POLITECNICO DI MILANO
Master's degree in Computer Science and Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

Learning Nash Equilibria in Simulation-based Games: a Best Arm Identification Approach

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Relatore: Nicola Gatti
Correlatore: Alberto Marchesi
Correlatore: Francesco Trovò

Tesi di Laurea di:
Federica Gianotti, matricola 920618

Anno Accademico 2019-2020

Abstract

We tackle a fundamental problem in the field of *empirical game-theoretic analysis* (EGTA), that is learning equilibria of *simulation-based games*. In such games, an explicit analytical description of the players' utilities is not available, as they are rather specified by means of a game simulator that can be queried to obtain noisy estimates of the utilities. Since running the simulator during the game is not feasible, there is the need to design algorithms that perform a preliminary pure exploration phase to learn the (approximate) equilibria, and then use them to play the game. As a consequence, solving these games calls for algorithms that iteratively sample the players' stochastic utilities, and, then, output some strategy profiles which represent good candidates for being (approximate) equilibria.

In our work, we study the general case of finding Nash equilibria (NE) in *n-player general-sum* games. We introduce a *multi-armed bandit* model and propose four *dynamic querying algorithms* that employ best arm identification techniques able to solve this task. We work both in the *fixed-confidence* setting, guaranteeing a desired confidence level while trying to minimize the total number of queries to the simulator, and in the *fixed-budget* setting, guaranteeing to not exceed the maximum number of queries while trying to reach the highest level of confidence possible. In particular, we propose two algorithms to learn one or more pure NEs in the fixed-confidence setting (PNE-LUCB and LUCB-NG), one algorithm to find a single mixed NE in the fixed-confidence setting (MNE-LUCB), and, finally, one algorithm to find all the pure NEs in the fixed-budget setting (SE). Then, we evaluate all our algorithms against each other on a wide set of simulated games with two or more players. With regard to the search for pure NEs the best performing algorithm resulted to be SE, while, among

the two algorithms in the fixed-confidence setting, PNE-LUCB showed to have better performances than LUCB-NG.

Sommario

Affrontiamo un problema fondamentale nel campo dell'*analisi empirica della teoria dei giochi*, ovvero apprendere equilibri nei cosiddetti simulaton-based games. In questo tipo di giochi una descrizione analitica esplicita delle utilità dei giocatori non è disponibile, in quanto essa è specificata tramite un simulatore che può essere interrogato per ottenerne stime rumorose di tale utilità. Poiché eseguire la simulazione durante l'esecuzione del gioco non è possibile, è necessario creare algoritmi che compiano una fase di pura esplorazione preliminare per apprendere gli equilibri (approssimati), e poi usare questi ultimi per durante l'esecuzione del gioco. Di conseguenza, risolvere questi giochi richiede algoritmi che campionino iterativamente l'utilità stocastica dei giocatori e poi producano dei profili di strategie che siano buoni candidati ad essere equilibri (approssimati).

Nel nostro lavoro studiamo il caso generale di trovare equilibri di Nash (NE) in giochi con *n-giocatori* a *somma-generale*. Introduciamo un modello di tipo *multi-armed bandit* e proponiamo quattro *algoritmi di interrogazione dinamica* che impiegano tecniche per la cosiddetta best-arm identification per risolvere il problema sopracitato. Diamo risultati sia nel setting a *confidenza fissata*, garantendo un livello di confidenza desiderato minimizzando, allo stesso tempo, il numero totale delle interrogazioni al simulatore, sia nel setting a *budget fissato*, garantendo di non superare un massimo numero di interrogazioni al simulatore mentre cerchiamo di raggiungere il più alto livello di confidenza possibile. In particolare, proponiamo due algoritmi per apprendere uno o più NEs nel setting a confidenza fissata, un algoritmo per trovare un singolo NE misto nel setting a confidenza fissata e, infine, un algoritmo per trovare tutti i NEs puri nel setting a budget fissato. Abbiamo valutato i nostri algoritmi uno contro l'altro in un ampio

set di giochi a due o più giocatori. Per quanto riguarda la ricerca di NEs puri, l'algoritmo più performante è risultato essere SE, mentre, tra i due algoritmi nel setting a confidenza fissata, PNE-LUCB ha mostrato di avere prestazioni migliori di LUCB-NG.

Contents

Abstract **I**

Sommario **V**

1 Introduction **1**

2 Preliminaries **5**
2.1 Game Theory 5
2.1.1 Normal-form games 5
2.1.2 Simulation-based games 6
2.1.3 Strategy representation 7
2.1.4 Nash equilibrium 8
2.2 Machine Learning 10
2.2.1 Multi-Armed Bandit 11
2.2.2 Best Arm Identification in Multi-Armed Bandit 12
2.2.3 PAC learning 13

3 Problem formulation **15**
3.1 Fixed-confidence setting 15
3.1.1 Learning pure Nash equilibria 15
3.1.2 Learning mixed Nash equilbria 16
3.2 Fixed-budget setting 16
3.2.1 Learning pure Nash equilibria 16
3.3 Dynamic querying algorithms for simulation-based games 17

4 State of the art **19**
4.1 General background 19

4.2 EGTA with best arm identification techniques and PAC guarantees 21
4.2.1 Fixed-confidence setting 21
4.2.2 Fixed-budget setting 24

5 Algorithms for identifying Nash Equilibria **25**
5.1 PNE-LUCB 25
5.1.1 Pseudocode 28
5.1.2 Example 30
5.2 Theoretical analysis 32
5.3 LUCB-NG 34
5.3.1 Pseudocode 38
5.3.2 Example 40
5.4 MNE-LUCB 43
5.4.1 Pseudocode 47
5.4.2 Examples 49
5.5 SE 55
5.5.1 Pseudocode 59
5.5.2 Example 60

6 Experiments **65**
6.1 Experimental setting 65
6.1.1 Data generation 65
6.1.2 Bound definition 66
6.1.3 Performance indexes 67
6.2 Results 68
6.2.1 Looking for pure NEs in 2-player games . . 69
6.2.2 Looking for pure NEs in 3-player games . . 72
6.2.3 Looking for a mixed NE 73
6.3 Comments 74

7 Conclusion and Future works **77**
7.1 Future works 78

Bibliografia **79**

A Additional experimental results **85**
A.1 Looking for pure NEs in 2-player games 85
A.2 Looking for pure NEs in 3-player games 90

Chapter 1

Introduction

Game theory studied scenarios in which opposing *players* compete with each other. Each player has a number of *actions* available and, according to the one he/she chooses to play and to the decisions of the other players, everybody obtains some reward, better called *utility*. Analyzing a game means predicting its outcome, that is predicting how each player will play. The most famous and adopted type of solution is the *Nash Equilibrium* (NE): an NE is a combination of actions of all the players such that none of them has any incentive to change his/her choice, i.e. none of them can obtain a greater utility by changing his/her action, given that the other players do not either. Finding one or more NEs in a game where a complete analytical description of the players' utilities for each combination of actions is available is a fairly simple task, but this is not the case in most real-world situations. In practice, we are often given access only to samples of data generated from some real phenomena, and so the problem becomes more complex, since before being able to compute the NEs of the game in question, we need to be able to estimate the real utilities. The field that studies this kind of games is know as *empirical game-theoretic analysis* (EGTA). In the analysis of such games the utilities are generally assumed to be generated by some random variables, and a *game simulator*, also called *oracle*, can be queried to obtain noisy estimates of them. Games defined this way are named *simulation-based games*. This setting is more realistic, since in practice randomness can naturally arise because of several factors, like incomplete information, noise. A simple example is the coin flipping game, where two players throw a coin and try to guess its landing side. Since the physical process of the landing of the coin can be influenced by many factors and the utilities of the players depend on that, then they are random.

The problem that we face in this work is to find NEs in *n-player* simulation-

based games. To solve this problem many different algorithms can be designed. However, they can be generally divided into two groups: those that make a complete first estimate of the utilities of the whole game and then, in a second step, compute the NEs of the estimated game, and those that design specific selection rules that prescribe which queries need to be performed to the oracle in order to minimize their number and focus only on "interesting" parts of the game. The algorithms that we propose belong to this second category. In particular, we introduce a *multi-armed bandit* model and we treat our problem as a *best arm identification* problem, which is a classic *reinforcement learning* problem.

In the following, we provide a brief description of the content of each chapter composing our work:

- Chapter 2: in this chapter we provide all the theoretical background of the concepts we use in our work. In particular, we start with an introduction about game theory, giving a particular attention to simulation-based games, and then we move to a brief overview of some machine learning concepts, focusing on Multi-Armed Bandit and PAC learning;

- Chapter 3: in this chapter we provide the formalization of the problem faced by our work, specifying the different settings in which we have developed our algorithms and providing the description of dynamic querying algorithms;

- Chapter 4: in this chapter we first present a general overview of the most important works and solutions that have been proposed until now for the problem we are facing, then we move to a more precise description of the works whose approach is closer to ours and from which we take more inspiration;

- Chapter 5: in this chapter we propose four algorithms, PNE-LUCB and LUCB-NG, for the identification of one or more pure Nash equilibria in a simulation-based game in the fixed-confidence setting, MNE-LUCB, for the identification of a mixed Nash equilibrium in a simulation-based game in the fixed-confidence setting and, finally, SE, for the identification of all the pure Nash equilibria in a simulation-based game in the fixed-budget setting;

- Chapter 6: in this chapter we first describe the way in which the games on which we run the tests have been generated and the parameters that are used for the evaluation, then we provide and comment the obtained results;

- Chapter 7: in this chapter we draw the conclusions of our work and propose possible future research directions.

Chapter 2

Preliminaries

In this chapter we present the main theoretical background of the concepts that are treated and used in our work. In particular, we start with some formal definitions about game theory, with a particular attention to simulation-based games, and then we move to a brief introduction to machine learning, focusing on Multi-Armed Bandit problems and PAC learning.

2.1 Game Theory

Game theory is the study of mathematical models, called *Games*, that represent situations in which there are conflicting interests among opposing players.

2.1.1 Normal-form games

The *Games* can be represented in different ways according to the type of situation that wants to be analyzed. The two most commonly adopted representations in literature are *normal-form game* and *extensive-form game*. In this work, since we only consider games in which all the players act simultaneously, we always use the first representation, while the second one is used in situations in which the players act sequentially.

Definition 1. *A* normal-form game $\Gamma \coloneqq (N, S, \boldsymbol{u})$ *is a tuple defined as follows:*

- $N \coloneqq \{1, \ldots, n\}$ *is a finite set of players;*
- $S \coloneqq \bigtimes_{i \in N} S_i$ *is a set of* strategy *(or* action*)* profiles $\boldsymbol{s} = (s_i)_{i \in N}$*, where, for each* $i \in N$*, the finite set* S_i *defines player* i*'s* strategies *(or* actions*);*

- $\boldsymbol{u} : S \to \mathbb{R}^n$ *is a utility vector function that assigns each strategy profile* $\boldsymbol{s} \in S$ *to a vector of players' utilities* $\boldsymbol{u}(\boldsymbol{s})$*, whose* i*-th component* $u_i(\boldsymbol{s})$ *is player* i*'s utility if the players play according to* $\boldsymbol{s}$*.*

An example of a *normal-form game* with 2 players and 3 strategies per player can be found in Figure 2.1, where, accordingly to the previous definition, $N = \{1, 2\}$, $S_1 = \{T, M, B\}$, and $S_2 = \{L, C, R\}$ and the utilities of each player at every strategy profile are the ones reported in the corresponding entry.

	L	C	R
T	4,5	0,1	0,2
M	0,0	1,1	1,2
B	1,1	0,2	0,1

Figure 2.1: Normal-form game.

2.1.2 Simulation-based games

Simulation-based games were firstly introduced by Vorobeychik and Wellman [42] and represent situations in which an explicit analytical description of the players' utilities is not available, as they are rather specified by means of a *game simulator* that can be queried to obtain noisy estimates of the utilities. Formally, there is an *oracle* $\mathcal{O}$ that produces, given a joint strategy profile, a (possibly noisy) sample from the players' joint utility functions. The study of this kind of games is known as *empirical game-theoretic analysis* (EGTA).

Definition 2. *A* simulation-based normal-form game $\Gamma \coloneqq (N, S, \tilde{\boldsymbol{u}})$ *is a tuple defined as follows:*

- N *is a finite set of players, defined as in standard normal-form games;*
- S *is a strategy space, defined as in standard normal-form games;*
- $\tilde{\boldsymbol{u}} : S \to \mathbb{R}^n$ *is a vector of random variables representing players' stochastic utilities, whose realizations can be interpreted as outputs provided by the oracle* $\mathcal{O}$*.*

In particular, given a strategy profile $\boldsymbol{s} \in S$, we denote with $\tilde{\boldsymbol{u}}(\boldsymbol{s})$ the subvector of random variables corresponding to the players' utilities when the actions in $\boldsymbol{s}$ are played, with $\tilde{u}_i(\boldsymbol{s})$ being the random variable representing the utility of player $i \in N$. With an abuse of notation, we define $\boldsymbol{u} \coloneqq$

$\mathbb{E}[\tilde{\boldsymbol{u}}]$ as the expectation of $\tilde{\boldsymbol{u}}$, with $\boldsymbol{u}(\boldsymbol{s})$ and $u_i(\boldsymbol{s})$ assuming their respective meanings.

A possible example of simulation-based game is when $N = \{1, 2\}$ and, for each strategy profile $\boldsymbol{s} \in S$, the winning player (receiving a payoff of 1, instead of -1 when losing) is determined according to a Bernoulli distribution with parameter $\mu_{\boldsymbol{s}} \in [0, 1]$ (assuming that these distributions are independent among each other). Formally, this can be modeled by letting, for every $\boldsymbol{s} \in S$, $\tilde{u}_1(\boldsymbol{s}) \sim \mathrm{Be}(\mu_{\boldsymbol{s}})$, while $\tilde{u}_2(\boldsymbol{s}) = -\tilde{u}_1(\boldsymbol{s})$. Another possibility is when players' utilities are subject to a Gaussian error (representing, *e.g.*, some noise in the simulation process). In this case, we can let $\tilde{u}_i(\boldsymbol{s}) \sim \mathcal{N}(u_i(\boldsymbol{s}), \lambda)$ for every $i \in N$ and $\boldsymbol{s} \in S$, where $\boldsymbol{u} : S \to \mathbb{R}^n$ is the utility vector function of some underlying (true) normal-form game and $\lambda \in \mathbb{R}_{>0}$ is the noise standard deviation.

