duopoly model.

Lastly, we present a modified version of Q-learning in algorithm 3 adapted to the sequential duopoly model called sequential Q-learning due to Klein (2021).

Algorithm 3: Sequential Q-learning

```
  Input: $\pi_{b_i}$ (behaviour policy) for $i = \{1, 2\}$, $\alpha_t$
  Initialize: $p_{1,1}$ and $p_{2,2}$ randomly, $Q^i(s_i, a_i)$ arbitrarily $\forall s_i \in \mathcal{S}_i$, $\forall a_i \in \mathcal{A}_i$ for $i = \{1, 2\}$. Set $i = 1$, $j = 2$
  for $t = 3, ..., T - 1$ do
1     Set $s_{i,t} = p_{j,t-1}$ and choose $a_{i,t} = p_{i,t}$ from $s_{i,t}$ using $\pi_{b_i}$ derived from $Q^i$ (e.g. $\varepsilon$-greedy)
2     Take action $a_{i,t}$ and observe $r_{i,t+1}$ and $s_{i,t+1}$, where:
          $r_{i,t+1} = \omega_i(s_{i,t}, a_{i,t}) + \gamma\omega_i(s_{i,t+1}, a_{i,t})$
3     $Q^i(s_{i,t}, a_{i,t}) \leftarrow Q^i(s_{i,t}, a_{i,t}) + \alpha_t \left[ r_{i,t+1} + \gamma^2 \max_{a'_i} Q^i(s_{i,t+1}, a'_i) - Q^i(s_{i,t}, a_{i,t}) \right]$
4     Update $\{i \leftarrow j, j \leftarrow i\}$
  end
```

Due to the sequential nature, the Q-function is now only updated every second period in algorithm 3. Further, the discount factor is now squared as the reward function represents the sum of two-period rewards[26].

4.3 WoLF-PHC

In this section, I introduce the WoLF-PHC (Win or Learn Fast Policy Hill Climbing) algorithm proposed and discussed in Bowling and Veloso (2001) and Bowling and Veloso (2002). The WoLF-PHC algorithm is one of the most used algorithms within

[26]We have defined the reward function as the joint reward received in two consecutive time steps due to the sequential nature. We could alternatively have defined the reward function to be the immediate reward received in each time step. In this case, the squared discount factor becomes evident from recursive substitution of the target in step 3 accounting for the fact that the action is fixed for two time steps: $Q^*(s, a) = E\left[r_{t+1} + r_{t+2} + \gamma^2 \max_{a'} Q^*(s_{t+1}, a') | s_t = s, a_t = a, a_{t+1} = a\right]$.

the MARL literature that historically has been considered state of the art Stimpson and Goodrich (2003). The algorithm is designed for mixed settings, but have been shown to perform well in fully competitive settings (see Bowling and Veloso (2002) and fully cooperative settings (see Matignon et al. (2012)). For these reasons, I argue that WoLF-PHC is a natural next step to investigate whether the emergence of algorithmic pricing collusion in the sequential duopoly model is robust to other choices of algorithms. WoLF-PHC combines policy hill climbing with the WoLF (Win or Learn Fast) principle. Policy hill climbing is an extension of Q-learning with hill climbing in order to enable the algorithm to learn stochastic policies[27]. This is particularly important in Markov games with only mixed equilibria. In relation to the sequential duopoly model, using sequential Q-learning, Klein (2021) finds a pattern of Edgeworth price cycles along the equilibrium path as described theoretically in Maskin and Tirole (1988). Klein (2021) finds that these Edgeworth price cycles are deterministic in the sense that it is always the same firm that undertakes the costly action of "resetting" the price by jumping up in price rather than undercutting, with the other firm being able to free-ride on this. This behaviour occurs due to Q-learning only being able to learn deterministic policies. In the original paper on the sequential duopoly model by Maskin and Tirole (1988), these Edgeworth price cycles are identified as stochastic Edgeworth price cycles which require stochastic policies. We will therefore further use WoLF-PHC to investigate whether being able to learn stochastic policies can give rise to the emergence of stochastic Edgeworth price cycles. The PHC algorithm is presented in algorithm 4 adopted to the sequential duopoly model.

In algorithm 4, $\pi_{i,t}(s_{i,t}, a_i)$ is a probability distribution at time t for agent i over all actions in state $s_{i,t}$. This policy takes the place of the behaviour policy in Q-learning. In algorithm 4, the hill climbing step in step 4 adds more probability towards the action that is believed to be the best action according to the current

[27]Note that Q-learning can only learn deterministic policies, since the final policy is derived as the greedy policy of the estimated optimal Q-function.

Algorithm 4: Sequential Policy Hill Climbing

```
Input: $\pi_{b_i}$ (behaviour policy) for $i = \{1, 2\}$, $\alpha_t$
Initialize: $p_{1,1}$ and $p_{2,2}$ randomly. For $i = \{1, 2\}$ and $\forall s_i \in \mathcal{S}_i$, $\forall a_i \in \mathcal{A}_i$: $\pi_i(s_i, a_i) = \frac{1}{|\mathcal{A}_i|}$, $Q^i(s_i, a_i)$
arbitrarily. Set $i = 1$, $j = 2$
for $t = 3, ..., T - 1$ do
1    Set $s_{i,t} = p_{j,t-1}$ and choose $a_{i,t} = p_{i,t}$ from $s_{i,t}$ using $\pi_{b_i}$ derived from $Q^i$ (e.g. $\varepsilon$-greedy)
2    Take action $a_{i,t}$ and observe $r_{i,t+1}$ and $s_{i,t+1}$, where:
         $$r_{i,t+1} = \omega_i(s_{i,t}, a_{i,t}) + \gamma\omega_i(s_{i,t+1}, a_{i,t})$$
3    $Q^i(s_{i,t}, a_{i,t}) \leftarrow Q^i(s_{i,t}, a_{i,t}) + \alpha_t \left[ r_{i,t+1} + \gamma^2 \max_{a'_i} Q^i(s_{i,t+1}, a'_i) - Q^i(s_{i,t}, a_{i,t}) \right]$
4    Update $\pi_i(s_{i,t}, a_i)$ wrt. to the Q-function and constrain it to a legal probability distribution,
         $$\pi_i(s_{i,t}, a_i) = \pi_i(s_{i,t}, a_i) + \begin{cases} \delta & \text{if} \quad a_i = \arg\max_{a'} Q^i(s_{i,t}, a') \\ \frac{-\delta}{|\mathcal{A}_i|-1} & \text{otherwise} \end{cases}$$
5    Update $\{i \leftarrow j, j \leftarrow i\}$
end
```

estimate of the optimal Q-function. In particular, it subtracts an equal amount of probability mass, $\frac{-\delta}{|\mathcal{A}_i|-1}$, from currently perceived suboptimal actions, and adds the sum of this probability mass, δ, to the currently perceived optimal action. If $\delta = 1$ and an ε-greedy strategy is used, the algorithm is equivalent to Q-learning with an ε-greedy strategy, since in this case, the policy moves to the greedy policy[28]. WoLF-PHC extends algorithm 4 with the WoLF principle. The WoLF principle is based on the idea that an agent should learn quickly when losing, and slowly when winning. This is achieved by letting δ become a variable learning rate such that $\delta = \delta_l$ when the agent is losing, and $\delta = \delta_w$ when the agent is winning, where $\delta_w < \delta_l$. The intuition behind the WoLF principle is that an agent should adapt quickly when it is doing more poorly than expected. In contrast, when it is doing better than expected, it should be cautious since the other agents are likely to change their policies Bowling and Veloso (2001). In this way, the WoLF principle aids in convergence towards stationary policies by letting winning agents make the environment seem more stationary to agents that are losing, so that they can adapt

[28]An ε-greedy strategy with PHC amounts to selecting an action uniformly at random with probability ε_t, and selecting an action according to the policy, $\pi_{i,t}(s_{i,t}, a)$, with probability $1 - \varepsilon_t$.

more quickly to winning agents' policies Bowling and Veloso (2001). The notion of winning in WoLF-PHC is determined by keeping track of an average policy over time. If the estimated expected value of the current policy is greater than the estimated expected value of the current average policy, then the agent is winning, and otherwise it is losing. The WoLF-PHC algorithm adopted to the sequential duopoly model is presented in algorithm 5.

Algorithm 5: Sequential WoLF-PHC

```
Input: $\alpha_t$, $\delta_l > \delta_w \in (0,1]$
Initialize: $p_{1,1}$ and $p_{2,2}$ randomly. For $i = \{1,2\}$ and $\forall s_i \in \mathcal{S}_i$, $\forall a_i \in \mathcal{A}_i$: $\pi_i(s_i,a_i) = \frac{1}{|\mathcal{A}_i|}$,
$\bar{\pi}_i(s_i,a_i) = 0$, $N_i(s_i) = 0$, $Q^i(s_i,a_i)$ arbitrarily. Set $i = 1$, $j = 2$
for $t = 3, ..., T-1$ do
1   Set $s_{i,t} = p_{j,t-1}$ and choose $a_{i,t} = p_{i,t}$ from $s_{i,t}$ using $\pi_i(s_{i,t},a_i)$ with suitable exploration (e.g. $\varepsilon$-greedy)
2   Take action $a_{i,t}$ and observe $r_{i,t+1}$ and $s_{i,t+1}$, where:
        $$r_{i,t+1} = \omega_i(s_{i,t},a_{i,t}) + \gamma\omega_i(s_{i,t+1},a_{i,t})$$
3   $$Q^i(s_{i,t},a_{i,t}) \leftarrow Q^i(s_{i,t},a_{i,t}) + \alpha_t\left[r_{i,t+1} + \gamma^2 \max_{a_i'} Q^i(s_{i,t+1},a_i') - Q^i(s_{i,t},a_{i,t})\right]$$
4   Update estimate of average policy, $\bar{\pi}_i(s_{i,t},a_i)$,
        $$N_i(s_{i,t}) \leftarrow N_i(s_{i,t}) + 1$$
        $$\forall a' \in \mathcal{A}_i : \bar{\pi}_i(s_{i,t},a') \leftarrow \bar{\pi}_i(s_{i,t},a') + \frac{1}{N_i(s_{i,t})}\left(\pi_i(s_{i,t},a') - \bar{\pi}_i(s_{i,t},a')\right)$$
5   Update $\pi_i(s_{i,t},a_i)$ wrt. to the Q-function and constrain it to a legal probability distribution,
        $$\pi_i(s_{i,t},a_i) = \pi_i(s_{i,t},a_i) + \begin{cases} \delta & \text{if} \quad a_i = \arg\max_{a'} Q^i(s_{i,t},a') \\ \frac{-\delta}{|\mathcal{A}_i|-1} & \text{otherwise} \end{cases}$$
    Where:
        $$\delta = \begin{cases} \delta_w & \text{if } \sum_{a'} \pi_i(s_{i,t},a')Q^i(s_{i,t},a') > \sum_{a'} \bar{\pi}_i(s_{i,t},a')Q^i(s_{i,t},a') \\ \delta_l & \text{otherwise} \end{cases}$$
6   Update $\{i \leftarrow j, j \leftarrow i\}$
end
```

4.4 CoLF & CK

In this section, I introduce the CoLF (Change or Learn Fast) and CK (Change & Keep) principles proposed by de Cote et al. (2006). These principles are intended to foster cooperation between self-interested agents in multi-agent systems with social dilemmas when no explicit communication is possible. Algorithms using these principles may therefore increase the share of joint-profit maximizing equilibria in the sequential duopoly model.

The CoLF principle is inspired by the works of Bowling and Veloso (2002), but with the following modification: if the reward received by the agent is unexpectedly changing, then learn slowly, otherwise learn quickly de Cote et al. (2006). The idea is that an agent should not make sudden changes to its policy due to "unexpected" rewards that arise from non-stationary causes such as other agents' exploration activities or normal learning dynamics de Cote et al. (2006). In essence, these non-stationary causes may induce poor rewards for the agent, which in turn might discourage cooperation. On the other hand, when the environment appears stationary, the agent should learn fast. In algorithm 6, I present the sequential Q-learning algorithm modified with the CoLF principle. In de Cote et al. (2006), they use the idea of modelling the state at time t, as the joint action observed at $t-1$, $s_t = \mathbf{a}_{t-1}$, as learning agents may benefit from this Sandholm and Crites (1996). I use the same idea in algorithm 6, where the joint action for firm i consists of its own fixed price/action in the previous period and firm j's adjusted price/action in the previous period. Intuitively, the joint action may provide better information for the firm about which action to take as opposed to considering only the single action of the opposing firm, since the firm in this case accounts for the impact that its own price adjustment had on the opposing firm's adjusted price.

In algorithm 6, the P-values are exponential averages of the collected rewards with weight factor λ for each state-action pair, whereas the W-values are exponential averages of the absolute differences between the collected rewards and the respective P-values with weight factor λ for each state-action pair de Cote et al. (2006)[29]. Therefore, the if-statement in step 4 assesses whether the distance between the currently received reward and average collected reward in a particular state is larger

[29]These exponential averages are similar to the Q-learning update, which itself is an exponential average.

Algorithm 6: Sequential CoLF

```
    Input: $\pi_{b_i}$ (behaviour policy) for $i = \{1,2\}$, $\alpha_S > \alpha_{NS}$, $\lambda$
    Initialize: $p_{1,1}$ and $p_{2,2}$ randomly. For $i = \{1,2\}$ and $\forall s_i \in \mathcal{S}_i$, $\forall a_i \in \mathcal{A}_i$: $P_i(s_i, a_i) = 0$, $W_i(s_i, a_i) = 0$,
    $Q^i(s_i, a_i)$ arbitrarily. Set $i = 1$, $j = 2$
    for $t = 3, ..., T-1$ do
1       Set $s_t = \mathbf{a}_{t-1} = (p_{i,t-2}, p_{j,t-1})$ and choose $a_{i,t} = p_{i,t}$ from $s_t$ using $\pi_{b_i}$ derived from $Q^i$ (e.g.
          $\varepsilon$-greedy)
2       Take action $a_{i,t}$ and observe $r_{i,t+1}$ and $s_{t+1} = \mathbf{a}_{t+1} = (p_{i,t}, p_{j,t+1})$, where:

            $$r_{i,t+1} = \omega_i(s_{i,t}, a_{i,t}) + \gamma\omega_i(s_{i,t+1}, a_{i,t})$$

3       $\Delta r_{i,t+1} \leftarrow |r_{i,t+1} - P_i(s_{i,t}, a_{i,t})|$
4       if $\Delta r_{i,t+1} > W_i(s_{i,t}, a_{i,t})$ then
            $\alpha \leftarrow \alpha_{NS}$
        else
            $\alpha \leftarrow \alpha_S$
        end

5       $Q^i(s_t, a_{i,t}) \leftarrow Q^i(s_t, a_{i,t}) + \alpha \left[ r_{i,t+1} + \gamma^2 \max_{a'_i} Q^i(s_{t+1}, a'_i) - Q^i(s_t, a_{i,t}) \right]$
6       $W_i(s_t, a_{i,t}) \leftarrow (1-\lambda) W_i(s_t, a_{i,t}) + \lambda \cdot \Delta r_{i,t+1}$
7       $P_i(s_t, a_{i,t}) \leftarrow (1-\lambda) P_i(s_t, a_{i,t}) + \lambda \cdot r_{i,t+1}$
8       Update $\{i \leftarrow j, j \leftarrow i\}$
    end
```

than the average variability of the collected rewards in the same state. If the distance is larger than the average variability, then the environment appears to be non-stationary and a learning rate of α_{NS} is used, else the environment appears to be stationary and a learning rate of α_S is used.

Next, we introduce the CK principle. This principle is based on the following observation: when an agent chooses new actions due to either exploration or learning, other agents cannot foresee this change in behaviour de Cote et al. (2006). Therefore, whenever an agent changes its behaviour, it is prone to collect a misleading reward which may discourage cooperation. The idea behind the CK principle is to suspend the Q-value update whenever an agent is changing its action in a particular state. The agent then repeats the same action and uses the new reward to perform the suspended update in the original state. Hereby, the agent allows more time for the other agents to appropriately adapt to its new action, which may give a more informative reward. The sequential Q-learning algorithm modified with the CK principle is given in algorithm 7.

The CoLF and CK principle can be combined to form the CK-CoLF algorithm.

Algorithm 7: Sequential CK

```
  Input: $\pi_{b_i}$ (behaviour policy) for $i = \{1, 2\}, \alpha_t$
  Initialize: $p_{1,1}$ and $p_{2,2}$ randomly. For $i = \{1, 2\}$ and $\forall s_i \in \mathcal{S}_i$, $\forall a_i \in \mathcal{A}_i$: $status_i = update$, $Q^i(s_i, a_i)$
  arbitrarily. Set $i = 1$, $j = 2$
  for $t = 3, ..., T-1$ do
1     Set $s_t = \mathbf{a}_{t-1} = (p_{i,t-2}, p_{j,t-1})$.
2     if $status_i = update$ then
          choose $a_{i,t} = p_{i,t}$ from $s_t$ using $\pi_{b_i}$ derived from $Q^i$ (e.g. $\varepsilon$-greedy)
      else
          $a_{i,t} \leftarrow a_{i,t-1}$
      end
3     Take action $a_{i,t}$ and observe $r_{i,t+1}$ and $s_{t+1} = \mathbf{a}_{t+1} = (p_{i,t}, p_{j,t+1})$, where:
          $$r_{i,t+1} = \omega_i(s_{i,t}, a_{i,t}) + \gamma\omega_i(s_{i,t+1}, a_{i,t})$$
4     if $status_i = update$ then
          if $a_{i,t} \neq a_{i,t-1}$ then
              $status_i \leftarrow keep$
              $s_{upd} \leftarrow s_t$
          else
              $status_i = update$
              $$Q^i(s_t, a_{i,t}) \leftarrow Q^i(s_t, a_{i,t}) + \alpha_t \left[ r_{i,t+1} + \gamma^2 \max_{a'_i} Q^i(s_{t+1}, a'_i) - Q^i(s_t, a_{i,t}) \right]$$
          end
      else
          $status_i = update$
          $$Q^i(s_{upd}, a_{i,t}) \leftarrow Q^i(s_{upd}, a_{i,t}) + \alpha_t \left[ r_{i,t+1} + \gamma^2 \max_{a'_i} Q^i(s_{t+1}, a'_i) - Q^i(s_{upd}, a_{i,t}) \right]$$
      end
5     Update $\{i \leftarrow j, j \leftarrow i\}$
  end
```

We will consider CoLF, CK, and CK-CoLF in our simulations.

5 Simulations & Results

In this section, I first present the setup and performance metrics for simulating and evaluating the different reinforcement learning algorithms in the sequential duopoly model. Based on these performance metrics, the results of the simulations for each algorithm are presented and analyzed.

5.1 Setup and Performance Metrics

The baseline setup we consider for the simulations are equivalent to the setup considered in Klein (2021). The reason for doing so is to make the results in this thesis comparable to the results found in Klein (2021). In particular, we look at a baseline setup with a time horizon of $T = 500.000$, $n = 1000$ runs, $k = 6$ price intervals between 0 and 1, $\alpha_t = 0.3$, and $\gamma = 0.95$[30]. All Q-values are initialized with all zeros. For WoLF-PHC, we set $\delta_l = 0.6$ and $\delta_w = 0.1$. For CoLF and CoLF-CK, we set $\alpha_S = 0.6$, $\alpha_{NS} = 0.1$, and $\lambda = 0.1$. The learning rates for WoLF-PHC are set so that they learn rather aggressively when winning, and the learning rates for CoLF and CoLF-CK are set so that they learn rather aggressively when the environment appears stationary. The algorithms will compete against each other in self-play, that is, both firms will employ the same type of algorithm. If an ε-greedy exploration strategy is used, we use $\varepsilon_t = (1-\theta)^t$ as in Klein (2021), where the decay parameter, θ, is set such that the probability of exploration gradually decreases from 100% at the beginning to 0.1% halfway through the run, reaching 0.0001% at the end. Klein (2021) does not consider a Boltzmann exploration strategy. To mimic a similar decaying exploration procedure as with the ε-greedy approach, the values for τ_t are chosen to be T evenly spaced values between 1 and -5 on a log scale with base 10. Therefore, with $T = 500.000$, then $\tau_1 = 10^1$, $\tau_{250.000} = 10^{-2}$, and $\tau_{500.000} = 10^{-5}$. Next, we consider a simulation experiment with the exact same setup as above, but now with $k = 12$ price intervals between 0 and 1 as in Klein (2021).

To evaluate the performance of the algorithms, we look at two measures: profitability and the type of equilibrium the algorithms have converged to. The profitability in equation (5.1) is defined as the average profit in the final 1.000 periods of the run at time t, where we omit using a subscript to indicate the particular run.

$$\text{Profitability: } \Omega_i = \frac{1}{1000} \sum_{t'=t-1.000}^{t} \omega_i(s_{i,t'}, a_{i,t'}) \tag{5.1}$$

To evaluate whether the algorithms converge to profitable types of equilibria, we will look at the shares of the different types of equilibria that each algorithm reaches across the 1.000 runs.

[30] Note that the learning rate is not assumed to be time-varying in Klein (2021).

5.2 Results with $k = 6$

When $k = 6$ we are considering the following set of prices:

$$P = [0, 0.167, 0.333, 0.5, 0.667, 0.833, 1]$$

In this setting, there are two types of convergence patterns that all algorithms display. The first type of convergence pattern is the convergence to asymmetric price cycles known as Edgeworth price cycles. These Edgeworth price cycles are characterized by firms gradually undercutting each other, but when prices have become too low, both firms have an incentive to raise their price and thereby resetting the price cycle, such that they once again gradually undercut each other. As shown in Klein (2021), these Edgeworth price cycles are deterministic when using Q-learning in the sense that for a specific run it is always the same firm that undertakes the costly action of resetting the price while the other firm is able to free-ride on this. For this reason, we motivated the choice of WoLF-PHC, as it is capable of learning stochastic policies. However, despite WoLF-PHC's capability of learning stochastic policies, it does not learn to break the pattern of deterministic Edgeworth price cycles in runs where they appear. Instead, they show the exact same deterministic Edgeworth price cycle pattern. The profitability curve and price selection behaviour associated with these deterministic Edgeworth price cycle runs are shown in Appendix A in figure 12c and figure 13c respectively, where in this particular run, it is firm 1 that free-rides[31]. It is important to note that there are several types of Edgeworth price cycles. For instance, sometimes the price spirals down starting from 0.833 rather than 0.667 as shown in figure 13d in Appendix A.

The other type of convergence pattern is the convergence to collusive equilibria. The outcome of a simulation run is considered to be a collusive equilibrium if firms collude on a fixed price above zero and the algorithms have adopted policies that constitute

[31]For other runs, it might as well be firm 2 that free-rides as shown in figure 12d in Appendix A.

a Nash equilibrium as defined in definition 5. As stated in Bowling and Veloso (2002), Q-learning is a *rational* algorithm, meaning that if other agents policies converge to stationary policies, then Q-learning will converge to a best-response policy to the other agents' policies, and hence constitute a Nash equilibrium[32]. The profitability curve and price selection behaviour are illustrated for two types of collusive equilibrium runs in figure 12a, figure 12b, figure 13a, and figure 13b in Appendix A. From these plots we can deduce that agents' policies indeed converge to stationary policies, and hence these types of runs must constitute a Nash equilibrium as Q-learning is rational.

The central question at hand is whether the MARL algorithms can outperform the Q-learning algorithm used in Klein (2021) in terms of profitability, and the extent to which the algorithms manage to form collusive equilibria. In figure 3, I therefore show the average profitability curve associated with each algorithm when an ε-greedy exploration strategy is used, and in table 1 I show the corresponding shares of the types of runs for each algorithm. Since firms employ the same algorithm in each simulation, the average profitability is practically the same for firm 1 and 2. Therefore, all average profitability curves to come in this section will only illustrate the average profitability for one of the two firms.

[32]It is furthermore shown in Klein (2021) by a Nash-type of test that in these types of collusive runs, the adopted policies indeed constitute a Nash equilibrium.

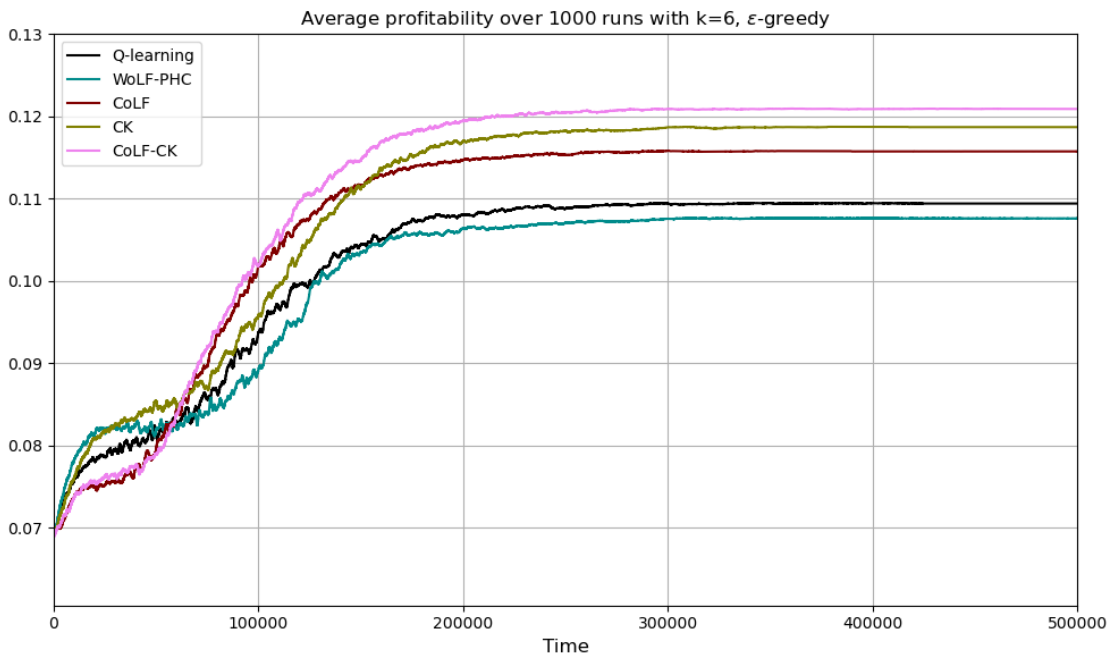


Figure 3: This figure shows the average profitability for the considered algorithms over 1000 runs with $T = 500.000$, $k = 6$, and using ε-greedy selection.

algorithm	Q-learning	WoLF-PHC	CoLF	CK	CoLF-CK
Type of run					
Edgeworth price cycle	300/1000	321/1000	60/1000	8/1000	4/1000
Collusive around 0.070	9/1000	10/1000	36/1000	9/1000	0/1000
Collusive around 0.111	461/1000	501/1000	390/1000	411/1000	293/1000
Collusive around 0.125 (optimal)	230/1000	168/1000	514/1000	572/1000	703/1000

Table 1: Shares of types of runs using ε-greedy selection. 1000 runs, 500.000 iterations, and $k = 6$. The collusive equilibria indicate which profit level the firms collude around.

Based on figure 3, we can first of all note that, as expected, the average profitability curve for Q-learning closely resembles the average profitability curve found in Klein (2021). Furthermore, it is evident from figure 3 that WoLF-PHC does not manage to increase the average profitability when compared to Q-learning. On the other hand, if we consider CoLF, CK, and CoLF-CK, they all manage to increase the average profitability significantly, with CoLF-CK providing the highest average profitability. By examining table 1, we can explain the results underlying these average profitability curves. If an algorithm in table 1 has large shares of collusive equilibria close to the joint-profit maximizing price and a low share of Edgeworth price cycles, this will increase the average profitability. This is because the profitability generated by Edgeworth price cycle runs half of the time generate high profitability, and half the time generate low profitability, but when averaged out, they do not provide higher profitability than what would occur when averaging over runs with high collusive equilibria[33]. We can see from table 1 that Q-learning and WoLF-PHC's shares of Edgeworth price cycles are large compared to CoLF, CK, and CoLF-CK. This explains why WoLF-PHC and Q-learning perform more poorly than the remaining algorithms in terms of average profitability. Further, the shares of Edgeworth price cycles for CoLF, CK, and CoLF-CK are very low, being almost close to zero for the latter two. Thus, the emergence of Edgeworth price cycles when using CoLF, CK, and CoLF-CK becomes the exception rather than the rule. Instead, CoLF, CK, and CoLF-CK increase the share collusive equilibria around the joint-profit level significantly with CoLF-CK having the largest share. This explains why CoLF-CK provides the highest average profitability curve in figure 3.