2.1.3 Strategy representation

In a *normal-form game* a *strategy* is a formal description of how a player acts in the game. Considering that when playing a game the players are usually allowed to randomize, we denote with Δ_{S_i} the infinite set containing all the available probability distributions of player i over his/her actions S_i, and we define a *mixed strategy* as:

Definition 3. *A* mixed strategy $x_i \in \Delta_{S_i}$*, for a player* $i \in N$*, is a probability distribution over his/her actions.*

Moreover, we denote with $x_i(s_i)$ the probability of playing $s_i \in S_i$ according to x_i. If there is a strategy $s_i \in S_i$ such that $x_i(s_i) = 1$ and, consequently, for every $s_i' \in S_i, s_i' \neq s_i, x_i(s_i') = 0$, then we say that player i is playing a *pure strategy*. It is most frequently the case that we want to consider the strategies of all the players together, so we define a *mixed strategy profile* as:

Definition 4. *A* mixed strategy profile $\boldsymbol{x} = (x_i)_{i \in N}$ *is a tuple of players' mixed strategies where* $x_i \in \Delta_{S_i}$ *for all* $i \in N$*.*

If for every player $i \in N$ there is a strategy $s_i \in S_i$ such that $x_i(s_i) = 1$ and, consequently, for every $s_i' \in S_i, s_i' \neq s_i, x_i(s_i') = 0$, then we say that $\boldsymbol{x}$ is a *pure strategy profile*. With an abuse of notation, we denote with:

$$\boldsymbol{u}(\boldsymbol{x}) \coloneqq \sum_{\boldsymbol{s} \in S} \boldsymbol{u}(\boldsymbol{s}) \prod_{j \in N} x_j(s_j), \tag{2.1}$$

i.e., players' utilities when playing a mixed strategy profile $\boldsymbol{x} = (x_i)_{i \in N}$, where $\boldsymbol{u}$ can represent either the utility vector function in a normal-form

game or the expectation of the players' stochastic utilities in a simulation-based game. Similarly, for $i \in N$, we define $u_i(\boldsymbol{x}) \coloneqq \sum_{s \in S} u_i(\boldsymbol{s}) \prod_{j \in N} x_j(s_j)$. Additionally, for every player $i \in N$ and for every action $s'_i \in S_i$, we let:

$$u_i(s'_i, \boldsymbol{x}) \coloneqq \sum_{\boldsymbol{s} \in S : s_i = s'_i} u_i(\boldsymbol{s}) \prod_{j \in N,\, j \neq i} x_j(s_j) \tag{2.2}$$

be player i's utility when playing action s'_i, given that the other players play the mixed strategies prescribed by $\boldsymbol{x}$. Given a game modeling an interaction, one of the most interesting questions is to solve it, *i.e.*, to find a joint set of mixed strategies $\boldsymbol{x}$ that are stable in some sense. To do that, one should define the conditions determining an *equilibrium*. Depending on these conditions solving the game might be trivial, i.e., zeros sum deterministic game using the maxmin equilibrium, or computationally hard [13, 11]. In what follows, we take into account a well-known definition of equilibrium: the Nash Equilibrium.

2.1.4 Nash equilibrium

In *Game Theory* a *solution concept* is a formal way to predict how a game will be played: it describes which strategies will be adopted by the players and, as a consequence, the result of the game. The most famous and commonly used solution concept is the one called the *Nash equilibrium* (NE) [25, 26].

Definition 5. *Given a normal-form game $\Gamma = (N, S, \boldsymbol{u})$, a mixed strategy profile $\boldsymbol{x} = (x_i)_{i \in N}$ is a* Nash equilibrium *if and only if, for every player $i \in N$, it is the case that $u_i(\boldsymbol{x}) \geq u_i(s'_i, \boldsymbol{x})$ for every possible deviation $s'_i \in S_i$.*

In an NE, each player must not have any incentive to change his/her current strategy, given that the others do not deviate either. An NE such that $\boldsymbol{x}$ is actually a mixed strategy profile is called *mixed Nash equilibrium*, while an NE in which $\boldsymbol{x}$ is instead a pure strategy profile is called *pure Nash equilibrium*. More formally:

Definition 6. *A strategy profile $\boldsymbol{s} \in S$ is a* pure Nash equilibrium *if and only if, for every player $i \in N$, it holds $u_i(\boldsymbol{s}) \geq u_i(\boldsymbol{s}')$ for every possible action deviation $s'_i \in S_i$, where we denote with $\boldsymbol{s}'$ the strategy profile obtained from $\boldsymbol{s} = (s_i)_{i \in N}$ by replacing its i-th component s_i with s'_i,* i.e.*, we let $\boldsymbol{s}' = (s_1, \ldots, s'_i, \ldots, s_n)$.*

Some examples of pure Nash equilibria are the strategy profiles coloured in green in Figure 2.2, while an example of mixed Nash equilibrium is the one represented by the probability distribution in Figure 2.3, where to every action is assigned a probability (reported in red) of being played by the correspondent player. In a similar way, for $\epsilon \in \mathbb{R}_+$, the concept of *$\epsilon$-Nash*

	L	C	R
T	2,1	1,2	2,3
M	1,1	2,1	1,2
B	3,3	1,1	1,2

Figure 2.2: Pure Nash equilibria in a normal-form game.

	L	C	R	
T	2,1	1,2	2,3	1/2
M	1,1	2,1	1,2	0
B	3,3	1,1	1,2	1/2
	1/3	0	2/3	

Figure 2.3: Mixed Nash equilibrium in a normal-form game.

equilibrium can be defined as:

Definition 7. *Given a normal-form game $\Gamma = (N, S, \boldsymbol{u})$, a mixed strategy profile $\boldsymbol{x} = (x_i)_{i \in N}$ is a* ϵ-Nash equilibrium *if and only if, for every player $i \in N$, it is the case that $u_i(\boldsymbol{x}) \geq u_i(s_i', \boldsymbol{x}) - \epsilon$ for every possible action deviation $s_i' \in S_i$.*

This definition is valid for both standard an simulation-based normal-form games, with the $u_i(\cdot)$ assuming their respective meanings. In the context of EGTA, the NEs (and ϵ-NEs) of a simulation-based game $\Gamma = (N, S, \tilde{\boldsymbol{u}})$ are usually defined with reference to the expectation of the players' stochastic utilities $\tilde{\boldsymbol{u}}$. Thus, the same definitions hold by recalling that, in such setting, we let $u_i(\boldsymbol{s}) = \mathbb{E}[\tilde{u}_i(\boldsymbol{s})]$ for every player $i \in N$ and strategy profile $\boldsymbol{s} \in S$.

Notice that, in standard normal-form games, computing NEs is (in principle) an easy task given that the players' utilities are available as given by the function $\boldsymbol{u}$. However, in simulation-based games, this is *not* the case, since the utilities are stochastic as specified by the multivariate random variable $\tilde{\boldsymbol{u}}$. Thus, in these settings, the goal is to learn (approximate) NEs with sufficiently high confidence by using samples drawn according to $\tilde{\boldsymbol{u}}$.

2.2 Machine Learning

Machine Learning represents a wide area of Computer Science. In particular, it is a subfield of Artificial Intelligence (AI) of problems in which parameters are affected by uncertainty. Its general goal is to understand structures from data to build models that can be used by people, more precisely, it tries to model systems able to automatically learn from experience without being explicitly programmed. A more formal definition is the one proposed by Mitchell [24] in 1997:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

The techniques employed to build up machine learning algorithms can be classified by the nature of the data available for the learning process. In particular:

- **Supervised Learning**: given *known inputs* and *known outputs*, the goal is to estimate the *unknown model* that maps those inputs to those outputs. Given a sufficient number of samples, once the model has been built, then it can be applied to new inputs to compute the estimated outputs. The typical supervised learning problems are Classification and Regression [4].

- **Unsupervised Learning**: given *unknown inputs*, the goal is to learn a more efficient representation for them. The typical unsupervised learning problems are Clustering [34] and Compression [30].

- **Reinforcement Learning**: it is close to unsupervised learning, in the sense that, in general, it has *unknown inputs*, however it is not properly about inputs and outputs, but it is based on the idea of an *agent* interacting with an *environment*, that can be identified as all that surrounds the agent and with which he/she can interact. The agent performs some *actions* that affect the environment, and the environment provides him/her some feedback, generally called *reward*, depending on each action. The objective of the agent is to learn the best behaviour to follow, called *policy*, which allows him/her to choose at every step the best action to play in order to maximize the expected cumulative reward [33].

The algorithms that we propose in this work fall into the third category, the one of Reinforcement Learning. In particular, they solve the so-called *Multi-Armed Bandit* problem.

2.2.1 Multi-Armed Bandit

The Multi-Armed Bandit (MAB) problem is a classic reinforcement learning setting and it has been firstly introduced by Lai and Robbins [19] in 1985. The general idea of such a problem is that there is an *agent* that, at each time step, has to choose an action (called *arm* in this field) among a set of available ones and, as a consequence of his/her choice, receives a reward from the *environment*. The most commonly adopted goal is to maximize the sum of cumulative rewards over a given time horizon. The agent does not know the rewards associated to the arms, and, therefore, he/she needs to try different arms (*exploration*) and then focus on the ones that seem to be the best (*exploitation*). This is the so called *exploration-exploitation trade-off* that has to be faced by Multi-Armed Bandit problems. In our work, we adopt a slightly different approach: our goal is not the one of maximizing the sum of cumulative rewards, but we try to build algorithms that, after a given number of steps, or once reached a given confidence, are able to output a set of recommended arms (that in our case represent the Nash Equilibria of the game being considered). This task is called *pure exploration problem* (Bubeck et al. [6]) and, as we can notice, in such a setting there is no exploitation-exploration trade-off to manage since the two phases are completely separated. Instead, what we try to do is to perform as few queries as possible, in the case in which we require a given confidence level for our solutions, or to reach the greater possible confidence, in the case in which we fix the number of available queries to the simulator. These concepts are better clarified in the following chapter. Before being able to write the formal definition of a Multi-Armed Bandit problem, we should underline the fact that, according to the model adopted for the arms rewards, the way in which the problem has to be faced could be completely different. In particular, the reward of an arm can be:

- *Deterministic*: we have a single value for the reward of each arm (in this case the problem could be solved trivially);
- *Stochastic*: the reward of an arm is drawn from a distribution that is stationary over time. For example if we assume that the reward of an arm follows a Bernoulli distribution with mean μ, $Be(\mu)$, then the expected reward of that arm will be equal to μ;

- *Adversarial*: an adversary chooses the reward we get from an arm at a specific time step, knowing the algorithm we are using to solve the problem.

The rewards that we consider in our algorithms, which represent the utilities of the players in the games under analysis, are of the second type, so we have stochastic rewards or, according to the formalization of the simulation-based game, we say that we have stochastic utilities. Now, we can provide the formal definition of a Multi-Armed Bandit problem as follows:

Definition 8. *A* Multi-Armed Bandit problem $(\mathcal{A}, \mathcal{R})$ *is a tuple defined as follows:*

- $\mathcal{A} \coloneqq \{a_1, a_2, \ldots, a_n\}$ *is a set of* N *possible arms;*
- $\mathcal{R}$ *is a set of unknown distributions such that, at every time step* t*,* $r_{i,t} \sim \mathcal{R}(a_i)$ *with* $\mathbb{E}[\mathcal{R}(a_i)] = R(a_i)$*.*

Solving a MAB problem calls for algorithms that at each time step $t \in \{1, \ldots, T\}$ select an arm to be played according to some criterion, receive a corresponding reward from the environment and then update their information. This means that, as a consequence of the feedback received by the environment for the action that has been played, the algorithms should be able to learn something new and to improve their knowledge gaining experience. In the case in which a classical MAB is being solved, so that there is the need to find a trade-off between exploitation and exploration, this acquired experience must be used both for trying to improve the expected cumulative reward and for proceeding with the exploration of the environment.

Notice that these techniques have been used in the past to tackle multiple application fields, like advertising allocation problems [16, 29, 27], dynamic pricing [35, 36], network routing [20], and persuasion [10].

2.2.2 Best Arm Identification in Multi-Armed Bandit

As we anticipated before, in our work we consider only pure exploration problems, so when designing the algorithms we do not care about the cumulative reward, but we only aim to reach a desired level of knowledge of the environment in order to be able to identify the best arm(s), given a maximum number of pulls T or some desired confidence level. In both cases, the general process that is performed by all the algorithms that we propose in this work is the one introduced before and summarized in Algorithm 1. We better specify in the following how we set (or not) the number of rounds T

Algorithm 1 Pure exploration problem

1: **for** $t = 1, 2, \ldots, T$ **do**
2: Select one arm a_i
3: Receive the reward from the environment $r_{i,t}$
4: Update the information
5: **end for**
6: Output a set of recommended arms

and how the arm to pull at each round is selected. Finally, notice that the way in which the information is updated depends on the specific formulation of the problem, which can be:

- *Frequentist formulation*: $R(a_1), \ldots, R(a_n)$ are *unknown parameters* and the information are given only by the observation history;
- *Bayesian formulation*: $R(a_1), \ldots, R(a_n)$ are *random variables* with prior distributions $f_1, \ldots, f_n$ and the information are given by the observation history and by the given priors.