Next, we examine whether the results in figure 3 and table 1 are robust to the choice of exploration strategy. Thus, we repeat the same experiment, but now with a Boltzmann exploration strategy. I present the results in figure 4 and table 2[34].

[33]This can be deduced by figure 12c. Similar conclusions hold for other types of Edgeworth price cycle runs.

[34]WoLF-PHC is not suited for a Boltzmann exploration procedure since it samples from the PHC policy, and hence omitted.

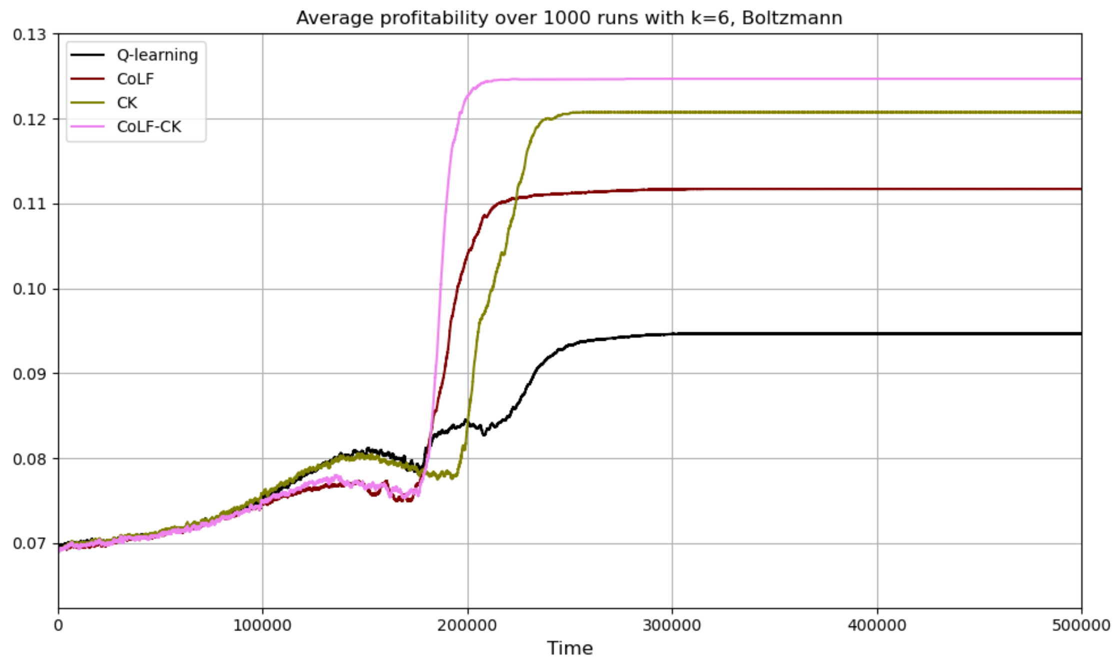


Figure 4: This figure shows the average profitability for the considered algorithms over 1000 runs with $T = 500.000$, $k = 6$, and using Boltzmann selection.

algorithm	Q-learning	WoLF-PHC	CoLF	CK	CoLF-CK
Type of run					
Edgeworth price cycle	832/1000	-	272/1000	6/1000	0/1000
Collusive around 0.070	51/1000	-	1/1000	2/1000	0/1000
Collusive around 0.111	49/1000	-	365/1000	295/1000	23/1000
Collusive around 0.125 (optimal)	68/1000	-	362/1000	697/1000	977/1000

Table 2: Shares of types of runs using Boltzmann selection. 1000 runs, 500.000 iterations, and $k = 6$. The collusive equilibria indicate which profit level the firms collude around.

Interestingly, when a Boltzmann exploration strategy is used as opposed to an ε-greedy strategy, the average profitability for Q-learning reduces significantly in figure 4. Based on table 2, this can be attributed to the fact that most runs now converge to Edgeworth price cycles rather than high collusive equilibria. In a similar vein, CoLF now performs more poorly for the same reasons, however using a Boltzmann strategy, it still outperforms Q-learning with an ε-greedy strategy in figure 3. On the other hand, CK and CoLF-CK actually performs better with a Boltzmann selection strategy, with the latter being improved significantly. For CoLF-CK, there was not even a single run of Edgeworth price cycles in all 1000 runs. Further, in almost all runs, CoLF-CK converged to the highest collusive equilibrium.

I conjecture that this result may be attributed to the CK principle working better with a Boltzmann strategy. In particular, when an agent changes its action and therefore performs its suspended update in the next period, the new reward used for the suspended update may provide better information when a Boltzmann selection strategy is used as opposed to an ε-greedy strategy. The reason is that when other agents adopt to the agent's new action using an ε-greedy strategy in the exploration phase (i.e. when ε_t is high), then the new reward is prone to be the result of a completely random action with probability ε_t. Some of these actions may not reflect what these other agents would do if they were not in the exploration phase, and hence provide a more uninformative reward. On the other hand, if the other agents adopt to the agent's new policy using a Boltzmann exploration strategy in the exploration phase, then they will select optimal actions with higher probabilities, and hence likely to provide more informative rewards.

5.3 Results with $k = 12$

When $k = 12$ we are considering the following set of prices:

$$P = [0, 0.083, 0.167, 0.25, 0.333, 0.417, 0.5, 0.583, 0.667, 0.75, 0.833, 0.917, 1]$$

In this setting, the types of convergence patterns are equivalent to when $k = 6$. The implication of increasing the number of prices is that the dimensionality of the state and action spaces increases. This affects the algorithms in two ways. First, the increased dimensionality implies that the environment becomes more complex to navigate through, and hence more difficult to learn. Secondly, the increased dimensionality implies that for a given learning horizon, each state-action pair are visited less frequently as there are now more state-action pairs to visit. In Klein (2021) it is shown that when k increases, Q-learning has increasing difficulties in learning collusive equilibria. Instead, Q-learning tends to converge to Edgeworth price cycles in most runs. As shown in section 5.2, CoLF, CK, and CoLF-CK converged to high collusive equilibria in most runs. In figure 5 and table 3 below, we therefore investigate whether these implications are robust when k is increased using ε-greedy selection.

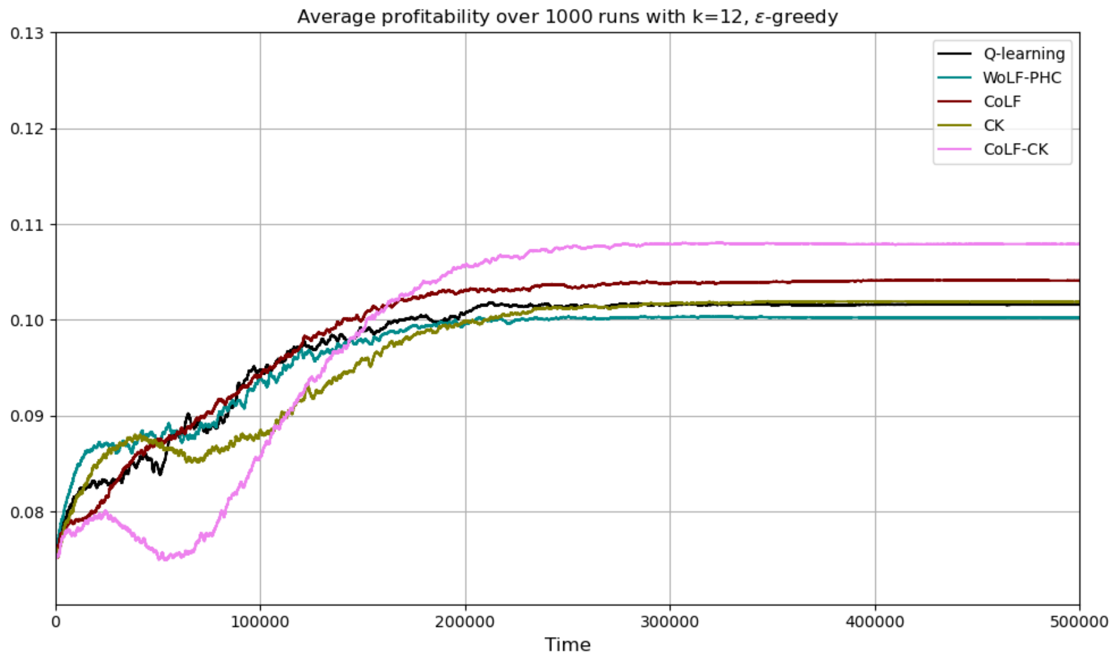


Figure 5: This figure shows the average profitability for the considered algorithms over 1000 runs with $T = 500.000$, $k = 12$, and using ε-greedy selection.

algorithm	Q-learning	WoLF-PHC	CoLF	CK	CoLF-CK
Type of run					
Edgeworth price cycle	849/1000	931/1000	257/1000	111/1000	36/1000
Collusive around 0.038	0/1000	0/1000	0/1000	0/1000	0/1000
Collusive around 0.070	0/1000	0/1000	7/1000	86/1000	19/1000
Collusive around 0.094	5/1000	2/1000	331/1000	323/1000	286/1000
Collusive around 0.111	42/1000	21/1000	245/1000	323/1000	370/1000
Collusive around 0.122	56/1000	27/1000	129/1000	129/1000	225/1000
Collusive around 0.125 (optimal)	48/1000	19/1000	31/1000	28/1000	64/1000

Table 3: Shares of types of equilibria using ε-greedy selection. 1000 runs, 500.000 iterations, and $k = 12$. The collusive equilibria indicate which profit level the firms collude around.

As shown in figure 5, CoLF, CK, and CoLF-CK once again provide the highest average profitability, with the latter being the highest. From table 3, this result can be attributed to the high shares of collusive equilibria. Now, however, CoLF, CK, and CoLF-CK have increasing difficulties in finding collusive equilibria close to the joint-profit maximizing level. Specifically, they now have relatively large shares of collusive equilibria around profits of 0.094. Notably, CK also has a relatively large share of collusive equilibria around profits of 0.070. This explains why the average profitability of CK is close to the average profitability of Q-learning. Moreover, while CoLF and CK now have a relatively large shares of Edgeworth price cycles compared to before, CoLF-CK still manages to keep the share of Edgeworth price cycles very low. This contrasts the result found for Q-learning in Klein (2021) (also illustrated in table 3) that Edgeworth price cycles are a frequent phenomenon when k increases. If we look at the average profitability for WoLF-PHC in figure 5, it again performs poorly which is not surprising given the previous results in section 5.2. Next, we examine the implications of using a Boltzmann exploration strategy when $k = 12$. The results are presented in figure 6 and table 4.

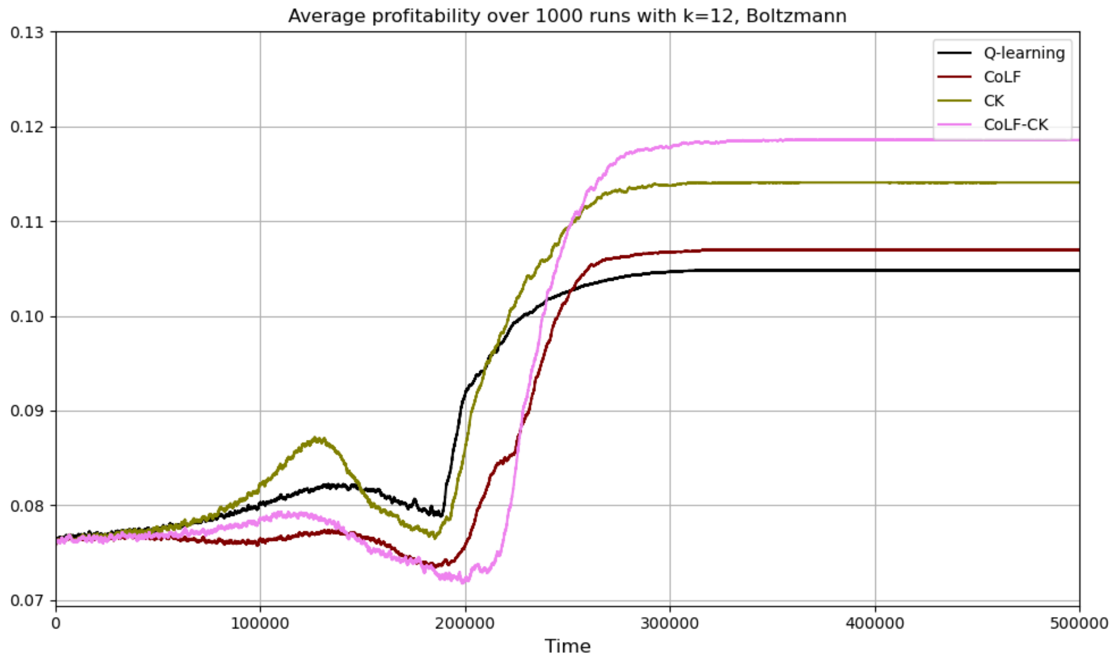


Figure 6: This figure shows the average profitability for the considered algorithms over 1000 runs with $T = 500.000$, $k = 12$, and using Boltzmann selection.

algorithm	Q-learning	WoLF-PHC	CoLF	CK	CoLF-CK
Type of run					
Edgeworth price cycle	982/1000	-	729/1000	40/1000	61/1000
Collusive around 0.038	0/1000	-	0/1000	1/1000	0/1000
Collusive around 0.070	0/1000	-	0/1000	23/1000	2/1000
Collusive around 0.094	3/1000	-	14/1000	92/1000	29/1000
Collusive around 0.111	5/1000	-	90/1000	322/1000	138/1000
Collusive around 0.122	7/1000	-	109/1000	364/1000	469/1000
Collusive around 0.125 (optimal)	3/1000	-	58/1000	158/1000	301/1000

Table 4: Shares of types of equilibria using Boltzmann selection. 1000 runs, 500.000 iterations, and $k = 12$. The collusive equilibria indicate which profit level the firms collude around.

Similar to when $k = 6$, a Boltzmann selection strategy significantly increases the amount Edgeworth price cycle runs for Q-learning and CoLF resulting in lower average profitability in figure 6. From table 4, we can see that Q-learning now exhibits Edgeworth price cycle behaviour in almost all runs, whereas CoLF now does it in 72,9% of the runs. On the other hand, CK and CoLF-CK manage to increase the share of high collusive equilibria compared to an ε-greedy strategy, while keeping the shares of Edgeworth price cycles low, resulting in higher average profitability.

6 Robustness

In this section, I first investigate the implications of increasing the learning horizon when $k = 12$. Next, I investigate the implication of initializing the Q-values in a particular way to produce what is known as a relaxation search. Lastly, I consider whether collusion can occur when agents are heterogeneous. Klein (2021) has performed robustness checks using different learning parameters and discount factors, and are therefore not considered here.

6.1 Increased Learning Horizon

In subsection 5.3, we showed that when $k = 12$, CK and CoLF-CK tended to keep the shares of Edgeworth price cycles low, however they had more difficulties finding high collusive equilibria compared to when $k = 6$. As alluded to earlier, this might occur because of the increased complexity of the environment and less frequent visits to each state-action pairs induced by the increased dimensionality. To alleviate these two issues, we investigate what happens when the time horizon is increased for $k = 12$. Hereby, we allow the algorithms more time to learn the more complex environment, while simultaneously allowing for more visits to each state-action pairs. I report the results with an increased time horizon of $T = 2.000.000$ for $n = 200$ runs in figure 7 and table 5 using a Boltzmann strategy.

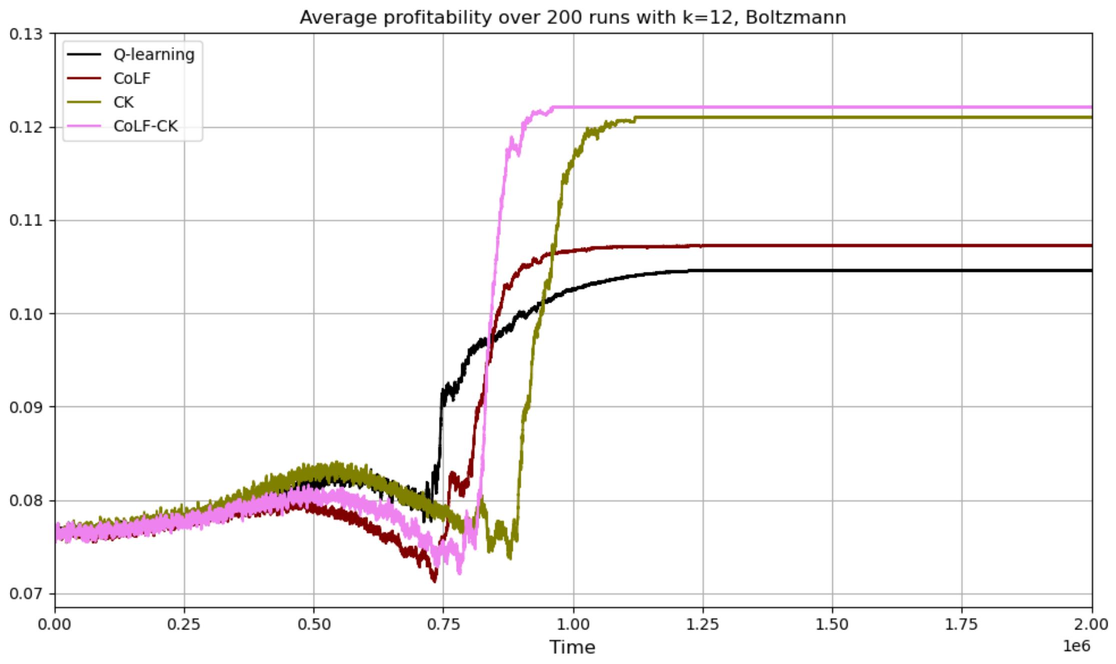


Figure 7: This figure shows the average profitability for the considered algorithms over 200 runs with $T = 2.000.000$, $k = 12$, and using Boltzmann selection.

algorithm	Q-learning	WoLF-PHC	CoLF	CK	CoLF-CK
Type of run					
Edgeworth price cycle	198/200	-	153/200	4/200	5/200
Collusive around 0.038	0/200	-	0/200	0/200	0/200
Collusive around 0.070	0/200	-	0/200	1/200	0/200
Collusive around 0.094	0/200	-	4/200	5/200	3/200
Collusive around 0.111	0/200	-	15/200	19/200	12/200
Collusive around 0.122	2/200	-	23/200	94/200	70/200
Collusive around 0.125 (optimal)	0/200	-	5/200	77/200	110/200

Table 5: Shares of types of equilibria using Boltzmann selection. 200 runs, 2.000.000 iterations, and $k = 12$. The collusive equilibria indicate which profit level the firms collude around.

As shown in table 5, while Q-learning and CoLF do not notably benefit from an increased learning horizon, the shares of high collusive equilibria for CK, and CoLF-CK increase significantly resulting in higher average profitability shown in figure 7. Similar conclusions hold for an ε-greedy strategy shown in figure 14 and table 10 in Appendix B.

6.2 Initialization of Q-values

In the baseline setup discussed in subsection 5.1, we initialized all Q-values to zero. This was done to make the results more transparent with the results presented in Klein (2021), which similarly initialize all Q-values to zero. However, in de Cote et al. (2006), they propose to initialize the Q-values as $\frac{r_{max}}{1-\gamma}$ to create a relaxation search, where r_{max} is the immediate maximum reward[35]. This strategy was originally suggested by Stimpson et al. (2001). The idea behind the strategy is to utilize that given $\gamma < 1$, the highest possible sum of discounted rewards is given by the convergence formula for a geometric series. Hereby, the Q-values are initialized at their highest possible level, and the Q-values for each state-action pair are lowered over time as a result of the Q-update. Consequently, the strategy creates an optimistic learning bias towards state-action pairs that have not been visited frequently in the past. After a sufficient number of time steps, the Q-values will eventually relax to realistic levels, hence the name relaxation search. In our setting, we let $r_{max} = 0.25$[36]. In figure 8 and table 6, I present results for $T = 500.000$, $n = 1000$ runs, and $k = 12$ using a relaxation search together with a Boltzmann strategy.

[35]A similar strategy is used in e.g. M-Qubed proposed by Crandall and Goodrich (2011), which also aims at fostering cooperation in mixed settings.

[36]Equivalently, we could let $r_{max} = 0.25 + \gamma \cdot 0.25$ to explicitly account for the sequential setting, in which case Q-values are initialized as $\frac{r_{max}}{1-\gamma^2}$.

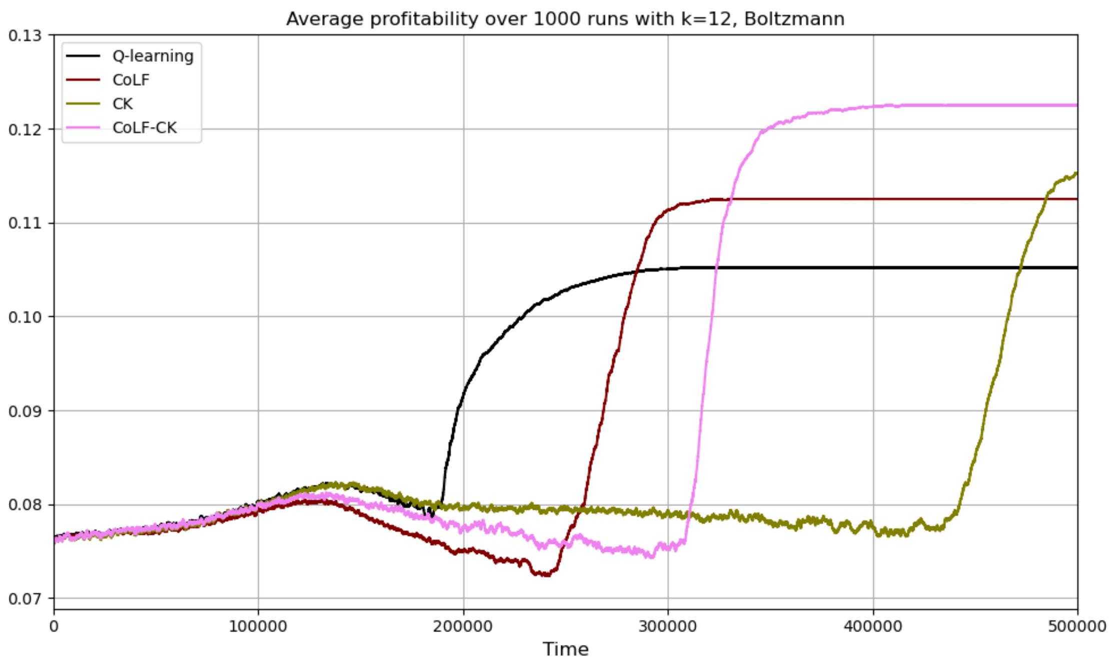


Figure 8: This figure shows the average profitability with a relaxation search for the considered algorithms over 1000 runs with $T = 500.000$, $k = 12$, and using Boltzmann selection.

algorithm	Q-learning	WoLF-PHC	CoLF	CK	CoLF-CK
Type of run					
Edgeworth price cycle	983/1000	-	362/1000	8/1000	18/1000
Collusive around 0.038	0/1000	-	0/1000	0/1000	0/1000
Collusive around 0.070	0/1000	-	1/1000	0/1000	0/1000
Collusive around 0.094	4/1000	-	29/1000	0/1000	7/1000
Collusive around 0.111	5/1000	-	156/1000	10/1000	21/1000
Collusive around 0.122	8/1000	-	297/1000	421/1000	476/1000
Collusive around 0.125 (optimal)	0/1000	-	155/1000	404/1000	478/1000

Table 6: Shares of types of equilibria using Boltzmann selection and relaxation search. 1000 runs, 500.000 iterations, and $k = 12$. The collusive equilibria indicate which profit level the firms collude around. Note that CK did not converge in 157 runs.

As is evident from table 6, the relaxation search significantly improves the share of high collusive equilibria for CoLF, CK and CoLF-CK. Meanwhile, the relaxation search does not affect the performance of Q-learning. Similar conclusions hold for a ε-strategy shown in figure 15 and table 11 in Appendix C.

Unlike before when Q-values were initialized as zeros, CK now struggles to converge. In fact, CK did not converge in 157 runs, and it is evident from figure 8 that 500.000 time periods appears to be close to the threshold where CK starts to converge. A closer inspection reveals that in runs where CK did not converge, the profitability for CK oscillates around a profitability level of 0.075-0.08 as shown in figure 16 in Appendix C. This can be explained by two factors. First, CK suspends its Q-update when a new action is chosen. This results in less frequent updates of the Q-values. Secondly, r_{max} in our model corresponds to one firm getting all of the profits. In a sequential setting, it is very unlikely that r_{max} is obtained in every period, as the opposing firm is incentivized to undercut its competitors price (or collude). This implies that the initial levels of the Q-values using a relaxation search are far above realistic levels of the true Q-values, and hence require longer time to relax to realistic levels. The fact that CoLF-CK still managed to converge in every iteration can be attributed to the higher learning rate used when the environment appears stationary. Next, we investigate the implications of using a relaxation search together with an increased learning horizon. Results are presented in figure 9 and table 7.

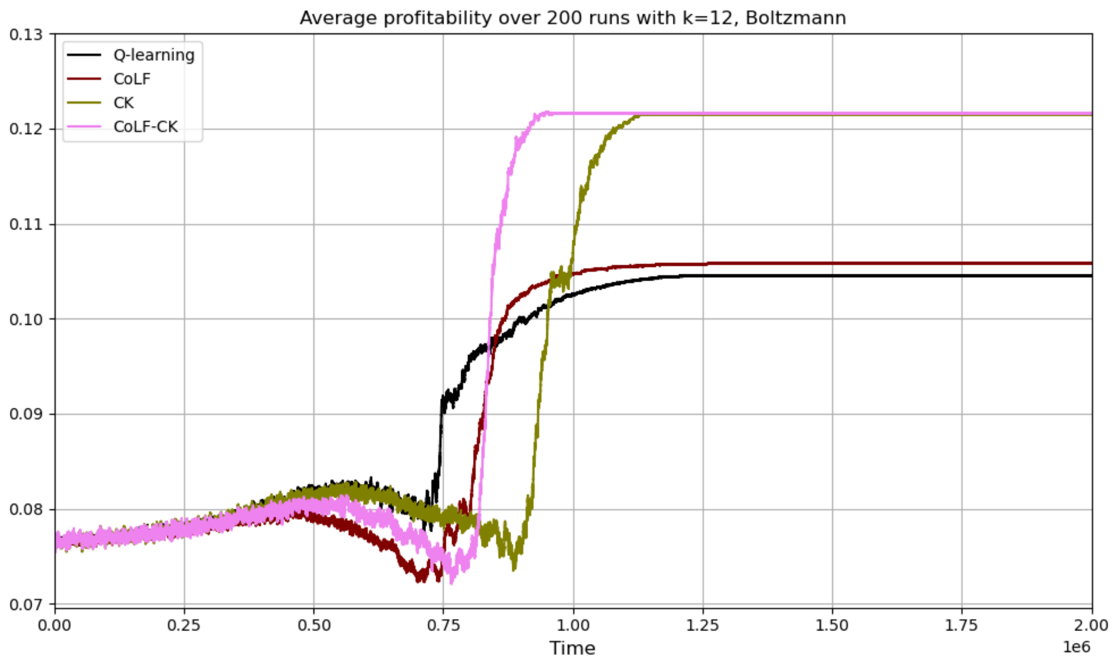


Figure 9: This figure shows the average profitability with a relaxation search for the considered algorithms over 200 runs with $T = 2.000.000$, $k = 12$, and using Boltzmann selection.

algorithm	Q-learning	WoLF-PHC	CoLF	CK	CoLF-CK
Type of run					
Edgeworth price cycle	198/200	-	152/200	4/200	6/200
Collusive around 0.038	0/200	-	0/200	0/200	0/200
Collusive around 0.070	0/200	-	0/200	0/200	0/200
Collusive around 0.094	0/200	-	2/200	5/200	3/200
Collusive around 0.111	0/200	-	20/200	18/200	9/200
Collusive around 0.122	2/200	-	16/200	79/200	84/200
Collusive around 0.125 (optimal)	0/200	-	10/200	94/200	98/200

Table 7: Shares of types of equilibria using Boltzmann selection and relaxation search. 200 runs, 2.000.000 iterations, and $k = 12$. The collusive equilibria indicate which profit level the firms collude around.

When the learning horizon is increased, CK now converges in every single run using a relaxation search. Based on the previous findings, one might expect that a relaxation search in conjunction with an increased learning horizon could improve the performance of the algorithms further. However, as shown in figure 9 and table 7, this does not seem to be the case. Similar conclusions hold for an ε-greedy strategy shown in figure 17 and table 12 in Appendix C.