2.2.3 PAC learning

To conclude with the theoretical background necessary for our work, we need to describe one last important framework, which is *Probably Approximately Correct (PAC) learning*. As we said, *PAC learning* is a framework, proposed by Valiant [37], used for the mathematical analysis of machine learning methods to determine under what conditions a learner L will probably output an approximately correct classifier. The learner L is given a number of instances (samples) taken from some unknown distribution and it should select a generalization function, from a given class of possible functions, such that it has, with high probability, a low generalization error. More formally, assume that we have:

- X: set of instances generated by a fixed, unknown probability distribution $\mathcal{D}$;
- H: hypothesis space;
- $\epsilon \in (0, 1)$: error tolerance;
- $\delta \in (0, 1)$: accuracy.

We define the error of an hypothesis $h \in H$, $error(h)$, as the probability that the predicted value of an instance $x \in X$ by h, $h(x)$, is different from

its actual value, y, that is:

$$error(h) := \mathbb{P}_{x \in X}\Big(h(x) \neq y \Big). \tag{2.3}$$

Given an hypothesis $h \in H$, we say that it is *approximately correct* if its error over the distribution $\mathcal{D}$ is bounded by ϵ, that is $error(h) \leq \epsilon$, and it is *probably approximately correct* if the learner L will output such hypothesis h with probability at least $1 - \delta$. Equivalently we can say that, if we divide the hypothesis space into:

- $H_0 := \{h \in H : error(h) \leq \epsilon\}$,
- $H_1 := \{h \in H : error(h) > \epsilon\}$,

then we want that the learner L does not select an hypothesis $h \in H_1$ with a probability greater than δ. More formally, we will say that the learner (or the algorithm) is δ-PAC when:

$$\mathbb{P}(h \in H_1) < \delta, \tag{2.4}$$

or equivalently:

$$\mathbb{P}(h \in H_0) \geq 1 - \delta. \tag{2.5}$$

Chapter 3

Problem formulation

The problem that we face in our work is the one of finding approximate NEs in simulation-based games. In particular, we propose different algorithms, all belonging to the same category of *dynamic querying algorithms*, that, following ad-hoc rules that we better describe in Section 3.3, allow us to solve the problem under analysis. We focus on dynamic querying algorithms satisfying the δ-PAC property, which are commonly split into two main categories [18]:

- *fixed-confidence setting*: fixing a desired accuracy $\delta \in (0, 1)$, the goal is to solve a problem keeping the total number of rounds T as small as possible;
- *fixed-budget setting*: fixing the available number of rounds T (*i.e.* the budget), the goal is to solve a problem reaching a confidence $1 - \delta$ as high as possible.

For what concerns the *fixed-confidence setting*, we aim to find both pure and mixed NEs, while, for what concern the *fixed-budget setting*, we focus only on finding pure NEs. In the following we better describe these problems.

3.1 Fixed-confidence setting

3.1.1 Learning pure Nash equilibria

As a special case of the general problem of learning equilibria in simulation-based games in the *fixed-confidence setting*, we, at first, focus on learning *pure-strategy Nash equilibria* (pure NEs) and then we move at a later time to learn *mixed-strategy Nash equilibria* (mixed NEs). In such setting, letting $E^{\mathrm{P}}(\Gamma) \subseteq S$ be the set of pure NEs of a simulation-based game $\Gamma = (N, S, \tilde{\boldsymbol{u}})$,

given $\delta \in (0,1)$, the goal is to design δ-PAC dynamic querying algorithms that output sets $\overline{E} \subseteq S$ such that:

$$\mathbb{P}\Big(\overline{E} \equiv E^{\mathrm{P}}(\Gamma)\Big) \geq 1 - \delta, \tag{3.1}$$

where, when focusing on pure NEs, $\overline{E} \equiv E^{\mathrm{P}}(\Gamma)$ is equivalent to require that the algorithm outputs all the pure NEs of the games or, if there is no pure NE, it returns the empty set $\overline{E} = \varnothing$. This must be done while minimizing the number of samples, *i.e.*, minimizing the number of needed queries to the simulator $\mathcal{O}$.

3.1.2 Learning mixed Nash equilbria

In the more general case of learning *mixed-strategy Nash equilibria* in simulation based games in the *fixed-confidence setting* we instead adopt a different approach, as we focus on finding only a single mixed NE. This decision is due to the fact that given a generic game $\Gamma = (N, S, \boldsymbol{u})$, denoting with $E(\Gamma) \subseteq S$ the set of its mixed NEs, in principle $|E|$ can be infinite, making impossible for any algorithm to find all its elements and then stop. On the other hand, thanks to the famous theorem of Nash [26], we know that every game admits at least one mixed NE, that is we are guaranteed that $E(\Gamma) \neq \varnothing$. For these reasons, with an abuse of notation, given a simulation-based game $\Gamma = (N, S, \tilde{\boldsymbol{u}})$, its set of mixed NEs $E(\Gamma) \subseteq S$ and $\delta \in (0,1)$, the goal is to design δ-PAC dynamic querying algorithms that output a mixed strategy profile $\boldsymbol{x}$ such that:

$$\mathbb{P}\Big(\boldsymbol{x} \in E(\Gamma)\Big) \geq 1 - \delta. \tag{3.2}$$

3.2 Fixed-budget setting

3.2.1 Learning pure Nash equilibria

In the last case that we treat in this work, that is the one of learning equilibria in simulation-based games in the *fixed-budget setting*, we use an approach analogous to the one described for the *fixed-confidence setting* in Section 3.1.1. However, since we are in a *fixed-budget setting*, we do not fix the desired δ but we fix the available number of queries to the simulator $\mathcal{O}$, which represent our budget, and we try to reach a confidence $1 - \delta$ as big as possible with which we can satisfy the condition of Equation (3.1).

3.3 Dynamic querying algorithms for simulation-based games

The problems that we described and that we want to solve call for algorithms that iteratively sample the players' stochastic utilities, and, then, output some strategy profiles which represent good candidates for being (approximate) NEs. In practice, we can interpret these algorithms as dynamically querying a game simulator to get estimates of the players' utilities. In the following, given a simulation-based game $\Gamma = (N, S, \tilde{\boldsymbol{u}})$ and $\epsilon \in \mathbb{R}^+$, we let $E_\epsilon(\Gamma)$ be the set of mixed strategy profiles $\boldsymbol{x} = (x_i)_{i \in N}$ that are ϵ-NEs of Γ. Moreover, we let $E(\Gamma) \coloneqq E_0(\Gamma)$ be the set of NEs of Γ.

Following the approach of Garivier et al. [14], we focus on *dynamic querying algorithms* with the following structure. During each round t, the algorithm selects a strategy profile $\boldsymbol{s}^t \in S$ and draws a sample $\boldsymbol{u}^t \in \mathbb{R}^n$ from the random variables in $\tilde{\boldsymbol{u}}(\boldsymbol{s}^t)$. We denote with u_i^t the i-th component of $\boldsymbol{u}^t$ which represents the sampled player i's utility value. Then, a dynamic querying algorithm is defined by the following components:

- a *querying rule* that indicates which strategy profile $\boldsymbol{s}^t \in S$ is chosen during each round t;
- a *stopping rule* that determines the round T after which the algorithm terminates its execution;
- a *guessing rule* that outputs a finite set of (possibly mixed) candidate strategy profiles $\overline{E}$ upon termination.

Given a desired approximation level $\epsilon \in \mathbb{R}^+$, the objective of a dynamic querying algorithm is to produce a set $\overline{E}$ that represents the set $E_\epsilon(\Gamma)$ of ϵ-NEs of the game with high accuracy, using as few samples (rounds) as possible. Formally, given $\delta \in (0, 1)$, the goal is to design algorithms that are δ-PAC, *i.e.*, they satisfy:

$$\mathbb{P}\Big(\overline{E} \equiv E_\epsilon(\Gamma) \Big) \geq 1 - \ \delta, \tag{3.3}$$

where by $\equiv$ we mean that the two sets $\overline{E}$ and $E_\epsilon(\Gamma)$ contain the same strategy profiles. Here, we make the additional assumption that the set $E_\epsilon(\Gamma)$ of ϵ-NEs of the game is finite. Whenever we set $\epsilon = 0$ (*i.e.*, we look for exact NEs of the game), the assumption is w.l.o.g. provided that the expected players' payoffs defined by $\boldsymbol{u} = \mathbb{E}[\tilde{\boldsymbol{u}}]$ are *generic* (see [41] for a formal definition). This is indeed the case in most of the real-world settings that are commonly modeled through simulation-based games. However, whenever $\epsilon > 0$ or the

payoffs are *not* generic, the set $E_\epsilon(\Gamma)$ may contain infinite mixed strategy profiles (*e.g.*, for $\epsilon = 1$, all the infinite mixed strategy profiles of a game with utilities in the range $[0, 1]$ are ϵ-NEs, whereas the popular *ultimatum game*, whose payoffs are not generic, admits an infinite number of exact NEs). In these cases, we relax the requirement in Equation (3.3) and only ask for $\overline{E} \subseteq E_\epsilon(\Gamma)$ or restrict the attention to dynamic querying algorithms that output sets $\overline{E}$ comprised of a single (possibly mixed) strategy profile. Recall that, while a game always admits an NE in mixed strategies [26], a pure NE may *not* exist.

Chapter 4

State of the art

In this chapter we review the current state of the art on empirical game-theoretic analysis (EGTA), the field that studies ways to learn equilibria in simulation-based games. Most of the available EGTA methods works by first estimating someway the utilities of the game in question, in order to build an empirical game, and then computing approximate Nash equilibria of that game. We instead use *best arm identification* techniques, so that our search does not require a global exploration, but it relies on specific selection rules in order to reduce the number of needed queries to the simulator $\mathcal{O}$. In the following, we first do a brief review of the works that have been proposed on the topic of simulation-based games and empirical game-theoretic analysis from the beginning till now, then we analyze more in depth the state-of-the-art works that are more related to ours and from which we take more inspiration.

4.1 General background

The first computational studies on simulation-based games have been carried out by Vorobeychik et al. [43], who addressed the problem of solving a game by generalizing from some data a complete payoff-function representation as a standard regression problem. Specifically, they focused their analysis on learning *n-player general-sum games* and the approach that they adopted could be summarized in the three steps that are reported in Algorithm 2. Their work has been also extended by Gatti and Restelli [15] to manage sequential games. This type of approach deeply differs from ours since they aimed to firstly estimate all the utilities of the game in question with a certain level of precision, and then, only in a second moment, to compute its Nash equilibria. Our work, instead, aims to reduce the number of samples

Algorithm 2 EGTA as regression problem

1: Sample a few strategy profiles
2: Learn functions that generalize utilities to unseen data from the obtained observations employing *regression techinques*
3: Compute Nash equilibria of the learned game

needed to find the result by exploiting best arm identification techniques and providing ad hoc querying rules, avoiding this way the need to learn precisely the entire game. Finally, notice that they employed standard regression techniques to generalize the utilities of the game from the available observations, but other methods can be used, as it has been done by other recent studies. In particular, there are two works [45, 31] that adopted an approach similar to the one proposed by Vorobeychik et al. [43], but employing different regression techniques. More precisely, Wiedenbeck et al. [45] tried to use both neural networks and GP as regression method, but then focused on the latter, since they experimented that it leads them better performance compared to other techniques, while, on the other hand, Sokota et al. [31] selected neural networks as regression technique and also provided a procedure for iteratively refine the learned model with new data by selecting which arm to sample so as to focus the search process on the neighborhood of candidate equilibrium points. Notice that this last introduced way to guide the querying process and to intelligently select which arms to sample is similar to what we aim to do in our work, however, our search is not strictly related with the one performed by these two works because both of them focused their analysis on games with *many symmetric players*, with the purpose of trying to exploit the advantages of symmetries in order to be able to handle larger games, while we always consider *n-player general-sum games*.

Completely different approaches, that we mention here for completeness but that we do not analyze in depth since they are too far from ours, are the ones proposed by Walsh et al. [44], whose sampling process was based on heuristics derived from information-theoretic principles, and by Burch et al. [7], whose sampling process was based on variance reduction techniques.

The last type of approach over which we want to focus our attention, which is the one to which our algorithms belong, is the one of those that rely on the PAC framework to obtain and prove theoretical guarantees [14, 38, 39, 47, 21]. More specifically, these works could be seen as they were divided into two sub-groups: those that, similarly to the one described in Algorithm 2, but without the employing of regression techniques, start

with a *global exploration*, learning the entire game with respect to a desired precision ϵ and accuracy δ, and then compute the ϵ-NEs of the game learned this way [38, 39] and those that focus on *dynamic querying algorithms*, treating the problem as a variant of the task of best-arm identification in multi-armed bandits field, focusing on identifying the arms that are useful to establish which strategy profile is an equilibrium, with respect to the same desired precision ϵ and accuracy δ [47, 21]. In particular, Viqueira et al. [38, 39] proposed two algorithms, one that pulls all the strategy profiles in a game the same given number of times and then outputs the estimated game along with its estimated error and accuracy, and then a second one that iteratively apply the first removing at every repetition some sets of sub-optimal arms. Since this approach is not the one that we adopt we do not describe their algorithms more in depth. Instead, in the following we better analyze the algorithms belonging to the other sub-group. Finally, notice that all the algorithms that we introduce here focus on learning *maxmin strategies* in *two-player zero-sum games* while in our work we focus on learning NEs in *n-player general-sum games*.

4.2 EGTA with best arm identification techniques and PAC guarantees

In the following, as in the rest of our work, we analyze two types of algorithms: the ones in fixed-confidence setting and the ones in fixed-budget setting.