6.3 Heterogeneous Agents

In this section, we investigate whether collusive equilibria can occur when firms employ different algorithms. Previously, we have shown that WoLF-PHC tended to perform poorly in self-play. However, as WoLF-PHC has been shown to perform well in competitive environments in Bowling and Veloso (2002), it might be able to outperform Q-learning in terms of average profitability in a heterogeneous setting. In figure 10 and table 8, I therefore show the average profitability when firm 1 employs WoLF-PHC and firm 2 employs Q-learning in the baseline setup with ε-greedy selection.

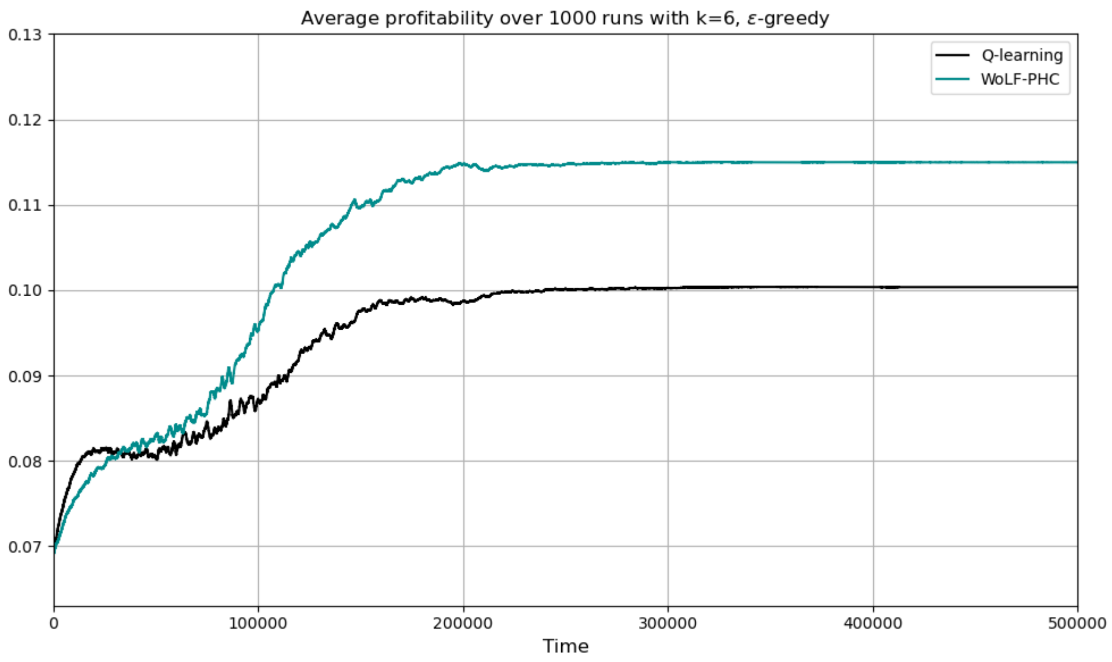


Figure 10: This figure shows the average profitability when WoLF-PHC is faced against a Q-learner. 1000 runs, $T = 500.000$, $k = 6$, and using ε-greedy selection.

algorithm	Q-learning vs WoLF-PHC
Type of run	
Edgeworth price cycle	347/1000
Collusive around 0.070	2/1000
Collusive around 0.111	473/1000
Collusive around 0.125 (optimal)	178/1000

Table 8: Shares of types of equilibria when WoLF-PHC is faced against a Q-learner. 1000 runs, 500.000 iterations, $k = 6$, and using ε-greedy selection. The collusive equilibria indicate which profit level the firms collude around.

Based on table 8, we can see that high collusive equilibria are indeed possible when agents are heterogeneous. Interestingly, figure 10 shows that WoLF-PHC converges to significantly higher average profitability compared to Q-learning, which is caused by the fact that whenever Edgeworth price cycle runs appear, it is Q-learning that tends to undertake the costly action of resetting the price cycle while WoLF-PHC learns to free-ride off this. Similar conclusions hold when $k = 12$ as shown in figure 18 and table 13 in Appendix D.

Next, I investigate what happens when firm 1 employs CoLF-CK and firm 2 employs Q-learning. Results are shown in figure 11 and table 9 in the baseline setup using ε-greedy selection.

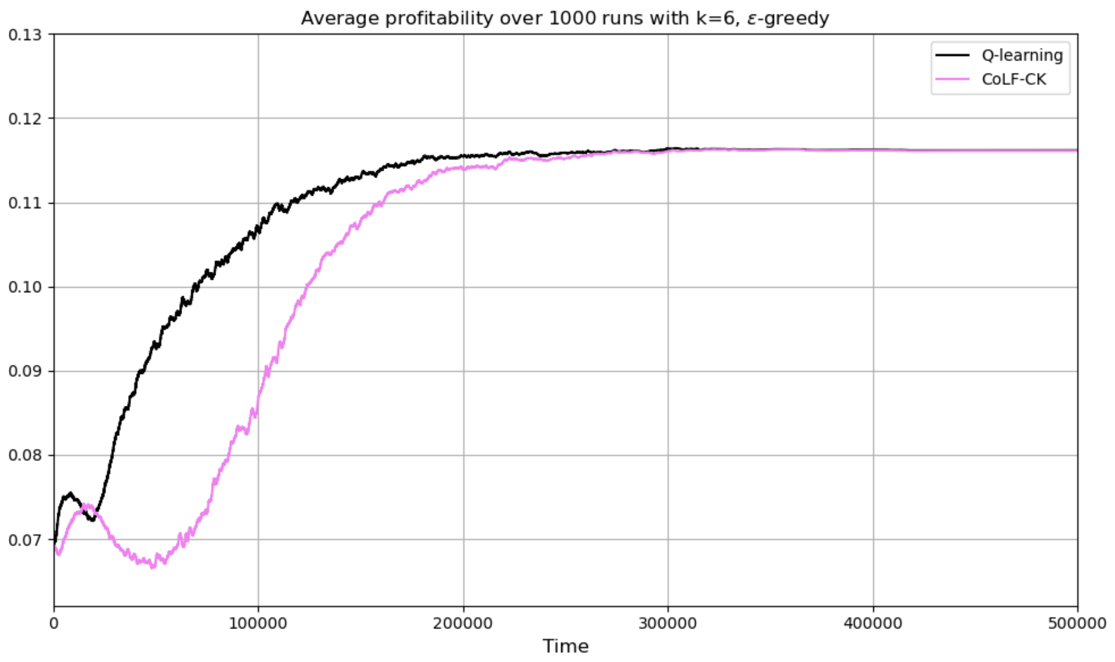


Figure 11: This figure shows the average profitability when CoLF-CK is faced against a Q-learner. 1000 runs, $T = 500.000$, $k = 6$, and using ε-greedy selection.

algorithm	Q-learning vs CoLF-CK
Type of run	
Edgeworth price cycle	7/1000
Collusive around 0.070	38/1000
Collusive around 0.111	477/1000
Collusive around 0.125 (optimal)	478/1000

Table 9: Shares of types of equilibria using ε-greedy selection. 1000 runs, 500.000 iterations, and $k = 6$. The collusive equilibria indicate which profit level the firms collude around.

From figure 11, we can see that Q-learning benefits significantly from facing a CoLF-CK learner as it converges to a higher level of average profitability than in self-play. This suggests that the principles of CoLF-CK manage to assist Q-learning in learning high collusive equilibria as shown in table 9. Similar conclusions hold when different (heterogeneous) exploration strategies are considered, and when the dimensionality is increased as shown in Appendix D.

7 Discussion

In light of the recent studies of algorithmic pricing collusion, this thesis intended to study the implications of algorithmic design on the emergence of autonomous algorithmic pricing collusion. Since most studies on the subject have only considered Q-learning, which is not inherently designed for multi-agent settings, it is of interest to see whether algorithms actually designed for multi-agent settings can lead to pricing collusion. There has been a lot thought behind the process of selecting algorithms from the MARL literature that suit the framework of the algorithmic pricing collusion literature. First of all, many algorithms designed for mixed settings within the MARL literature are based on game theory. For instance, popular game theoretic algorithms such as Nash-Q Hu and Wellman (2003), Friend-or-Foe-Q Littman (2001), and CE-Q Greenwald and Hall (2003) all require that agents can compute the possible Nash equilibria. This requires that firms have complete information of all reward functions, which in turn requires that firms have complete information of all firms demand functions and all firms cost functions[37]. Further, it may seem unlikely that firms in the real world would use game theory to compute Nash equilibria for pricing. While these game theoretic algorithms are designed for convergence to equilibrium behaviour, they are not specifically designed for fostering cooperation in mixed settings. One popular approach to foster cooperation in mixed settings is by means of communication channels, in which agents are able to communicate with each other. This communication could take the form of agents sharing their policies during learning (as in e.g. MADDPG proposed by Lowe et al. (2017)), or by exchanging external signals (this could e.g. be peer evaluation signals as in Hostallero et al. (2020)). In our setting, it is not realistic to assume that agents can communicate with each other, besides through their explicit choice of price. However, prices as a communication channel alone could potentially foster cooperation

[37]Alternatively, firms can estimate all firms demand functions and all firms cost functions, and use these to construct proxy reward functions to compute proxy Nash equilibria.

through punishment mechanisms (see e.g. Dasgupta and Musolesi (2023)), or by making the agent a predictable agent that chooses prices in a predictable manner to signalize its intentions (see e.g. Ezrachi and Stucke (2015))[38]. Alternatively, LOLA Foerster et al. (2017) and M-Qubed Crandall and Goodrich (2011) are designed to foster cooperation in mixed settings without the use of explicit communication, however both algorithms do not fit well within our setting. Specifically, LOLA is a policy gradient method with a parameterized policy that assumes other agents are naive REINFORCE[39] learners with a parameterized policy, and tries to steer the other agents' gradients to align with their own interests. As such, by explicitly considering the learning of other agents, LOLA manages to foster cooperation out of own self-interest. It does so, by estimating the parameters of other agents' policies. However, if LOLA is competing against a Q-learner, which is not gradient-based, the estimates of the Q-learner's parameters are not meaningful, since it is assumed to be a naive REINFORCE learner. With regards to M-Qubed, it relies on game theoretic concepts in the form of Nash bargaining solutions which require that the reward functions are known similar to the aforementioned game theoretic algorithms. For these reasons, I was led to algorithms that focused on adjusting the learning rates or altering the update step in order to foster cooperation in mixed settings, since these methods generally do not impose restrictive assumptions about the information available to the firm (if any).

As demonstrated in this thesis, algorithms focusing on adjusted learning rates or altered update steps can significantly increase the average profitability and the frequency of collusive equilibria compared to a simple Q-learner. These results thus provide further concerns that the emergence of autonomous pricing collusion may happen in reality, and as mentioned in Calvano et al. (2020), from the standpoint of

[38]One could argue that the CK principle in some sense is making the agent a predictable agent.

[39]REINFORCE is one of the simplest policy gradient methods proposed by Williams (1992). The idea behind policy gradient methods is basically to perform stochastic gradient ascent in the space of stochastic policies. This is feasible due to the celebrated policy gradient theorem formalized by Sutton et al. (1999). These methods somewhat resemble PHC introduced earlier.

competition policy, such findings should probably ring an alarm bell.

When it comes to the average profitability curve used as a performance metric in this thesis, it does not account for the variability in the profitability. This leaves out another story when it comes to the emergence of deterministic Edgeworth price cycles. As shown in Appendix E, frequent occurrences of these cycles induce a lot more variability in the profitability as opposed to when runs mainly consist of high collusive equilibria. In terms of profitability, there is therefore more risk involved for a firm if it engages in deterministic Edgeworth price cycle behaviour, as it may end up being the firm that repeatedly undertakes the costly action of resetting the price cycle. From an empirical perspective, it may not seem plausible that firms would engage in such behaviour. For instance, in Clemens and Rau (2022) they find that in the formation of partial cartels, firms seem to care about payoff asymmetries, i.e., partial price cartels are rejected in scenarios where an outsider firm profits excessively from the formation of the cartel. This somewhat resembles the emergence of deterministic Edgeworth price cycles studied here, except that the payoff asymmetries lie directly within the formed ”price cartel”.

For future research, I list three things that could enrich the literature of algorithmic pricing collusion. First, it may be fruitful to investigate the implications of using multi-agent algorithms in simultaneous-move frameworks. Secondly, it is left for future research to investigate the implications of using multi-agent algorithms with offline training. Firms presumably train their algorithms in artificial (offline) environments before being put to use, since the costs associated with the learning phase can be high. This raises the question whether algorithms that learned to collude offline can transfer its collusive capabilities to the true (online) environment once they are being employed. Lastly, I have considered a very stylized economic model with only two agents. A next step could therefore be to investigate the behaviour of multi-agent algorithms in more realistic models with more agents.

8 Conclusions

I have shown that when multi-agent algorithms are applied to the economic model considered in Klein (2021), the shares of high collusive equilibria can increase significantly compared to Q-learning. As a result, the multi-agent algorithms tend to converge to higher average profitability than Q-learning. Furthermore, when the complexity of the economic environment increases, the multi-agent algorithms still tend to have large shares of high collusive equilibria. This contrasts the findings in Klein (2021) that Q-learning tends to converge to deterministic Edgeworth price cycle patterns when the dimensionality is increased. The results for the multi-agent algorithms are generally more robust to a Boltzmann selection procedure unlike Q-learning, which in this case tends to converge to deterministic Edgeworth price cycle patterns far more frequently even when the dimensionality is low. By performing various robustness checks, I have shown that an increased learning horizon and a relaxation search can increase the shares of high collusive equilibria even further for the multi-agent algorithms. Moreover, I have shown that collusion remains possible when agents are heterogeneous. These findings enhance the concerns that AI algorithms used in the real world may learn to collude on prices far above the competitive level.

References

Asker, J., C. Fershtman, and A. Pakes (2021, March). Artificial Intelligence and Pricing: The Impact of Algorithm Design. Technical report, National Bureau of Economic Research.

Assad, S., R. Clark, D. Ershov, and L. Xu (2020, August). Algorithmic Pricing and Competition: Empirical Evidence from the German Retail Gasoline Market. Working Paper 1438, Economics Department, Queen's University.

Bellman, R. (1957). *Dynamic Programming.* Princeton University Press.

Bowling, M. and M. Veloso (2001). Rational and Convergent Learning in Stochastic Games. pp. 1021–1026.

Bowling, M. and M. Veloso (2002). Multiagent Learning Using a Variable Learning Rate. *Artificial Intelligence 136*(2), 215–250.

Brunskill, E. (2019). Lecture: Model-Free Policy Evaluation. *Stanford University.*

Busoniu, L., R. Babuska, and B. De Schutter (2008). A Comprehensive Survey of Multiagent Reinforcement Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 38*(2), 156–172.

Buşoniu, L., R. Babuška, and B. De Schutter (2010). *Multi-agent Reinforcement Learning: An Overview*, pp. 183–221. Springer Berlin Heidelberg.

Calvano, E., G. Calzolari, V. Denicolò, and S. Pastorello (2020, October). Artificial Intelligence, Algorithmic Pricing, and Collusion. *American Economic Review 110*(10), 3267–97.

Casella, G. and C. Robert (2004). *Monte Carlo Statistical Methods.* Springer US.

Chen, L., A. Mislove, and C. Wilson (2016). An Empirical Analysis of Algorithmic Pricing on Amazon Marketplace. pp. 1339–1349.

Clemens, G. and H. A. Rau (2022, September). Either With Us or Against Us: Experimental Evidence on Partial Cartels. *Theory and Decision 93*(2), 237–257.

Crandall, J. and M. Goodrich (2011, 03). Learning to Compete, Coordinate, and Cooperate in Repeated Games using Reinforcement Learning. *Machine Learning 82*, 281–314.

Danish Competition and Consumer Authority (2021). Prisalgoritmer og deres betydning for konkurrencen.

Dasgupta, N. and M. Musolesi (2023). Investigating the Impact of Direct Punishment on the Emergence of Cooperation in Multi-Agent Reinforcement Learning Systems.

de Cote, E. M., A. Lazaric, and M. Restelli (2006). Learning to Cooperate in Multi-Agent Social Dilemmas. AAMAS '06, New York, NY, USA, pp. 783–785. Association for Computing Machinery.

European Commission (2020). Antitrust: Commission Consults Stakeholders on a Possible New Competition Tool.

Ezrachi, A. and M. Stucke (2015, 01). Artificial Intelligence Collusion: When Computers Inhibit Competition. *SSRN Electronic Journal 2017*.

Foerster, J. N., R. Y. Chen, M. Al-Shedivat, S. Whiteson, P. Abbeel, and I. Mordatch (2017). Learning with Opponent-Learning Awareness.

Greenwald, A. and K. Hall (2003). Correlated-Q Learning. pp. 242–249.

Hostallero, D. E., D. Kim, S. Moon, K. Son, W. J. Kang, and Y. Yi (2020). Inducing Cooperation through Reward Reshaping Based on Peer Evaluations in Deep Multi-Agent Reinforcement Learning. pp. 520–528.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes.* Cambridge, MA: MIT Press.

Hu, J. and M. Wellman (2003, 01). Nash Q-Learning for General-Sum Stochastic Games. *Journal of Machine Learning Research 4*, 1039–1069.

Klein, T. (2021, 09). Autonomous Algorithmic Collusion: Q-learning under Sequential Pricing. *RAND Journal of Economics 52*(3), 538–558.

Littman, M. L. (2001). Friend-or-Foe Q-Learning in General-Sum Games. pp. 322–328.

Liu, L. and W. Jia (2021, 05). The Value Function with Regret Minimization Algorithm for Solving the Nash Equilibrium of Multi-Agent Stochastic Game. *International Journal of Computational Intelligence Systems 14*.

Lowe, R., Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch (2017). Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *CoRR*.

Maskin, E. and J. Tirole (1988, May). A Theory of Dynamic Oligopoly, II: Price Competition, Kinked Demand Curves, and Edgeworth Cycles. *Econometrica 56*(3), 571–599.

Matignon, L., G. Laurent, and N. Fort-Piat (2012, 03). Independent Reinforcement Learners in Cooperative Markov Games: A Survey Regarding Coordination Problems. *The Knowledge Engineering Review 27*, 1 – 31.

Nowé, A., P. Vrancx, and Y.-M. De Hauwere (2012). *Game Theory and Multi-Agent Reinforcement Learning*, pp. 441–470. Berlin, Heidelberg: Springer Berlin Heidelberg.

Sandholm, T. W. and R. H. Crites (1996). Multiagent Reinforcement Learning in the Iterated Prisoner's Dilemma. *Biosystems 37*(1), 147–166.

Silver, D. (2015). Lecture: Model-Free Prediction. *University College London.*

Singh, S., R. Sutton, and P. Kaelbling (1996, 11). Reinforcement Learning with Replacing Eligibility Traces. *Machine Learning 22.*

Stimpson, J. and M. Goodrich (2003, 01). Learning To Cooperate in a Social Dilemma: A Satisficing Approach to Bargaining. *Proceedings, Twentieth International Conference on Machine Learning 2*, 728–735.

Stimpson, J. L., M. A. Goodrich, and L. C. Walters (2001). Satisficing and Learning Cooperation in the Prisoner's Dilemma. In *Proceedings of the 17th International*

Joint Conference on Artificial Intelligence - Volume 1, pp. 535–540. Morgan Kaufmann Publishers Inc.

Sutton, R. S. and A. G. Barto (2018). *Reinforcement Learning: An Introduction* (Second ed.). The MIT Press.

Sutton, R. S., D. McAllester, S. Singh, and Y. Mansour (1999). Policy Gradient Methods for Reinforcement Learning with Function Approximation. pp. 1057–1063.

Waltman, L. and U. Kaymak (2008). Q-learning Agents in a Cournot Oligopoly Model. *Journal of Economic Dynamics and Control 32*(10), 3275–3293.

Watkins, C. and P. Dayan (1992, 05). Technical Note: Q-Learning. *Machine Learning 8*, 279–292.

Watkins, C. J. C. H. (1989). Learning From Delayed Rewards.

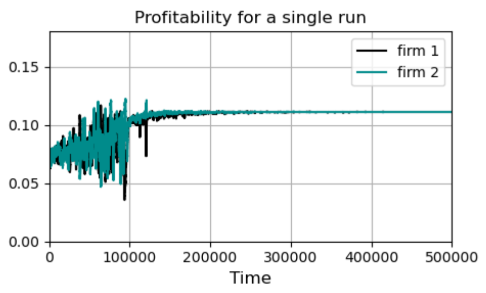
Williams, R. J. (1992, may). Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn. 8*(3–4), 229–256.

Xie, Q., Y. Chen, Z. Wang, and Z. Yang (2020). Learning Zero-Sum Simultaneous-Move Markov Games Using Function Approximation and Correlated Equilibrium. *CoRR*.

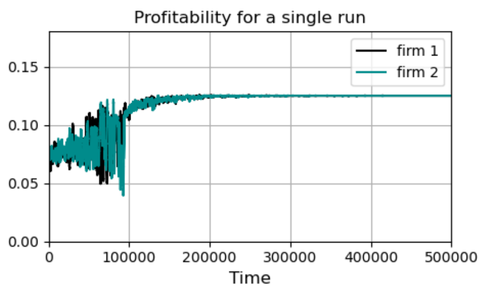
Zhang, K., Z. Yang, and T. Başar (2021). Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms.

Appendices

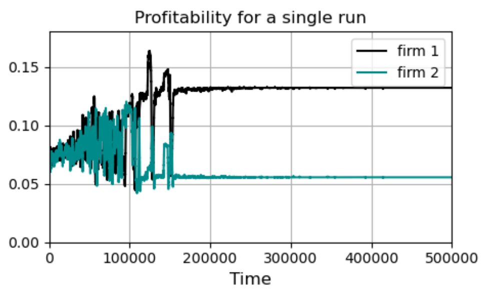
A Convergence Patterns with $k = 6$



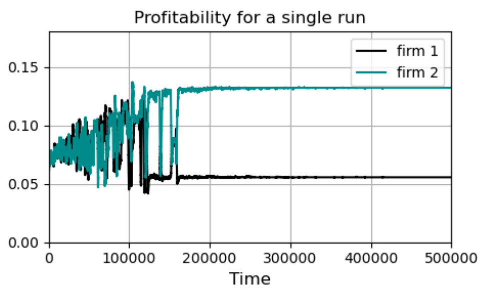
(a) Non-optimal collusive Nash equilibrium.



(b) Optimal collusive joint-profit maximizing Nash equilibrium.

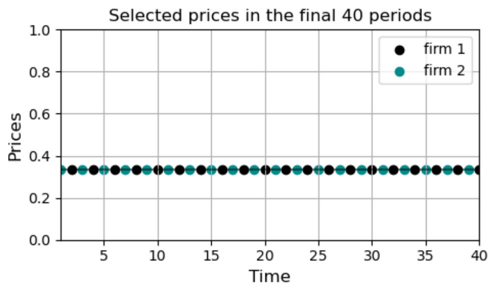


(c) Edgeworth price cycle where firm 1 free-rides.

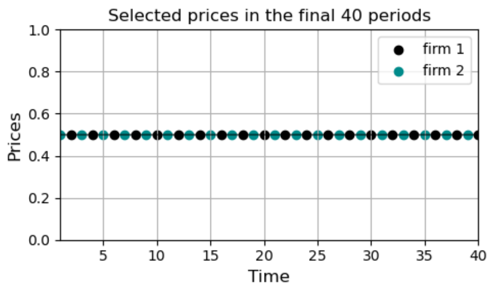


(d) Edgeworth price cycle where firm 2 free-rides.

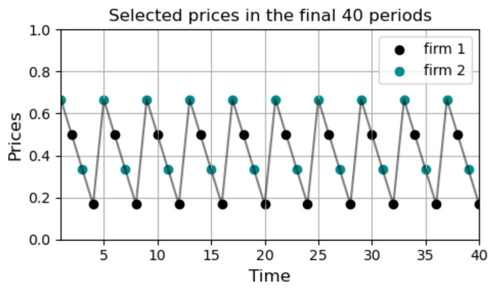
Figure 12: This figure shows the possible profitability curve scenarios for single runs.



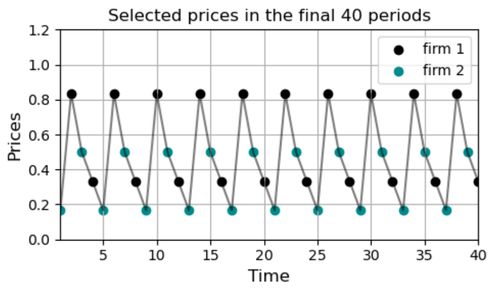
(a) Non-optimal collusive Nash equilibrium.



(b) Optimal collusive joint-profit maximizing Nash equilibrium.



(c) Edgeworth price cycle where firm 1 free-rides.



(d) Edgeworth price cycle where firm 2 free-rides.

Figure 13: This figure shows the selected prices in the final 40 periods associated with the profitability curves in figure 12. In **(c)** and **(d)**, it is always the same firm that undertakes the costly action of resetting the price cycle.

B Increased Learning Horizon

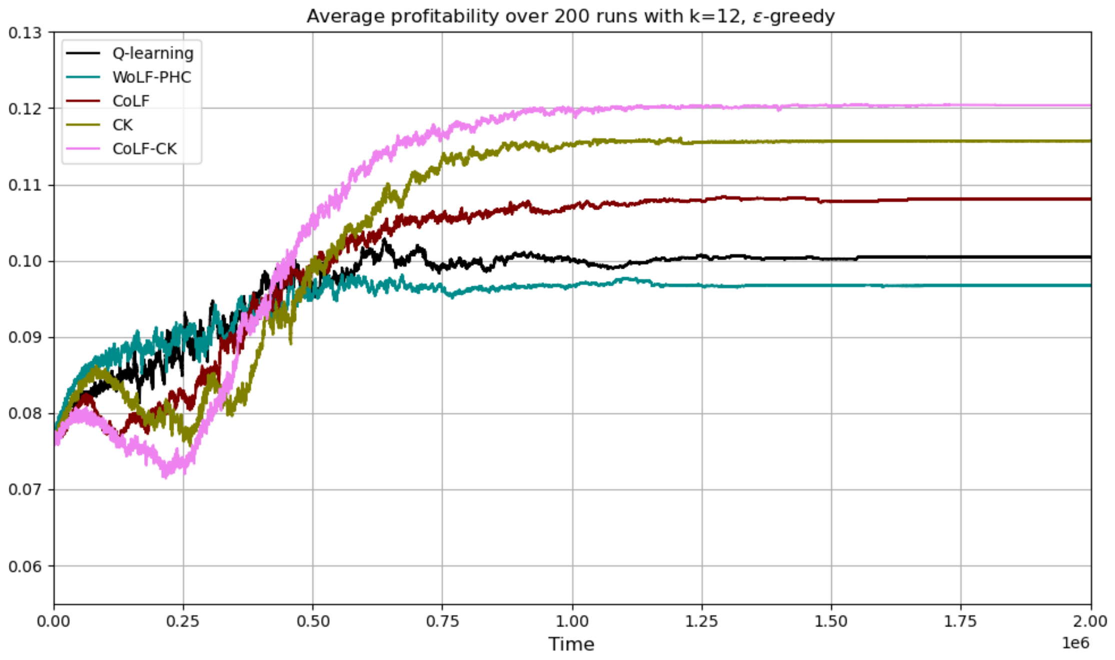


Figure 14: This figure shows the average profitability for the considered algorithms over 200 runs with $T = 2.000.000$, $k = 12$, and using ε-greedy selection.

algorithm	Q-learning	WoLF-PHC	CoLF	CK	CoLF-CK
Type of run					
Edgeworth price cycle	156/200	186/200	55/200	5/200	1/200
Collusive around 0.038	0/200	0/200	0/200	0/200	0/200
Collusive around 0.070	0/200	0/200	0/200	1/200	0/200
Collusive around 0.094	1/200	0/200	25/200	12/200	2/200
Collusive around 0.111	13/200	5/200	45/200	67/200	38/200
Collusive around 0.122	25/200	8/200	49/200	81/200	87/200
Collusive around 0.125 (optimal)	5/200	1/200	26/200	34/200	72/200

Table 10: Shares of types of equilibria using ε-greedy selection. 200 runs, 2.000.000 iterations, and $k = 12$. The collusive equilibria indicate which profit level the firms collude around.