4.2.1 Fixed-confidence setting

Putting ourselves in the fixed-confidence setting, we would like to provide here the main concepts and ideas behind the algorithms proposed by Garivier et al. [14], Marchesi et al. [21] and Zhou et al. [47]. They are all based on the idea of keeping track of both an upper and a lower bounds on the value of each arm, where in our discussion an arm always corresponds to a strategy profile and its value always corresponds to the utilities of the players at that strategy profile, and to search for the arm that can be considered the best given a required confidence δ and a maximum error ϵ. Moreover, notice that, recalling that these works that are being presented deal only with *two-player zero-sum games*, for every strategy profile it is enough to keep only track of the utility of player 1, because, in case of need, the utility of player 2 could always be obtained by simply shifting the sign of the one of player 1. Now, given two strategy profiles $\boldsymbol{s}, \boldsymbol{s'} \in S$, let $[L^t(\boldsymbol{s}), U^t(\boldsymbol{s})]$

and $[L^t(\boldsymbol{s}'), U^t(\boldsymbol{s}')]$ be their confidence intervals over the utility of player 1 computed at a certain time t with a confidence of δ. The strategy profile $\boldsymbol{s}$ is probably approximately better than the strategy profile $\boldsymbol{s}'$ if:

$$L^t(\boldsymbol{s}) > U^t(\boldsymbol{s}') - \epsilon. \tag{4.1}$$

In the example shown in Figure 4.1, assuming that arm $\boldsymbol{s}$ is represented by the green line and arm $\boldsymbol{s}'$ is represented by the red line, where the central points represent their expected values, it can be seen that the lower bound of $\boldsymbol{s}$ beats, up to an error of less than ϵ, the upper bound of $\boldsymbol{s}'$, so in a case like this $\boldsymbol{s}$ is considered to be better than $\boldsymbol{s}'$ with a confidence of δ. Finally,

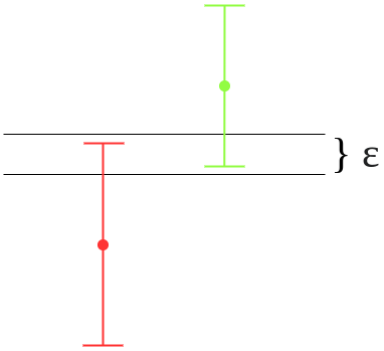

Figure 4.1: Identification of the best arm between two.

let us notice that, since we are dealing with *zero-sum games*, whenever we consider the situation in which player 1 selects an action, we take for granted that player 2, since he/she is a minimizer, selects the response with the lower utility among the available ones, uniquely identifying this way a single strategy profile as a consequence of a choice of an action by player 1. In a sense we are looking for a different kind of equilibrium, usually called Stackelberg or leader-follower equilibrium [40, 9, 12, 8, 23, 22], which is commonly used in security games models [5, 3].

Building on this concepts, Garivier et al. [14] proposed an algorithm, called M-LUCB, that aims to identify a strategy profile $\boldsymbol{s} \in S$, $\boldsymbol{s} = (s_1, s_2)$ whose lower bound beats, up to an error of at most ϵ, the upper bound of at least one arm $\boldsymbol{s}' \in S$, $\boldsymbol{s}' = (s_1', *)$ in each other action that could be selected by player 1, $s_1' \in S_1, s_1' \neq s_1$. In order to do this, they proposed an ad hoc querying rule, specifically built to advance towards this termination condition. More precisely, they proposed to pull every time two pairs of arms: one representing the *maxmin equilibrium* computed on the game built using the estimates collected until time t, and the second one representing the *maxmin equilibrium* computed on the game built using the upper bounds

on the utilities at time t, with the only limitation that player 1 can not choose again the action selected for the first pull. The idea is that these two strategy profiles should represent the first and the second more probable candidates to be the maxmim equilibrium, and so the algorithm stops when the first one is probably approximately better than the second one according to the condition explained before. In the analysis of their algorithm they used Hoeffding-type confidence bounds, they proved under which condition their algorithm is δ-PAC and provided a bound on the number of rounds T_δ needed to the reach the termination condition. What we aim to do in the first algorithm that we propose in our work, PNE-LUCB, is to extend this kind of approach to *n-player general-sum games*.

Marchesi et al. [21] proposed an algorithm, called M-GP-LUCB, based on the one of Garivier et al. [14]. The two algorithms share the same structure, however the one proposed by Marchesi et al. [21] uses confidence bounds relying on GP assumptionsand, as a consequence, they provided different proofs to show that it maintains its δ-PAC properties.[1]

Finally, Zhou et al. [47], basing on the same concepts, proposed a slightly different approach. Their algorithm, called LUCB-G, is inspired by the works of Kalyanakrishnan et al. [17] and Auer et al. [2] and the idea is to decompose the problem of finding a *maxmin equilibrium* in a *two-player zero-sum game* into more and smaller best arm identification problems. More specifically, in their algorithm they proceed by trying to find, for each player, the best response he/she has to every action of the other player. Then, when they find a strategy profile $\boldsymbol{s} \in S$, $\boldsymbol{s} = (s_1, s_2)$ such that s_1 has been selected to be the best response to s_2 and s_2 has been selected to be the best response to s_1, then they assert that $\boldsymbol{s}$ is the *maxmin equilibrium* that was being searched. Of course, they proposed an ad hoc querying strategy built to reach this termination condition, however we do not report it here since we deeply describe it in the context of the second algorithm we propose, LUCB-NG, that is an extension of their algorithm to handle *n-player general-sum games*. Finally, lets underline that, as Garivier et al. [14], they used Hoeffding-type confidence bounds, they proved under which condition their algorithm is δ-PAC and provided a bound on the number of rounds T_δ needed to the reach the termination condition.

[1] See the work by Williams and Rasmussen [46] for more details about GP and the ones by [32, 28] for their use in the MAB setting.

4.2.2 Fixed-budget setting

Putting ourselves in the fixed-budget setting, we would like to underline here the reason why different algorithms with respect to the ones proposed for the fixed-confidence setting must be elaborated and to briefly illustrate the algorithms from which we have taken more inspiration in building the algorithm in fixed-budget setting that we present in this work. The reason why a different approach must be considered when dealing with fixed-budget setting is that, since, as the name says, the *budget* is fixed, we can not rely as before on conditions related to relationship between bounds because, regardless of the chosen bound type, they shrink as the number of pulls increases, but the number of pulls needed to obtain the desired results could not fit into the budget. For this reason, we do not see any bounds when dealing with algorithms in fixed-budget, but everything relies on the estimated values computed with the samples obtained until every considered time step t.

We would like to mention here the algorithm proposed by Audibert et al. [1], named Successive Reject, that, informally, works by dividing the available budget into $K - 1$ phases, where K is the number of arms in the problem, and then to dismiss, at the end of each phase, the arm with the lowest empirical mean. During each phase they pull equally often all the arms that have not been dismissed a number of times computed ad hoc to exhaust but not to overcome the budget. The arm that remains at the end of the last phase is the one that is recommended. Finally, they also provided a bound on the error that could be done by their algorithm.

Marchesi et al. [21] in their work proposed an algorithm, called GP-SE, inspired by the one of Audibert et al. [1], but with an elimination rule that is suitably defined for the problem of identifying *maxmin equilibirum* in *two-player zero-sum game*. Informally, at the end of each phase, they eliminate the arm, that in their case corresponds to a strategy profile, that has the lowest chance of being a maxmin equilibrium. In our algorithm we propose this same approach but setting up the arms and designing the elimination rule to find the Nash equilibria of *n-player general-sum games*.

Chapter 5

Algorithms for identifying Nash Equilibria

We propose two algorithms to learn pure NEs in simulation-based games in the fixed-confidence setting, one algorithm to learn one mixed NE in the fixed-confidence setting and one algorithm to learn all the pure NEs in the fixed-budget setting. Since all the algorithms in the fixed-confidence setting are based on the same idea of using upper and lower confidence bounds on the players' utilities so as to guide the querying process, we provide here some general concepts and useful notation that are valid for all of them, then we describe each one of them specifically.

For every strategy profile $\boldsymbol{s} \in S$ and player $i \in N$, we keep track of a confidence interval $[L_i^t(\boldsymbol{s}), U_i^t(\boldsymbol{s})]$ on $u_i(\boldsymbol{s}) = \mathbb{E}\left[\tilde{u}_i(\boldsymbol{s})\right]$ built using the samples $\tilde{\boldsymbol{u}}^t$ drawn up to round t. Moreover, using the same samples, we also maintain an estimate $\hat{u}_i^t(\boldsymbol{s})$ for the value of $u_i(\boldsymbol{s})$. The lower and upper bounds determining the confidence intervals, respectively, $L_i^t(\boldsymbol{s})$ and $U_i^t(\boldsymbol{s})$, as well as the estimates $\hat{u}_i^t(\boldsymbol{s})$, can be defined in different ways depending on the specific assumptions on the distributions of the random variables $\tilde{u}_i(\boldsymbol{s})$ (see the following chapter for more details).

5.1 PNE-LUCB

The first algorithm that we propose is inspired by the M-LUCB algorithm proposed by Garivier et al. [14], which is based on the idea of using upper and lower confidence bounds on the players' utilities so as to guide the querying process. In particular, the difference between their algorithm and ours is that they focus their search on *two-player zero-sum games*, while we extend their approach to *n-player general-sum games*. As the letter M in the name

of their algorithm stood for the fact that the algorithm was designed to find a Maxmin equilibrium, we call our algorithm PNE-LUCB, to indicate that its goal is to find Pure Nash Equilibria.

As we anticipated in Chapter 3, our algorithms belong to the category of *dynamic querying algorithms*: they work on rounds, and every round is guided by the *querying rule*, the *guessing rule*, and the *stopping rule*, which we describe in the following. Moreover, at the end of each round, our algorithm also checks if there are strategy profiles it can categorize, with a confidence of at least $1 - \delta$ and with an error of at most ϵ, as NEs or not-NEs. In particular, a strategy profile $\boldsymbol{s} \in S$ is considered to be an NE if, at some time t, for every player $i \in N$, and action deviation $s'_i \in S_i$:

$$L_i^t(\boldsymbol{s}) > U_i^t(s'_i, \boldsymbol{s}_{-i}) - \epsilon. \tag{5.1}$$

On the other hand, a strategy profile is considered not to be an NE if for at least a player $i \in N$, there is an action deviation $s'_i \in S_i$ such that:

$$L_i^t(s'_i, \boldsymbol{s}_{-i}) > U_i^t(\boldsymbol{s}) - \epsilon. \tag{5.2}$$

During its execution, the PNE-LUCB also keeps track of two sets, $\bar{E}$ and S^t, of strategy profiles in S:

- $\bar{E}$ is the set of strategy profiles the algorithm is sure that they are NEs according to the condition defined by Equation (5.1). This is also the set that is output by the algorithm upon termination;
- S^t is the set of strategy profiles the algorithm, at time t, has not yet established whether they are NEs or not, that is they have not yet fulfilled neither the condition defined by Equation (5.1) nor the one defined by Equation (5.2).

We propose two variants of our algorithm, one in which the goal is to find only one Nash equilibrium and then one in which the goal is to find all the Nash equilibria. The only difference between these two variants is in an additional possibility in the stopping rule in the case in which we are looking for a single NE. The algorithm, in both variants, can handle the situation in which there are no Nash equilibria.

The main components of the PNE-LUCB algorithm are defined as follows.

- *Querying rule*: At the end of every round t such that there is no strategy profile left to be queried, the algorithm selects the strategy profiles to be queried in the next rounds as follows. Let $E^t \subseteq S$ be the

set of pure NEs in the normal-form game whose utility vector function is defined by the estimates $\hat{u}_i^t(\boldsymbol{s})$ computed up to round t. In the case in which E^t is not empty, the algorithm checks whether there are strategy profiles $\boldsymbol{s} \in E^t$ that are already in $\bar{E}$ and, if it is the case, it removes them from E^t since it is already sure that they are NEs and so it does not want to work on them anymore. In the case in which, before or after this check, E^t is empty, the algorithm takes a strategy profile $\boldsymbol{s} \in S^t$ at random and it considers it as the only element of E^t. Let us assume that the equilibria in E^t are ordered according to some (arbitrary) criterion, and that each of them is assigned a label in $\{0, \dots, |E^t|\}$ according to its position in the ordering. Moreover, for each label $k \in \{0, \dots, |E^t|\}$, let $t_k \coloneqq t + 1 + k(n+1)$. Following the ordering, the algorithm takes each strategy profile in E^t in turn, and, for each of them, it queries a set of $n+1$ strategy profiles, defined as follows. Suppose that $\boldsymbol{s} \in E^t$ is the k-th strategy profile in E^t, according to the ordering. Then, the algorithm queries the following strategy profiles during the rounds going from t_k to $t_{k+1} - 1$:

$$\boldsymbol{s}^{t_k} \coloneqq \boldsymbol{s} \tag{5.3a}$$

$$\boldsymbol{s}^{t_k+i} \coloneqq \Big(\underset{s_i' \in S_i : s_i' \neq s_i}{\arg\max}\ U_i^t(s_i', \boldsymbol{s}_{-i}), \boldsymbol{s}_{-i} \Big) \quad \text{for every} \quad i \in N, \tag{5.3b}$$

where $\boldsymbol{s}^{t_k+i} = (s_i', \boldsymbol{s}_{-i})$ is the strategy profile obtained by replacing s_i, the strategy of player i in $\boldsymbol{s}^{t_k}$, with s_i', assuming that all the other players continue to play the joint strategy profile $\boldsymbol{s}_{-i}$ prescribed by $\boldsymbol{s}^{t_k}$. So, Equation (5.3a) prescribes to query exactly the NE that is being taken into consideration, that is the k-th, while Equation (5.3b) looks for the best response, according to the values of the upper bounds, to the considered NE for every player $i \in N$. Notice that, at round t, the algorithm decides which strategy profiles are to be queried in the following $(n+1)|E^t|$ rounds.

- *Stopping rule*: The algorithm terminates at the end of a round t if one of the following conditions is satisfied:
  - (*Looking for a single NE*): There is at least one element in $\bar{E}$, that is $|\bar{E}| > 0$;
  - S^t is empty, that is $|S^t| = 0$, which means that every strategy profile $\boldsymbol{s} \in S$ of the game has fulfilled either the condition defined by Equation (5.1) or the one defined by Equation (5.2) and so the algorithm has established whether it is, or not, a NE.

- *Guessing rule*: Upon termination the algorithm outputs the set $\bar{E}$. Notice that it could contain zero, one or multiple equilibria.

The idea behind the selection process adopted by the algorithm is to advance towards the termination condition. In order to move towards the termination condition, the algorithm needs to refine the lower bounds $L_i^t(\boldsymbol{s})$ (*i.e.*, it must increase them) and the upper bounds $U_i^t(\boldsymbol{s}')$ (*i.e.*, it must decrease them). In order to do so, the relevant strategy profiles must be queried so as to shrink their confidence intervals. These are exactly the strategy profiles selected by the querying rule defined by Equation (5.3).

5.1.1 Pseudocode

In the following, we provide the pseudocode of the algorithm we have just described in order to better clarify its functioning. Notice that we use some procedures in order to not make the reading too heavy. In particular:

- pullStrategyProfile($\boldsymbol{s}$): queries the oracle $\mathcal{O}$ at the strategy profile $\boldsymbol{s}$, and then updates the estimated utilities using the new sample according to the properties of the distribution that is being considered;
- pullAllStrategyProfiles(S): for every strategy profile $\boldsymbol{s} \in S$ calls the procedure pullStrategyProfile($\boldsymbol{s}$);
- computePureNE($\bar{\Gamma}$): computes the pure Nash equilibria in the estimated game $\bar{\Gamma}$, removes from them, if there are, the NEs already contained in $\bar{E}$ and, if there are no strategy profiles left, takes a strategy profile at random among the ones in S^t;
- checkNE($\boldsymbol{s}$): checks the condition of Equation (5.1) for the strategy profile $\boldsymbol{s}$ and returns True if it is satisfied, False otherwise;
- checkNotNE($\boldsymbol{s}$): checks the condition of Equation (5.2) for the strategy profile $\boldsymbol{s}$ and returns True if it is satisfied, False otherwise.

Algorithm 3 PNE-LUCB

Input: boolean allNE, True if looking for all the Nash
Output: $\bar{E}$

Initialize $t \leftarrow 0$, $\bar{E} \leftarrow \emptyset$, $S^t \leftarrow S$
pullAllStrategyProfiles(S^t)
while (allNE **or** $\bar{E} == \emptyset$) **and** $S^t \neq \emptyset$ **do** ▷ Stopping rules
    $E^t \leftarrow$ computePureNE($\bar{\Gamma}$)
    for $\boldsymbol{s} \in E^t$ **do**
        pullStrategyProfile($\boldsymbol{s}$)
        for $i \in N$ **do**
            $\boldsymbol{s}^i \leftarrow \left(\arg\max_{s_i' \in S_i : s_i' \neq s_i} U_i^t(s_i', \boldsymbol{s}_{-i}), \boldsymbol{s}_{-i} \right)$ ▷ Equation (5.3b)
            pullStrategyProfile($\boldsymbol{s}^i$)
        end for
    end for
    $t \leftarrow t + (n+1) \cdot |E^t|$
    $\bar{E} \leftarrow \bar{E} + \{\boldsymbol{s} \in E^t | \text{checkNE}(\boldsymbol{s})\}$ ▷ Equation (5.1)
    $S^t \leftarrow S^t - \bar{E}$
    $S^t \leftarrow S^t - \{\boldsymbol{s} \in S^t | \text{checkNotNE}(\boldsymbol{s})\}$ ▷ Equation (5.2)
end while
return $\bar{E}$

5.1.2 Example

Assume that we are running PNE-LUCB to learn a game and that, at a certain time step t, the utilities of the estimated game, denoted with $\hat{\boldsymbol{u}}^t(\boldsymbol{s})$, are the ones reported in Figure 5.1, while the lower and upper bounds are,

	L	C	R
T	4,5	0,1	0,2
M	0,0	1,1	1,2
B	1,1	0,2	0,1

Figure 5.1: Estimated game.

for simplicity, for every strategy profile $\boldsymbol{s} \in S$, $\hat{\boldsymbol{u}}^t(\boldsymbol{s}) \pm 2$. We can represent them as two games, whose utilities are defined by the values of the lower an upper bounds for each strategy profile $\boldsymbol{s} \in S$, as they are reported in Figure 5.2 and Figure 5.3 respectively. Firstly, our algorithm computes

	L	C	R
T	2,3	-2,-1	-2,0
M	-2,-2	-1,-1	-1,0
B	-1,-1	-2,0	-2,-1

Figure 5.2: Lower bounds game.

	L	C	R
T	6,7	2,3	2,4
M	2,2	3,3	3,4
B	3,3	2,4	2,3

Figure 5.3: Upper bounds game.

the equilibria in the estimated game, so in our case it indicates the strategy profiles (T, L) and (M, R). Considering the first NE, (T, L) (colored in green in Figure 5.4):

- The best deviation for player 1, considering the game defined by the upper bounds, is (B, L) (colored in light blue in Figure 5.4);
- The best deviation for player 2, considering the game defined by the upper bounds, is (T, R) (colored in yellow in Figure 5.4);

	L	C	R
T	6,7	2,3	2,4
M	2,2	3,3	3,4
B	3,3	2,4	2,3

Figure 5.4: Upper bounds game.

so the algorithm pulls the strategy profiles (T, L), (B, L) and (T, R). Then, considering the second NE, (M, R) (colored in green in Figure 5.5):

- The best deviation for player 1, considering the game defined by the upper bounds, is one at random among (T, R) and (B, R), since they are equal from his/her point of view. Here we take (B, R) (colored in light blue in Figure 5.5);
- The best deviation for player 2, considering the game defined by the upper bounds, is (M, C) (colored in yellow in Figure 5.5);

	L	C	R
T	6,7	2,3	2,4
M	2,2	3,3	3,4
B	3,3	2,4	2,3

Figure 5.5: Upper bounds game.

so the algorithm pulls the strategy profiles (M, R), (B, R) and (M, C). Notice that, as we said in the description of the algorithm, at time t the algorithm decided which strategy profiles were to be queried in the following $(n + 1)|E^t|$ rounds, that in our case corresponds to $(2 + 1) \cdot 2 = 6$ strategy profiles pulled. At this point, assuming for simplicity that the estimated game does not change and that only the lower and upper bounds of the pulled strategy profile change, and that they all change in the same way, that is they become, for every strategy profile $\boldsymbol{s} \in S$ that has benen pulled, $\hat{\boldsymbol{u}}^t(\boldsymbol{s}) \pm 1$, as reported in Figure 5.6 and Figure 5.7, we need to check the

	L	C	R
T	3,4	-2,-1	-1,1
M	-2,-2	0,0	0,1
B	0,0	-2,0	-1,0

Figure 5.6: New lower bounds game.

	L	C	R
T	5,6	2,3	1,3
M	2,2	2,2	2,3
B	2,2	2,4	1,2

Figure 5.7: New upper bounds game.

condition of Equation (5.1) for (T, L) and (M, R) and the condition of Equation (5.2) for all the strategy profiles. Here, for the sake of brevity, we show only the checking of the condition of Equation (5.1) since it is the most interesting part. Considering the first NE, (T, L), we see that the condition is met, since its lower bounds (colored in green in Figure 5.8) are greater then the upper bounds of every possible deviation of player 1 (colored in light blue in Figure 5.9) and of player 2 (colored in yellow in Figure 5.9). So, we can add (T, L) to the set of the NEs of the game $\bar{E}$, remove it from the set of strategy profiles to be taken into consideration S^t, and do not analyze it

	L	C	R
T	3,4	-2,-1	-1,1
M	-2,-2	0,0	0,1
B	0,0	-2,0	-1,0

Figure 5.8: New lower bounds game.

	L	C	R
T	5,6	2,3	1,3
M	2,2	2,2	2,3
B	2,2	2,4	1,2

Figure 5.9: New upper bounds game.

anymore. On the other hand, considering the second NE, (M, R), we see that the condition is not met, since its lower bounds (colored in green in Figure 5.10) are lesser then the upper bounds of every possible deviation of player 1 (colored in light blue in Figure 5.11) and of player 2 (colored in yellow in Figure 5.11). So, we can not conclude anything.

	L	C	R
T	3,4	-2,-1	-1,1
M	-2,-2	0,0	0,1
B	0,0	-2,0	-1,0

Figure 5.10: New lower bounds game.

	L	C	R
T	5,6	2,3	1,3
M	2,2	2,2	2,3
B	2,2	2,4	1,2

Figure 5.11: New upper bounds game.

5.2 Theoretical analysis

Theorem 1. *Assume to have a two-player zero-sum simulation-based game where each reward $u_i(\boldsymbol{s})$ is distributed as independent Gaussian distributions for each strategy profile with known variance σ^2. Assume that b_t is a generic non-decreasing exploration term, λ is the variance of the process, and $H^*(u) = \max_i \sum_{\boldsymbol{s}\in S} c_i(\boldsymbol{s})$, where $c_i(\boldsymbol{s}) = \frac{1}{\max\{(u_i(\boldsymbol{s})-u_i(s_i^*,s_{-i})^2,(c_i-u_i(s_i^*,s_{-i})^2\}}$ and $c_i = \frac{u_i(s_i^*,s_{-i}^*)+u_i(s_i^{**},s_{-i}^{**})}{2}$. The algorithm PNE-LUCB with $\epsilon = 0$ requires at least:*

$$T_\delta \leq \inf\left\{t \in \mathbb{N} : 12 \max_i H^*(u) b_t \lambda - \frac{3\lambda|S_1||S_2|}{\sigma^2} \leq t\right\} \tag{5.4}$$

samples before the termination.

Proof. In what follows, we provide a sketch of the proof, highlighting the points in common and the differences w.r.t. previous works on similar topic. In this specific setting we are sure that a single pure NE is present, and it coincides with the max-min/min-max equilibrium. Notice that the guarantees provided by Marchesi et al. [21] are not sufficient to provide a given

confidence that the equilibrium is a Nash. Indeed, the outcome of the M-GP-LUCB algorithm provides as output only the action of the first player, while an NE is composed by a strategy for each player.

The existence of the max-min, even in the empirical form of the game is ensured, therefore the PNE-LUCB selects a single candidate in E^t. Let us assume that at each time instant it is unique (no ties in terms of utility among arms are present). We would like to compute the value of τ_δ, *i.e.*, a time instant for which the confidence of returning an NE is at least $1-\delta$.

Assume the event $\mathcal{E}_t = \cup_{\boldsymbol{s}\in S} u_i(\boldsymbol{s}) \in [L_i^t(\boldsymbol{s}), U_i^t(\boldsymbol{s})]$ and $\mathcal{E} = \cup_{t\in 3\mathbb{N}}\mathcal{E}_t$. The suggested strategy satisfies $\hat{\boldsymbol{s}}$: $u_i(\hat{\boldsymbol{s}}) \geq L_i^t(\hat{\boldsymbol{s}}) \geq \min_{s_i'\in S_i} U_i^t(s_i'; s_{-i}) > \min_{s_i'\in S_i} u_i(s_i'; s_{-i})$ and this is guaranteed for each player $i \in N$, therefore deviations from the suggested strategy only decreases the expected utility of the player. This ensures that if the algorithm terminates and $\mathcal{E}$ holds, the suggested is guaranteed to be a NE.

Assume that the strategy $\boldsymbol{s}^{NE} = (s_i^{NE}, s_{-i}^{NE})$ is a NE, define c_i as a value in $\left[\max_{s_i'\in S_i, s_i'\neq s_i^{NE}} u_i(s_i', s_{-i}^{NE}), u_i(s^{NE})\right]$ and $t \in 3\mathbb{N}$. If $\mathcal{E}_t$ holds and $\tau_\delta > t$, then the bounds of one of the points selected by the algorithm for turn t contains c_i. Proving this is a straightforward extension of what is provided in Lemma 10 by Garivier et al. [14], where for each player we consider the point in E^t and the one determined selecting the upper bound corresponding to the best deviation from E^t. Notice that differently from [14] we have a value of c_i for each player, since we have a second best action per player.

The last part of the proof follows what has been presented in Theorem 4.1 in [21], where the Sherman-Morrison inversion formula is used to provide a bound on the variance on the utility, and, therefore, a bound in the number of samples used for an arm, given that c_i is still inside the confidence bounds. Formally, we have:

$$N_{t,i}(\boldsymbol{s}) \leq 4b_t\lambda \frac{1}{\max\{(u_i(\boldsymbol{s}) - u_i(s_i^*, s_{-i})^2, (c_i - u_i(s_i^*, s_{-i})^2\}} - \frac{\lambda}{\sigma^2}, \quad (5.5)$$

where s_i^* is the best response to s_{-i} of the i-th player, λ is the variance of the process and σ^2 is the known variance of the utility function. Choosing:

$$c_i = \frac{u_i(s_i^*, s_{-i}^*) + u_i(s_i^{**}, s_{-i}^{**})}{2}, \quad (5.6)$$

where s_i^* is the best action for player i, s_{-i}^* is the best response of the other player to s_i^*, s_i^{**} is the second best action for player i and s_{-i}^{**} is the best response of the other player to s_i^{**}, we have the final number of pulls, given that $b_t = 2\log(\frac{|S_1||S_2|S^2t^2}{6\delta})$ and summing over all the arms.