C Initialization of Q-values

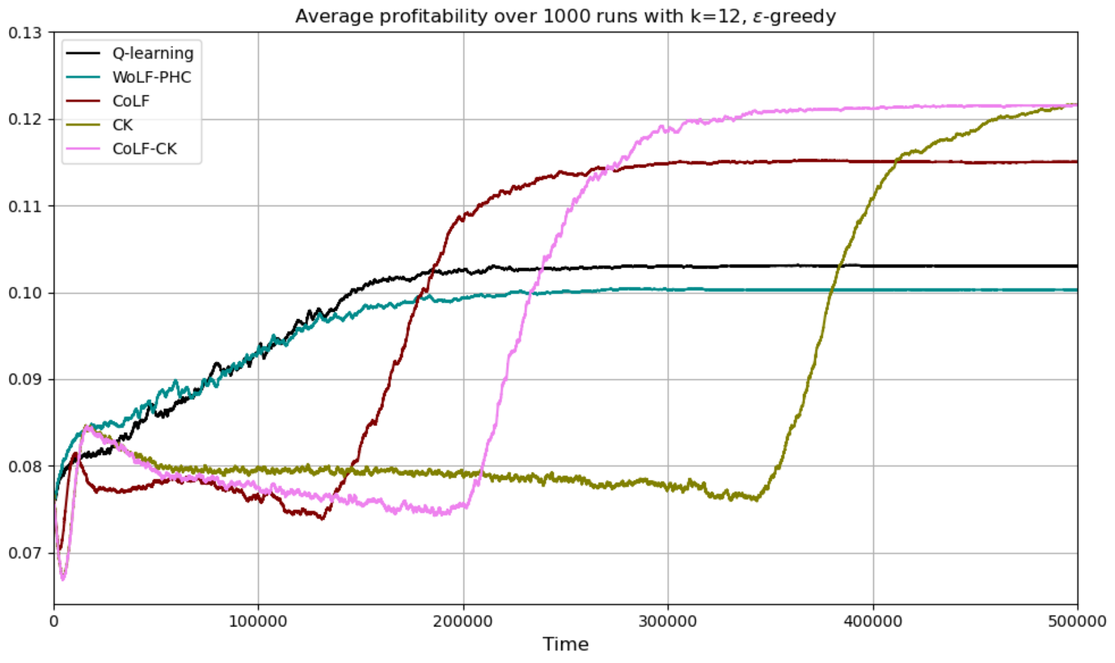


Figure 15: This figure shows the average profitability with a relaxation search for the considered algorithms over 1000 runs with $T = 500.000$, $k = 12$, and using ε-greedy selection.

algorithm	Q-learning	WoLF-PHC	CoLF	CK	CoLF-CK
Type of run					
Edgeworth price cycle	832/1000	929/1000	266/1000	9/1000	14/1000
Collusive around 0.038	0/1000	0/1000	0/1000	0/1000	0/1000
Collusive around 0.070	0/1000	0/1000	1/1000	0/1000	2/1000
Collusive around 0.094	7/1000	1/1000	25/1000	3/1000	6/1000
Collusive around 0.111	42/1000	24/1000	156/1000	42/1000	69/1000
Collusive around 0.122	75/1000	34/1000	358/1000	497/1000	493/1000
Collusive around 0.125 (optimal)	44/1000	12/1000	194/1000	432/1000	416/1000

Table 11: Shares of types of equilibria using ε-greedy selection and relaxation search. 1000 runs, 500.000 iterations, and $k = 12$. The collusive equilibria indicate which profit level the firms collude around. Note that CK did not converge in 17 runs.

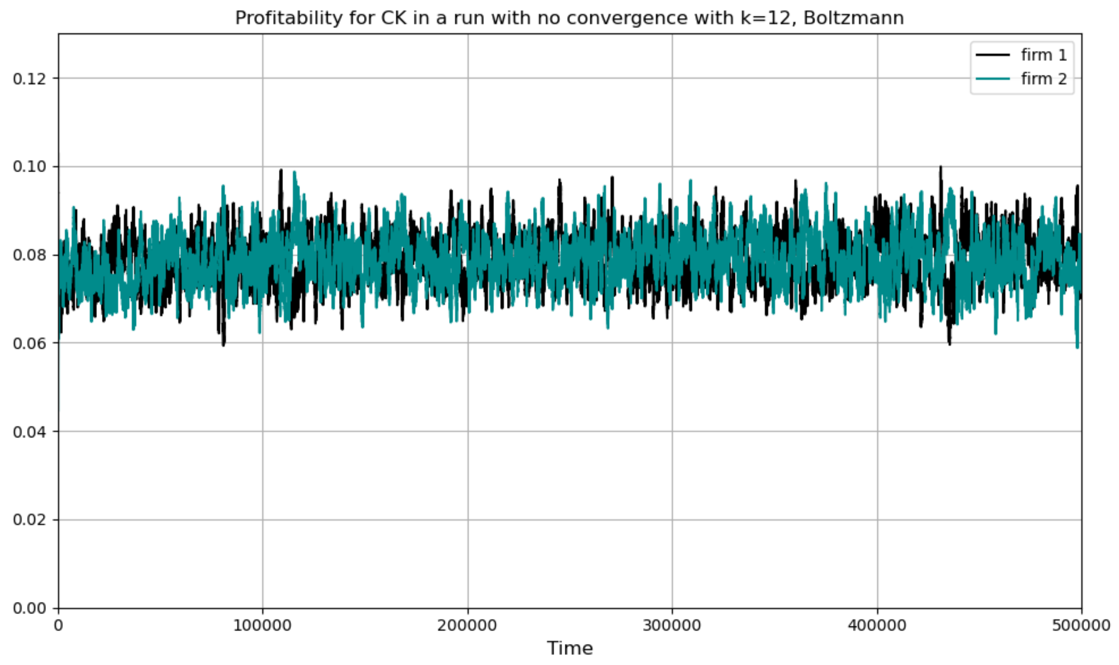


Figure 16: This figure shows how CK oscillates around profits of 0.075-0.080 in runs where CK did not converge using a relaxation search. $T = 500.000$, $k = 12$, and using Boltzmann selection.

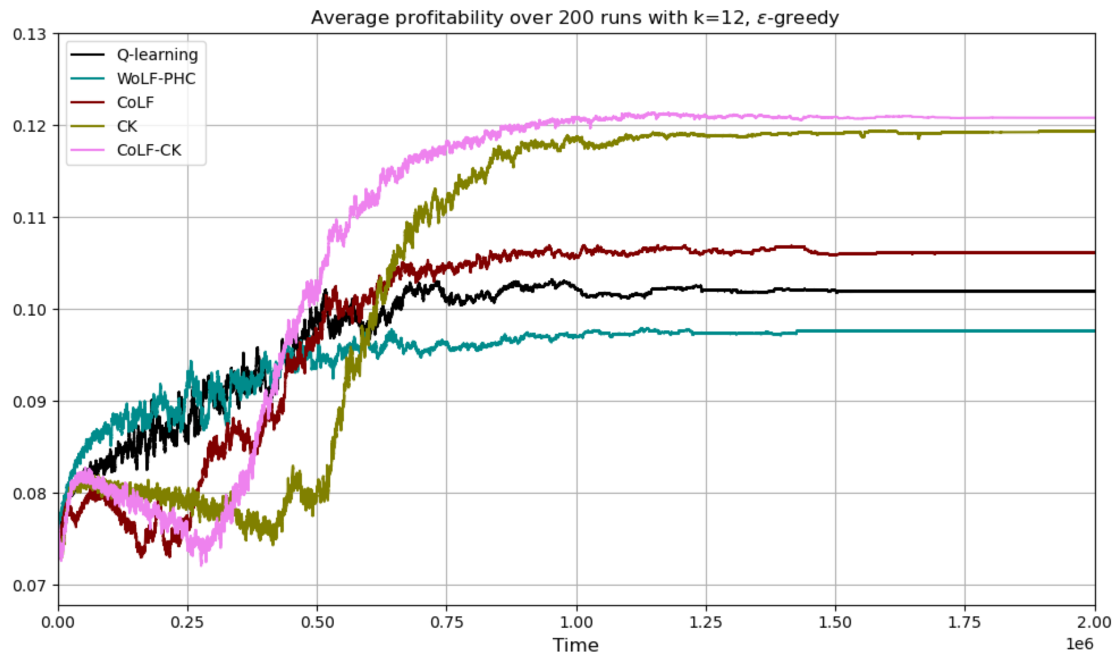


Figure 17: This figure shows the average profitability with a relaxation search for the considered algorithms over 200 runs with $T = 2.000.000$, $k = 12$, and using ε-greedy selection.

algorithm	Q-learning	WoLF-PHC	CoLF	CK	CoLF-CK
Type of run					
Edgeworth price cycle	148/200	182/200	69/200	5/200	0/200
Collusive around 0.038	0/200	0/200	0/200	0/200	0/200
Collusive around 0.070	0/200	0/200	0/200	0/200	0/200
Collusive around 0.094	1/200	0/200	17/200	6/200	3/200
Collusive around 0.111	8/200	5/200	42/200	39/200	27/200
Collusive around 0.122	29/200	10/200	50/200	93/200	106/200
Collusive around 0.125 (optimal)	14/200	3/200	22/200	57/200	64/200

Table 12: Shares of types of equilibria using ε-greedy selection and relaxation search. 200 runs, 2.000.000 iterations, and $k = 12$. The collusive equilibria indicate which profit level the firms collude around.

D Heterogeneous Agents

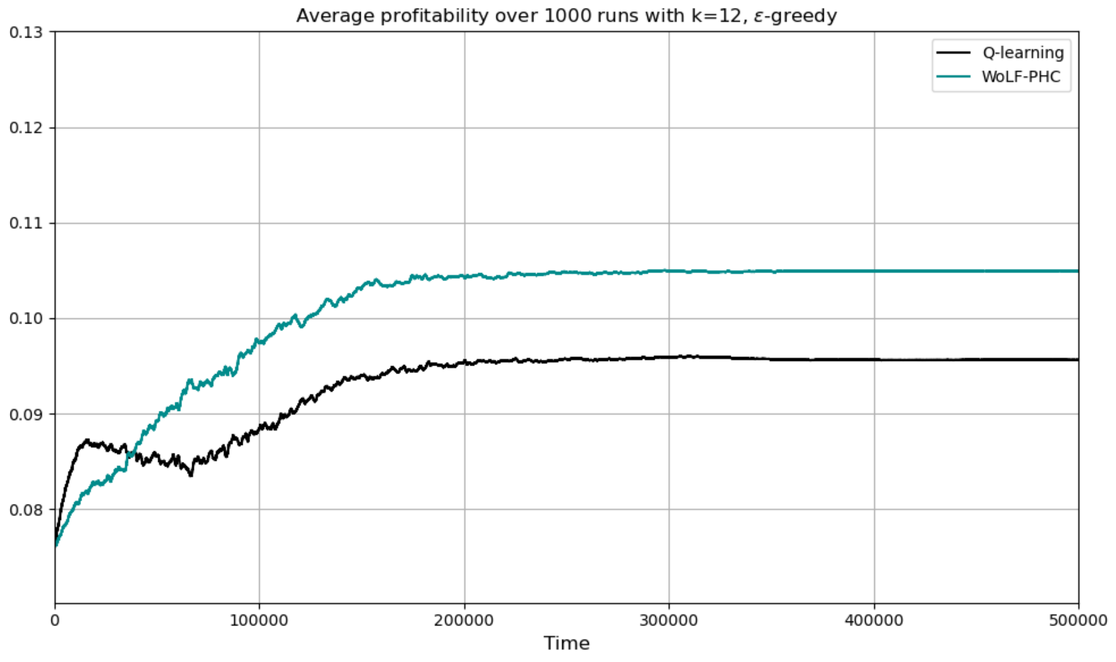


Figure 18: This figure shows the average profitability when WoLF-PHC is faced against a Q-learner. 1000 runs, $T = 500.000$, $k = 12$, and using ε-greedy selection.

algorithm	Q-learning vs WoLF-PHC
Type of run	
Edgeworth price cycle	902/1000
Collusive around 0.038	0/1000
Collusive around 0.070	0/1000
Collusive around 0.094	1/1000
Collusive around 0.111	28/1000
Collusive around 0.122	54/1000
Collusive around 0.125 (optimal)	15/1000

Table 13: Shares of types of equilibria using ε-greedy selection when WoLF-PHC is faced against a Q-learner. 1000 runs, 500.000 iterations, and $k = 12$. The collusive equilibria indicate which profit level the firms collude around.

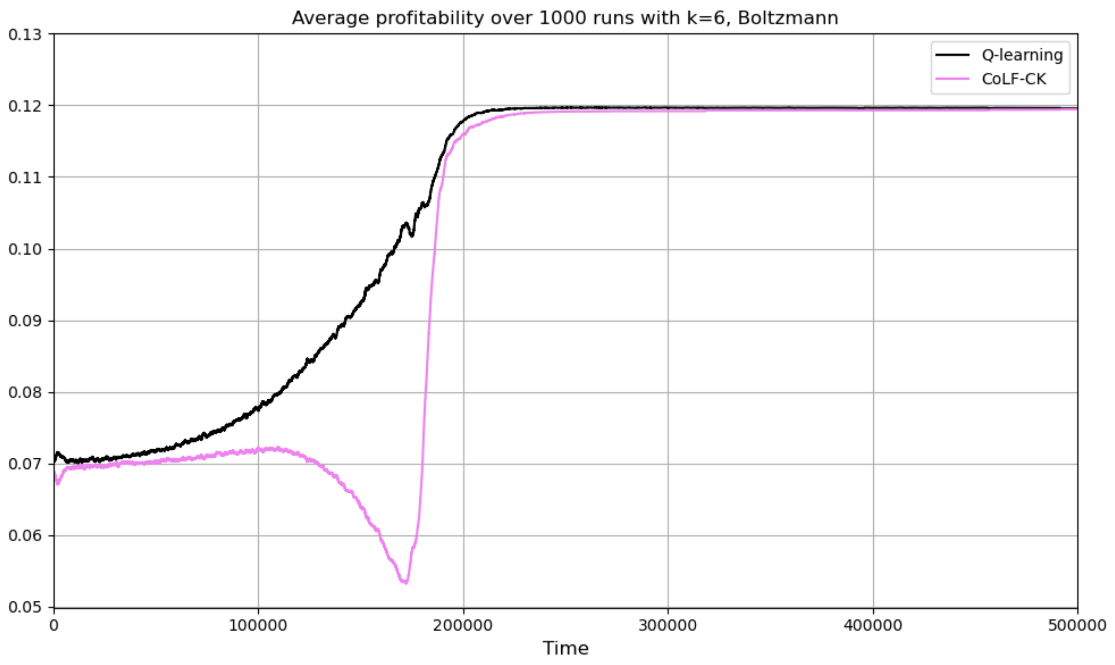


Figure 19: This figure shows the average profitability when CoLF-CK is faced against a Q-learner. 1000 runs, $T = 500.000$, $k = 6$, and using Boltzmann selection.

algorithm	Q-learning vs CoLF-CK
Type of run	
Edgeworth price cycle	2/1000
Collusive around 0.070	3/1000
Collusive around 0.111	377/1000
Collusive around 0.125 (optimal)	618/1000

Table 14: Shares of types of equilibria when CoLF-CK is faced against a Q-learner. 1000 runs, 500.000 iterations, $k = 6$, and using Boltzmann selection. The collusive equilibria indicate which profit level the firms collude around.