Defining $H^*(u) = \max_i \sum_{\boldsymbol{s} \in S} c_i(\boldsymbol{s})$, where:

$$c_i(\boldsymbol{s}) = \frac{1}{\max\{(u_i(\boldsymbol{s}) - u_i(s_i^*, s_{-i})^2, (\frac{u_i(s_i^*, s_{-i}^*) + u_i(s_i^{**}, s_{-i}^{**})}{2} - u_i(s_i^*, s_{-i})^2\}}, \tag{5.7}$$

and following the proof of Theorem 4.1 in [21] we have:

$$\begin{aligned}
&min\{T_\delta, t\} \\
&\leq \sum_{h \in 3\mathbb{N}, h<t} \mathbb{1}\{h \leq T_\delta\} \\
&\leq 3 \sum_{h \in 3\mathbb{N}, h<t} \mathbb{1}\{\exists \boldsymbol{s} \in \S \quad s.t. \quad \max_i N_{i,h}(\boldsymbol{s}) 4 b_h \lambda c_i(\boldsymbol{s}) - \frac{\lambda}{\sigma^2}\} \\
&\leq 12 H^*(u) b_t \lambda - \frac{3\lambda |S_1||S_2|}{\sigma^2} \leq t,
\end{aligned}$$

which concludes the proof. □

5.3 LUCB-NG

The second algorithm we propose, LUCB-NG, is another algorithm to learn pure NEs in simulation-based games. This is inspired by the LUCB-G algorithm proposed by Zhou et al. [47] which is, as we anticipated, based on the idea of using upper and lower confidence bounds on the players' utilities so as to guide the querying process. In particular, the difference between their algorithm and ours is that they focus only on *two-player zero-sum games*, while we extend the approach to *n-player general-sum games*. As the name of their algorithm LUCB-G was due to the fact that they used LUCB for Games, we call our algorithm LUCB-NG, to underline the fact that ours can handle *n-player games*. On the other hand, the difference between LUCB-NG and the algorithm we proposed in the previous section, PNE-LUCB, is that the previous one treated the problem of finding the NEs in the game as a single best arm identification problem, while this second one divides the search into smaller sub-problems, always of best arm identification. Specifically, it aims to find, for every player, the *best response* to every strategy profile of the other players. Now, before going into the specifics of the algorithm, we need to provide some additional notation, useful to better describe the problem that is being treated.

Let us consider a player $i \in N$ and let $\boldsymbol{s}_{-i} \in S_{-i}$ be a strategy profile of the other players (that we also call sub-problem), we want to find, as we said, the best response of player i to $\boldsymbol{s}_{-i}$, that is:

$$s_i^* (\boldsymbol{s}_{-i}) := \arg\max_{s_i \in S_i} u_i (s_i, \boldsymbol{s}_{-i}) . \tag{5.8}$$

Since we do not have the true utility function $u_i(\boldsymbol{s})$ for every player and for every strategy profile but only an estimate of it, we can use the lower and upper bounds to identify an estimate of $s_i^*(s_{-i})$, namely $\hat{s}_i^*(\boldsymbol{s}_{-i})$. Specifically, given a player $i \in N$ and a strategy profile of the other players $\boldsymbol{s}_{-i} \in S_{-i}$, at a certain time step t, $\hat{s}_i^*(\boldsymbol{s}_{-i})$ is defined as:

$$\hat{s}_i^*(\boldsymbol{s}_{-i}) = \{s_i \in S_i | L_i^t(s_i, \boldsymbol{s}_{-i}) > U_i^t(s_i', \boldsymbol{s}_{-i}) - \epsilon \quad \forall s_i' \in S_i, s_i' \neq s_i\}. \quad (5.9)$$

Notice that, if at a certain time step t a strategy of player $i \in N$, namely s_i, is selected to be $\hat{s}_i^*(\boldsymbol{s}_{-i})$, then for every time step t', $t' > t$ this continues to hold because the bounds can only be refined, so the lower bounds can only be increased and the upper bounds can only be decreased. If $\hat{s}_i^*(\boldsymbol{s}_{-i})$ has already been defined for a player $i \in N$ and for a given strategy profile of the other players $\boldsymbol{s}_{-i} \in S_{-i}$, then we do not want to analyze it anymore, so we would like to be able to quickly identify which are the sub-problems that we should take into consideration at time step t. For this reason, we define a set, $as_i(t)$, that for every player $i \in N$ keeps track of the strategy profiles of the other players $\boldsymbol{s}_{-i} \in S_{-i}$ for which, at round t, the best strategy of player i has not already been defined. Specifically, given a player $i \in N$ and a time step t, $as_i(t)$ is defined as:

$$as_i(t) := \{\boldsymbol{s}_{-i} \in S_{-i} : s_i^*(\boldsymbol{s}_{-i}) \text{ not identified until round } t\}. \quad (5.10)$$

Moreover, the algorithm keeps track of how many times a sub-problem has been selected, that is, how many times, given a player $i \in N$, a strategy profile of the other players $\boldsymbol{s}_{-i} \in S_{-i}$ has been selected with the purpose of finding player i's best response. We denote this with $sel(\boldsymbol{s}_{-i})$. Finally, as for the previous algorithm, the algorithm keeps track of a set $\bar{E}$ of strategy profiles it is already sure they are NEs. The condition that they need to satisfy to be considered NEs is equivalent to the one provided for the previous algorithm, but for clarity we rewrite it here again with the newly introduced notation. A strategy profile $\boldsymbol{s} \in S$ is considered to be an NE if, at some time t, it holds that:

$$s_i = \hat{s}_i^*(\boldsymbol{s}_{-i}) \quad \text{for every} \quad i \in N, \quad (5.11)$$

where $\boldsymbol{s} := (s_i, \boldsymbol{s}_{-i})$.

We propose two variants of the algorithm, one in which the goal is to find only one Nash equilibrium and then one in which the goal is to find all the Nash equilibria. The only difference between these two variants is an additional possibility in the stopping rule in the case in which we are

looking for a single NE. The algorithm, in both the variants, can handle the situation in which there are no Nash equilibria.

The main components of the LUCB-NG algorithm are defined as follows.

- *Querying rule*: At the end of every round t such that there is no strategy profile left to be queried, the algorithm selects the strategy profiles to be queried in the next rounds as follows. Let $E^t \subseteq S$ be the set of pure NEs in the normal-form game whose utility vector function is defined by the estimates $\hat{u}_i^t(\boldsymbol{s})$ computed up to round t. In the case in which E^t is not empty, the algorithm checks whether there are strategy profiles $\boldsymbol{s} \in E^t$ that are already in $\bar{E}$ and, if it is the case, it removes them from E^t since it is already sure that they are NEs and it does not want to work on them anymore. If, after this check, E^t is not empty, let us assume that the equilibria it contains are ordered according to some (arbitrary) criterion, and that each of them is assigned a label in $\{0, \dots, |E^t|\}$ according to its position in the ordering. Moreover, for each label $k \in \{0, \dots, |E^t|\}$, let $t_k \coloneqq t + 1 + k(n+1)$. Following the ordering, the algorithm takes each strategy profile in E^t in turn, and, for each of them, it queries a set of $n+1$ strategy profiles, defined as follows. Suppose that $\boldsymbol{s} \in E^t$ is the k-th strategy profile in E^t, according to the ordering. Then, the algorithm queries the following strategy profiles during the rounds going from t_k to $t_{k+1} - 1$:

$$\boldsymbol{s}^{t_k} \coloneqq \boldsymbol{s} \tag{5.12a}$$

$$\boldsymbol{s}^{t_k+i} \coloneqq \Big(\operatorname*{arg\,max}_{s_i' \in S_i : s_i' \neq s_i} U_i^t(s_i', \boldsymbol{s}_{-i}), \boldsymbol{s}_{-i} \Big) \quad \text{for every} \quad i \in N. \tag{5.12b}$$

where $\boldsymbol{s}^{t_k+i} = (s_i', \boldsymbol{s}_{-i})$ is the strategy profile obtained by replacing s_i, the strategy of player i in $\boldsymbol{s}^{t_k}$, with s_i', assuming that all the other players continue to play the strategy profile $\boldsymbol{s}_{-i}$ prescribed by $\boldsymbol{s}^{t_k}$. So, Equation (5.12a) prescribes to query exactly the NE that is being taken into consideration, that is the k-th, while Equation (5.12b) looks for the best response, according to the values of the upper bounds, to the considered NE for every player $i \in N$. Notice that, at round t, the algorithm decides which strategy profiles are to be queried in the following $(n+1)|E^t|$ rounds. At this point, we can notice that this first part of the LUCB-NG algorithm is almost equivalent to the one of PNE-LUCB. What makes the big difference between the two is the way of acting in the case in which E^t, at a certain time step t, results to be empty. If E^t is empty, the algorithm selects, for every player $i \in N$, if it exists, a strategy profile of the other players $\boldsymbol{s}_{-i} \in S_{-i}$ that

is the one that has been selected the lowest number of times among the ones in $as_i(t)$ at round t, that is:

$$\boldsymbol{s}^t_{-i} = \min_{\boldsymbol{s}_{-i} \in as_i(t)} sel(\boldsymbol{s}_{-i}). \tag{5.13}$$

Notice that $\boldsymbol{s}^t_{-i}$ could not exist for a player $i \in N$ if $\hat{s}^*_i(\boldsymbol{s}_{-i})$ has already been defined $\forall \boldsymbol{s}_{-i} \in S_{-i}$. Equivalently, we could say that for that player, at round t, $as_i(t)$ is empty. So, let us denote with $N^t \subseteq N$ the subset of players for which a strategy profile of the other players has been selected at round t, let us order them according to some (arbitrary) criterion, and let us assign to everyone of them a label in $\{0, \ldots, |N^t|\}$ according to its position in the ordering. Moreover, for each label $k \in \{0, \ldots, |N^t|\}$, let $t_k \coloneqq t+1+2k$. Following the ordering, the algorithm takes each player in N^t in turn, and, for each of them, it queries 2 strategy profiles, defined as follows. Suppose that $i \in N^t$ is the k-th player in N^t, according to the ordering. Then, the algorithm queries the following strategy profiles during the rounds t_k and $t_k + 1$:

$$\boldsymbol{s}^{t_k} \coloneqq \Big(\arg\max_{s_i \in S_i} \hat{u}_i(s_i, \boldsymbol{s}^t_{-i}), \boldsymbol{s}^t_{-i} \Big) \tag{5.14a}$$

$$\boldsymbol{s}^{t_k+1} \coloneqq \Big(\arg\max_{s'_i \in S_i : s'_i \neq s_i} U^t_i(s'_i, \boldsymbol{s}^t_{-i}), \boldsymbol{s}^t_{-i} \Big), \tag{5.14b}$$

where $\boldsymbol{s}^{t_k} = (s_i, \boldsymbol{s}^t_{-i})$ represents the best response of player i, according to the estimated utilities, to the strategy profile of the other players $\boldsymbol{s}^t_{-i}$ selected to be analyzed at time t, while Equation (5.14b) looks for the strategy profile $\boldsymbol{s}^{t_k+1} = (s'_i, \boldsymbol{s}^t_{-i})$ that represents the best response of player i, according to the values of the upper bounds, to the same strategy profile of the other players $\boldsymbol{s}^t_{-i}$ selected to be analyzed at time t, but excluding the strategy s_i selected by Equation (5.14a). Notice that, at round t, the algorithm decides which strategy profiles are to be queried in the following $2|N^t|$ rounds.

- *Stopping rule*: The algorithm terminates at the end of a round t if one of the following conditions is satisfied:
  - *(*Looking for a single NE): There is at least one element in $\bar{E}$, that is $|\bar{E}| > 0$;
  - $\hat{s}^*_i(\boldsymbol{s}_{-i})$ has been determined for all $i \in N$ and for all $\boldsymbol{s}_{-i} \in S_{-i}$. Equivalently we could say that at round t, for every player $i \in N$, $as_i(t)$ is empty.

- *Guessing rule*: Upon termination the algorithm outputs the set $\bar{E}$. Notice that it could contain zero, one or multiple equilibria.

As we said for the previous algorithm, the idea behind the selection process adopted by the algorithm is to advance towards the termination condition. In order to move towards the termination condition, the algorithm needs to refine the lower bounds $L_i^t(\boldsymbol{s})$ (*i.e.*, it must increase them) and the upper bounds $U_i^t(\boldsymbol{s}')$ (*i.e.*, it must decrease them). In order to do so, the relevant strategy profiles must be queried so as to shrink their confidence intervals. These are exactly the strategy profiles selected by the querying rules defined by Equation (5.12) and Equation (5.14).

5.3.1 Pseudocode

In the following, we provide the pseudocode of the algorithm we have just described in order to better clarify its functioning. Notice that the procedures that we use are the same, or are very similar, to the one provided for the previous algorithm. For this reason, here we report only the small changes that we have to take into account and not again their complete description. In particular:

- computePureNE($\bar{\Gamma}$): computes the pure NE in the estimated game as before but, if there are no strategy profiles left, it does not take a strategy profile at random but it returns the empty set $\emptyset$;
- checkNE($\boldsymbol{s}$): checks the condition of Equation (5.11) for the strategy profile $\boldsymbol{s}$ and returns True if it is satisfied, False otherwise;

Algorithm 4 LUCB-NG

Input: boolean allNE, True if looking for all the Nash
Output: $\bar{E}$

Initialize $t \leftarrow 0$, $\bar{E} \leftarrow \emptyset$
Initialize $as_i(t) \leftarrow \emptyset$, $sel(\boldsymbol{s}_{-i}) \leftarrow 0 \quad \forall i \in N, \boldsymbol{s}_{-i} \in S_{-i}$
pullAllStrategyProfiles(S)
while (allNE **or** $\bar{E} == \emptyset$) **and any** $as_i(t) \neq \emptyset$ **do** ▷ Stopping rules
  $E^t \leftarrow$ computePureNE($\bar{\Gamma}$)
  if $E^t \neq \emptyset$ **then**
    for $\boldsymbol{s} \in E^t$ **do**
      pullStrategyProfile($\boldsymbol{s}$)
      for $i \in N$ **do**
        $sel(\boldsymbol{s}_{-i}) \leftarrow sel(\boldsymbol{s}_{-i}) + 1$ ▷ $\boldsymbol{s} = (s_i, \boldsymbol{s}_{-i})$
        $\boldsymbol{s}^i \leftarrow \left(\arg\max_{s'_i \in S_i : s'_i \neq s_i} U_i^t(s'_i, \boldsymbol{s}_{-i}), \boldsymbol{s}_{-i} \right)$ ▷ Equation (5.12b)
        pullStrategyProfile($\boldsymbol{s}^i$)
      end for
    end for
    $t \leftarrow t + (n+1) \cdot |E^t|$
  else
    $N^t \leftarrow \{i \in N | as_i(t) \neq \emptyset\}$
    for $i \in N^t$ **do**
      $\boldsymbol{s}_{-i} \leftarrow \min_{\boldsymbol{s}_{-i} \in as_i(t)} sel(\boldsymbol{s}_{-i})$ ▷ Equation (5.13)
      $sel(\boldsymbol{s}_{-i}) \leftarrow sel(\boldsymbol{s}_{-i}) + 1$
      $\boldsymbol{s} = \left(\arg\max_{s_i \in S_i} \hat{u}_i(s_i, \boldsymbol{s}_{-i}), \boldsymbol{s}_{-i} \right)$ ▷ Equation (5.14a)
      pullStrategyProfile($\boldsymbol{s}$)
      $\boldsymbol{s}' = \left(\arg\max_{s'_i \in S_i : s'_i \neq s_i} U_i^t(s'_i, \boldsymbol{s}_{-i}), \boldsymbol{s}_{-i} \right)$ ▷ Equation (5.14b)
      pullStrategyProfile($\boldsymbol{s}'$)
    end for
    $t \leftarrow t + 2 \cdot |N^t|$
  end if
  $\bar{E} \leftarrow \bar{E} + \{\boldsymbol{s} \in E^t | \text{checkNE}(\boldsymbol{s})\}$ ▷ Equation (5.11)
end while
return $\bar{E}$

5.3.2 Example

Since the steps performed by the algorithm LUCB-NG in the case in which, at a certain time step t, there are pure NEs in the estimated game are very similar to the ones performed by the previous algorithm, PNE-LUCB, that have been already shown in the related example, here we prefer to show the behaviour of the algorithm in the case in which, at a certain time step t, it does not find any pure NE in the estimate game.