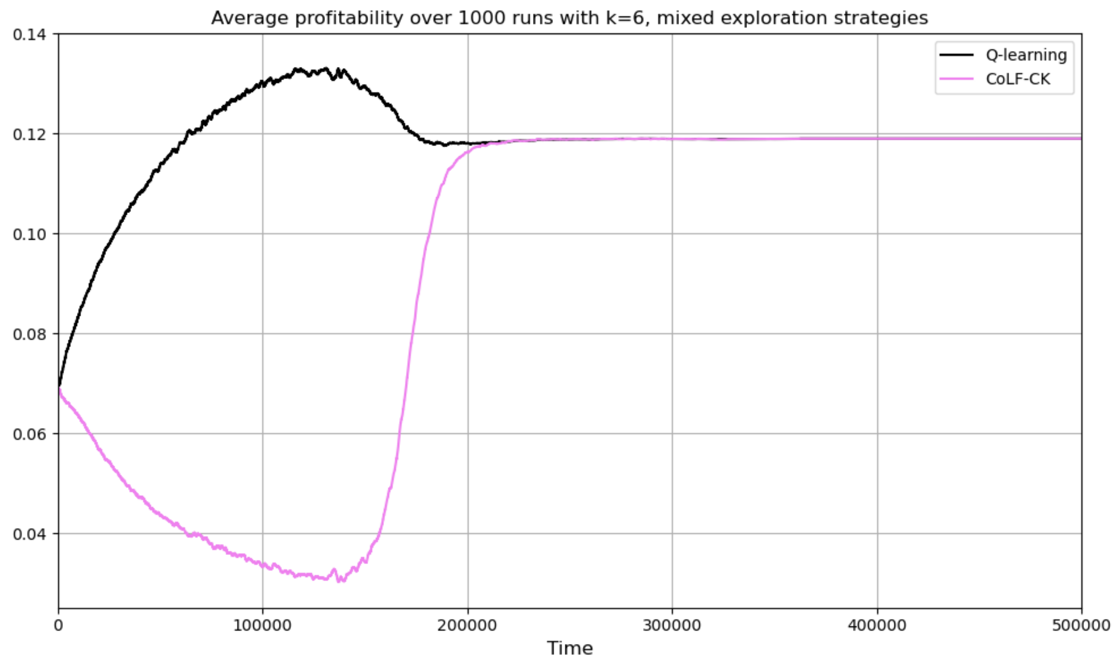


Figure 20: This figure shows the average profitability when a Boltzmann CoLF-CK is faced against an ε-greedy Q-learner. 1000 runs with $T = 500.000$ and $k = 6$.

algorithm	Q-learning ε-greedy vs CoLF-CK Boltzmann
Type of run	
Edgeworth price cycle	2/1000
Collusive around 0.070	7/1000
Collusive around 0.111	401/1000
Collusive around 0.125 (optimal)	590/1000

Table 15: Shares of types of equilibria using mixed exploration strategies. 1000 runs, 500.000 iterations, and $k = 6$. The collusive equilibria indicate which profit level the firms collude around.

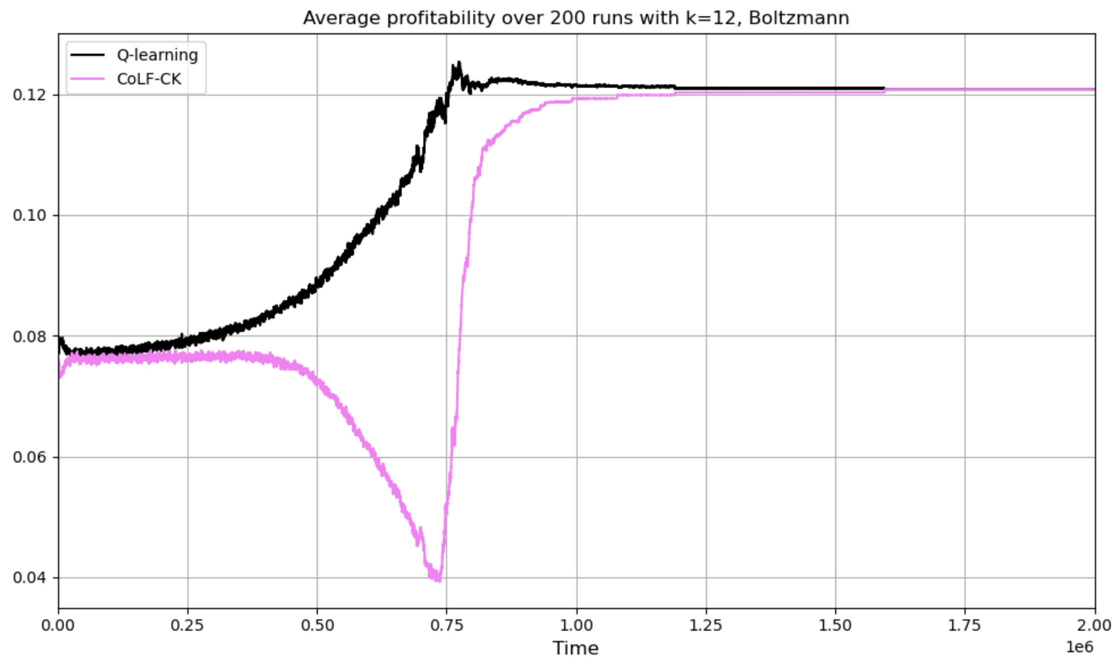


Figure 21: This figure shows the average profitability when CoLF-CK is faced against a Q-learner. 200 runs, $T = 2.000.000$, $k = 12$, and using Boltzmann selection.

algorithm	Q-learning vs CoLF-CK
Type of run	
Edgeworth price cycle	0/200
Collusive around 0.038	0/200
Collusive around 0.070	0/200
Collusive around 0.094	3/200
Collusive around 0.111	32/200
Collusive around 0.122	87/200
Collusive around 0.125 (optimal)	78/200

Table 16: Shares of types of equilibria using Boltzmann selection. 200 runs, 2.000.000 iterations, and $k = 12$. The collusive equilibria indicate which profit level the firms collude around.

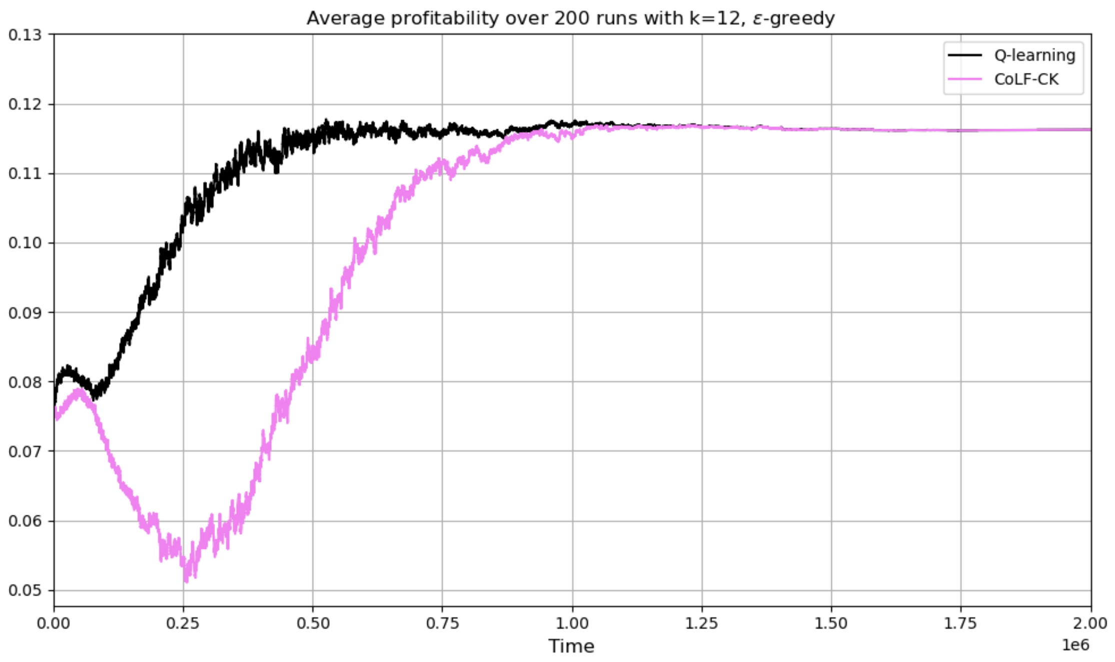


Figure 22: This figure shows the average profitability when CoLF-CK is faced against a Q-learner. 200 runs, $T = 2.000.000$, $k = 12$, and using ε-greedy selection.

algorithm	Q-learning vs CoLF-CK
Type of run	
Edgeworth price cycle	2/200
Collusive around 0.038	0/200
Collusive around 0.070	2/200
Collusive around 0.094	17/200
Collusive around 0.111	56/200
Collusive around 0.122	90/200
Collusive around 0.125 (optimal)	33/200

Table 17: Shares of types of equilibria using ε-greedy selection. 200 runs, 2.000.000 iterations, and $k = 12$. The collusive equilibria indicate which profit level the firms collude around.

E Variability of Profitability

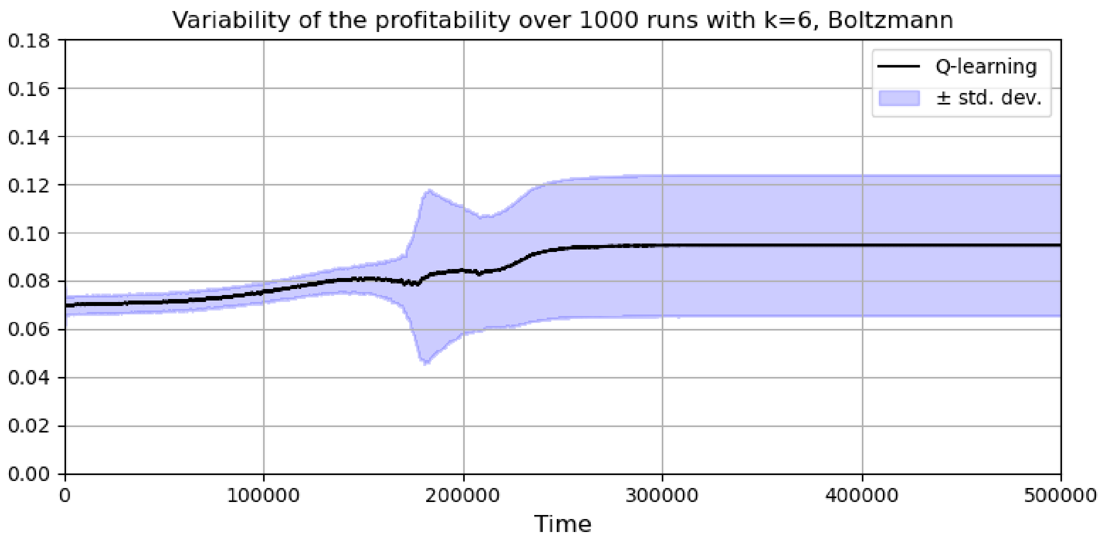


Figure 23: This figure shows the variability in the profitability for Q-learning in the baseline setup with Boltzmann selection. Variability is here expressed as ± the standard deviation of the profitability calculated at each time step.

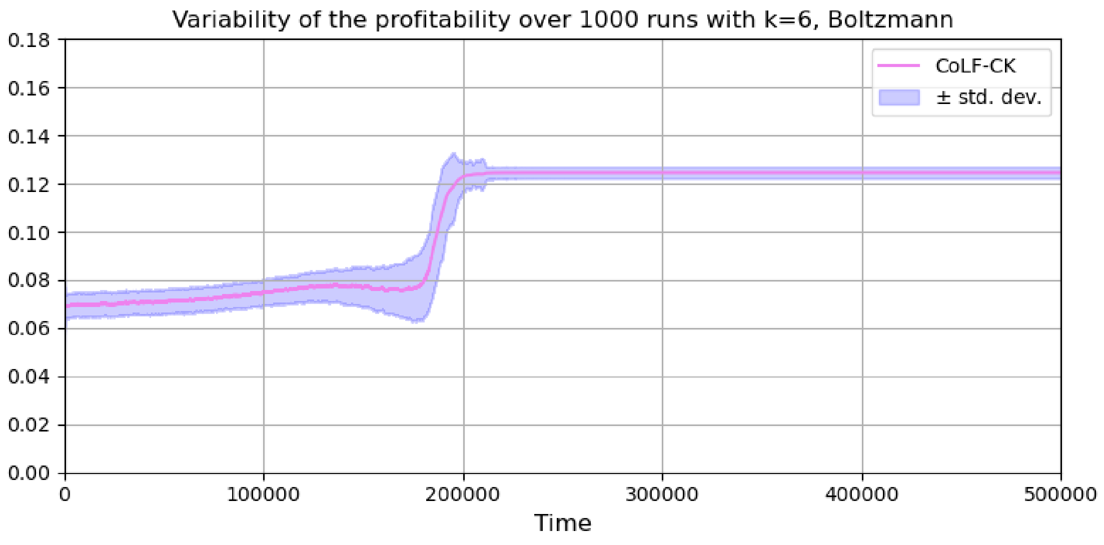


Figure 24: This figure shows the variability in the profitability for CoLF-CK in the baseline setup with Boltzmann selection. Variability is here expressed as ± the standard deviation of the profitability calculated at each time step.