Assume that we are running LUCB-NG to learn our a game and that, at a certain time step t, the utilities of the estimated game, denoted with $\hat{\boldsymbol{u}}^t(\boldsymbol{s})$ are the ones reported in Figure 5.12, while the lower and upper bounds

	L	C	R
T	1,2	2,1	3,0
M	2,2	5,0	2,1
B	3,1	1,0	1,2

Figure 5.12: Estimated game.

are, for simplicity, for every strategy profile $\boldsymbol{s} \in S$, $\hat{\boldsymbol{u}}^t(\boldsymbol{s}) \pm 2$. We can represent them as two games, whose utilities are defined by the values of the lower an upper bounds for each strategy profile $\boldsymbol{s} \in S$, as they are reported in Figure 5.13 and Figure 5.14 respectively. Firstly our algorithm tries to

	L	C	R
T	-1,0	0,-1	1,-2
M	0,0	3,-2	0,-1
B	1,-1	-1,-2	-1,0

Figure 5.13: Lower bounds game.

	L	C	R
T	3,4	4,3	5,3
M	4,4	7,2	4,3
B	5,3	3,2	3,4

Figure 5.14: Upper bounds game.

compute the equilibria in the estimated game but, since it finds out that there are no NE in the game it is considering at time t, it proceeds with the alternative steps: for every player $i \in N$, it looks for the sub-problem that has been selected the lowest number of times. In this specific case, assume that no sub-problem has been solved yet, so $as_1(t) = \{L, C, R\}$, since player 1 has to find a best response to every strategy of player 2, while $as_2(t) = \{T, M, B\}$, since player 2 has to find a best response to every strategy of player 1. So the algorithm needs to select for player 1 a sub-problem among $\{L, C, R\}$ and for player 2 a sub-problem among $\{T, M, B\}$. In order to take this decision, it looks at the number of times that every sub-problem has been selected in the previous steps, which for what concerns our

example are reported in Figure 5.15. In particular, the number at the end of a row, for example T, represents the number of times the sub-problem T has been selected for player 2, that is $sel(T)$, while the number at the end of a column, for example L, represents the number of times the sub-problem L has been selected for player 1, that is $sel(L)$. At this point, considering

	L	C	R	
T	1,2	2,1	3,0	3
M	2,2	5,0	2,1	2
B	3,1	1,0	1,2	1
	3	1	2	

Figure 5.15: Normal form game.

player 1, the algorithm selects the sub-problem C, since it is the one that has been selected the lowest number of times, and:

- $sel(C) = sel(C) + 1$;
- The best response of player 1 to the strategy C of player 2, considering the estimated game, is M, so the strategy profile (M, C) is selected (colored in light blue in Figure 5.16);
- The best response of player 1 to the strategy C of player 2, considering the game defined by the upper bounds and excluding the strategy selected in the previous step (M), is T, so the strategy profile (T, C) is selected (colored in light blue in Figure 5.17);

		↓	
	L	C	R
T	1,2	2,1	3,0
M	2,2	5,0	2,1
B	3,1	1,0	1,2

Figure 5.16: Estimated game.

		↓	
	L	C	R
T	3,4	4,3	5,3
M	4,4	7,2	4,3
B	5,3	3,2	3,4

Figure 5.17: Upper bounds game.

so the algorithm pulls the strategy profiles (M, C) and (T, C). Considering player 2, the algorithm selects the sub-problem B, since it is the one that has been selected the lowest number of times, and:

- $sel(B) = sel(B) + 1$;

- The best response of player 2 to the strategy B of player 1, considering the estimated game, is R, so the strategy profile (B, R) is selected (colored in yellow in Figure 5.18);
- The best response of player 2 to the strategy B of player 1, considering the game defined by the upper bounds and excluding the strategy selected in the previous step (R), is L, so the strategy profile (B, L) is selected (colored in yellow in Figure 5.19);

	L	C	R
T	1,2	2,1	3,0
M	2,2	5,0	2,1
→ B	3,1	1,0	1,2

Figure 5.18: Estimated game.

	L	C	R
T	3,4	4,3	5,3
M	4,4	7,2	4,3
→ B	5,3	3,2	3,4

Figure 5.19: Upper bounds game.

so the algorithm pulls the strategy profiles (B, R) and (B, L). We can see that, as we said in the description of the algorithm, at time t, in the case in which there are no NE in the estimated game, the algorithm decided which strategy profiles were to be queried in the following $2|N^t|$ round, that in our case corresponds to $2 \cdot (2) = 4$ strategy profiles pulled. At this point, that is at the end of the querying phase, regardless of which rules have been followed to decide which strategies to query (The case in which there are NE in the estimated game at time t versus the case we have described in which there are no NE in the estimated game at time t) the algorithm, before checking if there is some strategy profile that has revealed to be an NE thanks to the new samples, that is that it satisfies the condition of Equation (5.11), needs to update, for every player $i \in N$, the auxiliary structures $as_i(t)$ and $\hat{s}_i^*(\cdot)$. In order to this, it has to check, for every sub-problem, if a solution has been found, that is that there is a strategy profile $\boldsymbol{s}$ which is, with high probability, the best strategy profile for that sub-problem, this means that it should satisfy the condition of Equation (5.9). After this, it should check if there is a strategy profile $\boldsymbol{s}$ that is the best for every sub-problem in which it is involved, that is that it satisfies the condition of Equation (5.11) and so it is an NE. For the sake of brevity, here we only show, for every player $i \in N$, the update of $as_i(t)$ for the sub-problem he/she has selected in this example. Assume for simplicity that the estimated game does not change and that only the lower and upper bounds of the pulled strategy profile change, and that they all change in the same way, that is they become, for every strategy profile $\boldsymbol{s} \in S$ that has been pulled, $\hat{u}_i^t(\boldsymbol{s}) \pm 1$, as reported in Figure 5.20 and Figure 5.21. Considering the sub-problem selected by player 1, that is C, we

	L	C	R
T	-1,0	1,0	1,-2
M	0,0	4,-1	0,-1
B	2,0	-1,-2	0,1

Figure 5.20: New lower bounds game.

	L	C	R
T	3,4	3,2	5,3
M	4,4	6,1	4,3
B	4,2	3,2	2,3

Figure 5.21: New upper bounds game.

see that there is a strategy, M, and so, as a consequence, a strategy profile, (M, C), that for him/her satisfies the condition of Equation (5.9). In fact, its lower bound (colored in green in Figure 5.22) is greater than the upper bounds of every possible deviations (colored in light blue in Figure 5.23 and so we can say that $\hat{s}_1^*(C) = M$ and remove C from the sub-problems that player 1 has to solve. Notice that we cannot assert that (M, C) is a NE until it is not true that also $\hat{s}_2^*(M) = C$, and so that the condition of Equation (5.11) is satisfied. On the other hand, considering the sub-problem

	L	C	R
T	-1,0	1,0	1,-2
M	0,0	4,-1	0,-1
B	2,0	-1,-2	0,1

Figure 5.22: New lower bounds game.

	L	C	R
T	3,4	3,2	5,3
M	4,4	6,1	4,3
B	4,2	3,2	2,3

Figure 5.23: New upper bounds game.

selected by player 2, that is B, we see that there is not a strategy that for him/her satisfies the condition of Equation (5.9). In fact, even the greatest of its lower bounds, the one associated to the strategy R, and so to strategy profile (B, R) (colored in green in Figure 5.24), is lesser than the upper bounds of every possible deviations (colored in yellow in Figure 5.25) and so we can not conclude nothing.

	L	C	R
T	-1,0	1,0	1,-2
M	0,0	4,-1	0,-1
B	2,0	-1,-2	0,1

Figure 5.24: New lower bounds game.

	L	C	R
T	3,4	3,2	5,3
M	4,4	6,1	4,3
B	4,2	3,2	2,3

Figure 5.25: New upper bounds game.

5.4 MNE-LUCB

In this section, differently from the previous ones, we propose an algorithm to learn a mixed Nash equilibrium in simulation-based games, named MNE-

LUCB. It extends our PNE-LUCB to learn a mixed Nash equilibrium so, as the previous algorithms, it is based on the same idea of using upper and lower confidence bounds on the players' utilities to guide the querying process. The meaning of the name is Mixed Nash Equilibrium LUCB. The advantage of learning a mixed NE instead of pure NE relies on the fact that it is guaranteed to exist (Nash [26]), so we do not need to handle situations in which the game does not have an equilibrium like we had to do for the algorithms in which we were only looking for pure NE. Moreover, recall that a pure NE can also be seen as a mixed NE with all the probabilities set to 0 or 1, but the opposite is not possible, so the algorithm presented in this section could also be used, in principle, to learn pure NE, but none of the previous algorithms could be used to learn a mixed NE. On the other hand, there are also some disadvantages: first of all, as it is true that at least one mixed NE is always guaranteed to exist, it is also true that the mixed NEs in a game could be an infinite number, making impossible to design an algorithm able to learn all of them and then stop. For this reason our algorithm only looks for one mixed NE and it stops after it finds it. Secondly, unfortunately, proving theoretical guarantees for such an algorithm is much more complex than for the previous ones, for this reason we only provide experimental and empirical results.

The main components of the MNE-LUCB algorithm are defined as follows.

- *Querying rule*: At the end of every round t such that there is no strategy profile left to be queried, the algorithm selects the strategy profiles to be queried in the next rounds as follows. Let consider the normal-form game whose utility vector function is defined by the estimates $\hat{u}_i^t(\boldsymbol{s})$ computed up to round t. Let consider a mixed strategy profile $\boldsymbol{x}^t = (x_i^t)_{i \in N}$ that is a mixed NE for this game at round t and assume, for easiness of notation, that $t' = t + 1$, which means that t' represents the time step when we start the prescribed samples. Denoting with S_i^t the set of strategies of player i played with positive probability in $\boldsymbol{x}^t$, the algorithm, for every player $i \in N$, looks for the mixed strategy profile $\boldsymbol{x}^{t'+i}$ such that:

$$\boldsymbol{x}^{t'+i} \coloneqq \Big(\arg\min_{s_i' \in S_i^t} L_i^t(s_i', \boldsymbol{x}^t), \boldsymbol{x}_{-i}^t \Big), \tag{5.15}$$

  where $\boldsymbol{x}^{t'+i} = (s_i', \boldsymbol{x}_{-i}^t)$ is the strategy profile obtained by replacing the strategy of player i in $\boldsymbol{x}^t$ with the pure strategy s_i', assuming that all the other players continue to play the joint strategy profile $\boldsymbol{x}_{-i}^t$

prescribed by $\boldsymbol{x}^t$, and $L_i^t(s_i', \boldsymbol{x}_t) \coloneqq \sum_{\boldsymbol{s} \in S : s_i = s_i'} L_i^t(\boldsymbol{s}) \prod_{j \in N,\, j \neq i} x_j(s_j)$. The mixed strategy profile prescribed by Equation (5.15) is the one that has the worst value among the lower bounds on the utility of player $i \in N$, assuming that player i plays a pure strategy among the ones he/she played with positive probability in $\boldsymbol{x}^t$ and that all the other players continue playing the same mixed strategies prescribed by $\boldsymbol{x}^t$. Notice that, since $\boldsymbol{x}^t$ has been selected to be a mixed NE in the estimated game at time t, the values of these strategy profiles, looking at the estimated utilities, should be all the same, but they could, in principle, be different considering the lower bounds since their values are based on how many times each of them has been pulled during the searching. At this point, since we can only sample pure strategy profiles, we need to select only one of the pure strategy profiles $\boldsymbol{s} \in S$ played with positive probability by all the players in the mixed strategy profile $\boldsymbol{x}^{t'+i}$. In order to do this, the algorithm assigns to every pure strategy profile a probability of being pulled equal to the probability with which it is played in the mixed strategy profile $\boldsymbol{x}^{t'+i}$, that is, for every $\boldsymbol{s} \in S$, the probability of being pulled is $\mathbb{P}(\boldsymbol{s}) = \prod_{i \in N} x_i^{t'+i}(s_i)$, where $\boldsymbol{x}^{t'+i} = (x_i^{t'+i})_{i \in N}$ and s_i is the strategy of player i in $\boldsymbol{s}$. Finally, the algorithm selects the strategy profile to be pulled taking into account the selection probability just assigned. We have resorted to this kind of selection because we would have liked to pull, as we did in the previous algorithms, the NE found for the estimated game in order to refine its bounds but, since it is a mixed NE, there could have been a variable number of involved pure strategy profiles and pulling all of them would have made impossible for us to correctly count the number of strategy profiles sampled in each round of the algorithm. Following this procedure, we can say that at this point the algorithm has pulled one strategy profile for player, so $|N|$ strategy profiles. Notice that, if the equilibrium $\boldsymbol{x}^t$ is pure and not mixed, we have pulled $|N|$ times just $\boldsymbol{x}^t$. Assuming again, for easiness of notation, that $t'' = t + 1 + |N|$, which means that t'' represents the time step when we start the prescribed samples after the ones that we have already done. So again, for every player $i \in N$, the algorithm looks for the mixed strategy profile $\boldsymbol{x}^{t''+i}$ such that:

$$\boldsymbol{x}^{t''+i} \coloneqq \Big(\arg\max_{s_i' \in S_i} U_i^t(s_i', \boldsymbol{x}^t), \boldsymbol{x}_{-i}^t \Big), \tag{5.16}$$

where $\boldsymbol{x}^{t''+i} = (s_i', \boldsymbol{x}_{-i}^t)$ is the strategy profile obtained by replacing the strategy of player i in $\boldsymbol{x}^t$ with the pure strategy s_i', assuming that

all the other players continue to play the joint strategy profile $\boldsymbol{x}_{-i}^t$ prescribed by $\boldsymbol{x}^t$. Moreover, analogously to what we have reported for the lower bound, recall that $U_i^t(s_i', \boldsymbol{x}_t) \coloneqq \sum_{\boldsymbol{s} \in S : s_i = s_i'} U_i^t(\boldsymbol{s}) \prod_{j \in N,\, j \neq i} x_j(s_j)$. The mixed strategy profile prescribed by Equation (5.16) represents the best pure deviation of player $i \in N$, according to the values of the upper bounds, from the considered mixed NE $\boldsymbol{x}^t$, assuming that all the other players continue playing the same mixed strategies prescribed by $\boldsymbol{x}^t$. Notice that, in this case, he/she can selects any of his/her actions and not only the ones he/she played with positive probability in $\boldsymbol{x}^t$. In the case in which the strategy played in $\boldsymbol{x}^t$ by player i is pure, he/she cannot select it again in $\boldsymbol{x}^{t''+i}$ because it has to be a deviation, so different from the original one. In that case, it would be better to say that, if in the mixed strategy profile $\boldsymbol{x}^t$ player i is playing the pure strategy s_i, then the mixed strategy profile $\boldsymbol{x}^{t''+i}$ that the algorithm is looking for should be defined as:

$$\boldsymbol{x}^{t''+i} \coloneqq \Big(\underset{s_i' \in S_i : s_i' \neq s_i}{\arg\max}\ U_i^t(s_i', \boldsymbol{x}^t), \boldsymbol{x}_{-i}^t \Big), \tag{5.17}$$

where $\boldsymbol{x}^{t''+i} = (s_i', \boldsymbol{x}_{-i}^t)$ is the strategy profile obtained by replacing the strategy of player i in $\boldsymbol{x}^t$, s_i, with the pure deviation s_i', assuming that all the other players continue to play the joint strategy profile $\boldsymbol{x}_{-i}^t$ prescribed by $\boldsymbol{x}^t$. As before, since we can only sample pure strategy profiles, we need to select only one of the pure strategy profiles $\boldsymbol{s} \in S$ played with positive probability by all the players in the mixed strategy profile $\boldsymbol{x}^{t''+i}$. The algorithm assigns to every pure strategy profile a probability of being pulled equal to the probability with which it is played in the mixed strategy profile $\boldsymbol{x}^{t''+i}$, that is, for every $\boldsymbol{s} \in S$, the probability of being pulled is $\mathbb{P}(\boldsymbol{s}) = \prod_{i \in N} x_i^{t''+i}(s_i)$, where $\boldsymbol{x}^{t''+i} = (x_i^{t''+i})_{i \in N}$ and s_i is the strategy of player i in $\boldsymbol{s}$. Finally, the algorithm selects the strategy profile to be pulled taking into account the selection probability just assigned. As we explained before, also in this case we would have liked to pull the best pure deviation of each player $i \in N$ from the mixed NE in order to refine its bounds but, even if player i deviates to a pure strategy, all the other players continue playing the mixed strategies prescribed by $\boldsymbol{x}^t$ and so, again, there could have been a variable number of involved pure strategy profiles and pulling all of them would have made impossible for us to correctly count the number of strategy profiles sampled in each round of the algorithm. Following this procedures, we can say that at this point the algorithm has pulled again one strategy profile for player,

so we can say that, at round t, the algorithm decides which strategy profiles are to be queried in the following $2\,|N|$ rounds.

- *Stopping rule*: The algorithm terminates at the end of a round t in which the following condition is satisfied for the considered strategy profile $\boldsymbol{x}^t$: for every player $i \in N$, and for every action deviation $s'_i \in S_i$:

$$L_i^t(\boldsymbol{x}^t) > U_i^t(s'_i, \boldsymbol{x}^t) - \epsilon, \tag{5.18}$$

  where the left term represents the lower bound on the expected utility of player i given the strategy profile prescribed by $\boldsymbol{x}^t$, while the right term represents player i's expected utility upper bound when playing the pure strategy s'_i at time t, given that the other players play the mixed strategies prescribed by $\boldsymbol{x}^t$.

- *Guessing rule*: Upon termination, the algorithm outputs $\boldsymbol{x}^t$, which is the last computed mixed strategy profile and the first that has satisfied the condition of Equation (5.18).

5.4.1 Pseudocode

In the following, we provide the pseudocode of the algorithm we have just described in order to better clarify its functioning. Notice that the procedures that we use are the same, or are very similar, to the one provided for the previous algorithms. For this reason, here we report only the small changes that we have to take into account and not again their complete description. In particular:

- computeOneMixedNE($\bar{\Gamma}$): computes only one mixed Nash equilibria in the estimated game $\bar{\Gamma}$;

- pullOnePureProfile($\boldsymbol{x}$): assigns to every pure strategy profile $\boldsymbol{s} \in S$ a probability of being pulled equal to the probability with which it is played in the mixed strategy profile $\boldsymbol{x}$, selects a pure a strategy profile $\boldsymbol{s}$ and then calls pullStrategyProfile($\boldsymbol{s}$);

- playsPureStrategy($i, \boldsymbol{x}$): returns True if player i is playing a pure strategy in the mixed strategy profile $\boldsymbol{x}$, False otherwise;

- checkNE($\boldsymbol{x}$): checks the condition of Equation (5.18) for the strategy profile $\boldsymbol{x}$ and returns True if it is satisfied, False otherwise.

Algorithm 5 MNE-LUCB

Output: $\boldsymbol{x}^t$

Initialize $t \leftarrow 0$, *stopping* = FALSE
pullAllStrategyProfiles(S)
while not *stopping* **do**
  $\boldsymbol{x}^t \leftarrow$ computeOneMixedNE($\bar{\Gamma}$)
  for $i \in N$ **do**
    $\boldsymbol{x}^i \leftarrow \left(\arg\min_{s_i' \in S_i^t} L_i^t(s_i', \boldsymbol{x}^t), \boldsymbol{x}_{-i}^t \right)$ ▷ Equation (5.15)
    pullOnePureProfile($\boldsymbol{x}^i$)
    if not playsPureStrategy($i, \boldsymbol{x}^t$) **then**
      $\boldsymbol{x}^{i'} \leftarrow \left(\arg\max_{s_i' \in S_i} U_i^t(s_i', \boldsymbol{x}^t), \boldsymbol{x}_{-i}^t \right)$ ▷ Equation (5.16)
    else
      $\boldsymbol{x}^{i'} \leftarrow \left(\arg\max_{s_i' \in S_i : s_i' \neq s_i} U_i^t(s_i', \boldsymbol{x}^t), \boldsymbol{x}_{-i}^t \right)$ ▷ Equation (5.17)
    end if
    pullOnePureProfile($\boldsymbol{x}^{i'}$)
  end for
  $t \leftarrow t + 2 \cdot |N|$
  stopping $\leftarrow$ checkNE($\boldsymbol{x}^t$) ▷ Equation (5.18)
end while
return $\boldsymbol{x}^t$

5.4.2 Examples

Example 1: General case

Assume that we are running MNE-LUCB to learn a game and that, at a certain time step t, the utilities of the estimated game, denoted with $\hat{\boldsymbol{u}}^t(\boldsymbol{s})$, are the ones reported in Figure 5.26, while the lower and upper bounds

	L	C	R
T	1,0	2,-2	2,1
M	0,1	2,3	3,0
B	0,-1	3,-1	-1,2

Figure 5.26: Estimated game.

are for some strategy profile $\boldsymbol{s} \in S$, $\hat{\boldsymbol{u}}^t(\boldsymbol{s}) \pm 1$, for some others $\hat{\boldsymbol{u}}^t(\boldsymbol{s}) \pm 2$ and again for some others $\hat{\boldsymbol{u}}^t(\boldsymbol{s}) \pm 3$. The assignment is quite casual but is important to differentiate a bit among the bounds in order to show the functioning of the algorithm that otherwise could not be clear. As usual, we can represent them as two games, whose utilities are defined by the values of the lower an upper bounds for each strategy profile $\boldsymbol{s} \in S$, as they are reported in Figure 5.27 and Figure 5.28 respectively. Firstly our algorithm

	L	C	R
T	0,-1	0,-4	1,0
M	-2,-1	0,1	1,-2
B	-3,-4	0,-4	-4,-1

Figure 5.27: Lower bounds game.

	L	C	R
T	2,1	4,0	3,2
M	2,3	3,2	5,2
B	3,2	6,2	2,5

Figure 5.28: Upper bounds game.

computes one mixed Nash equilibrium of the estimated game, that in our case is the one reported in Figure 5.29 and that we indicate with $\boldsymbol{x}$. Notice

	L	C	R	
T	1,0	2,-2	2,1	1/2
M	0,1	2,3	3,0	1/2
B	0,-1	3,-1	-1,2	0
	1/6	2/3	1/6	

Figure 5.29: Estimated game.

that, in principle, the equilibrium computed could also be pure and the algorithm would work correctly also in that case. We show it in the following

example. At this point, considering the first player, the algorithm selects the strategy, among the ones played with positive probability by player 1 in the equilibrium (that in this case are T and M) that is the one that, given that player 2 continue playing the strategy prescribed by the NE, has the worst lower bound assuming that player 1 plays a pure strategy. Recall that, since $\boldsymbol{x}$ has been selected to be an NE in the estimated game, the estimated values of the mixed strategy profiles in which player 1 plays a pure strategy between T and M and player 2 continue playing the mixed strategy prescribed by $\boldsymbol{x}$ results always to be equal. More precisely:

- The expected value of player 1 for playing the pure strategy T, looking at the estimated utilities, is: $\mathbb{E}[T] = 1 \cdot \frac{1}{6} + 2 \cdot \frac{2}{3} + 2 \cdot \frac{1}{6} = \frac{11}{6}$, while looking at the lower bounds is: $\mathbb{E}[T] = 0 \cdot \frac{1}{6} + 0 \cdot \frac{2}{3} + 1 \cdot \frac{1}{6} = \frac{1}{6}$;
- The expected value of player 1 for playing the pure strategy M, looking at the estimated utilities, is: $\mathbb{E}[M] = 0 \cdot \frac{1}{6} + 2 \cdot \frac{2}{3} + 3 \cdot \frac{1}{6} = \frac{11}{6}$, while looking at the lower bounds is: $\mathbb{E}[M] = -2 \cdot \frac{1}{6} + 0 \cdot \frac{2}{3} + 1 \cdot \frac{1}{6} = -\frac{1}{6}$.

As we said, the two values considering the estimated utilities are the same, but looking at the lower bounds are different. In particular, we have that $\mathbb{E}[M] < \mathbb{E}[T]$ and so the algorithm, according to Equation (5.15), selects strategy M. At this point, we have 3 candidates strategy profiles, those reported in light blue in Figure 5.30, to which of them is assigned a prob-

		L	C	R
	T	0,-1	0,-4	1,0
→	M	-2,-1	0,1	1,-2
	B	-3,-4	0,-4	-4,-1
		1/6	2/3	1/6

Figure 5.30: Lower bounds game.

ability to be pulled equal to the probability with which they are played by player 2. Assume that the algorithm selects and pulls the strategy profile (M, C). Then, considering the second player, the algorithm selects the strategy, among the ones played with positive probability by player 2 in the equilibrium (that in this case are all, so L, C and R) that is the one that, given that player 1 continue playing the strategy prescribed by the NE, has the worst lower bound assuming that player 2 plays a pure strategy. Recall that, since $\boldsymbol{x}$ has been selected to be an NE in the estimated game, the estimated values of the mixed strategy profiles in which player 2 player a pure strategy among L, C and R and player 1 continue playing the mixed strategy prescribed by $\boldsymbol{x}$ results always to be equal. More precisely:

- The expected value of player 2 for playing the pure strategy L, looking at the estimated utilities, is: $\mathbb{E}[L] = 0 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} - 1 \cdot 0 = \frac{1}{2}$, while looking at the lower bounds is: $\mathbb{E}[L] = -1 \cdot \frac{1}{2} + -1 \cdot \frac{1}{2} - 4 \cdot 0 = -\frac{2}{2}$;

- The expected value of player 2 for playing the pure strategy C, looking at the estimated utilities, is: $\mathbb{E}[C] = -2 \cdot \frac{1}{2} + 3 \cdot \frac{1}{2} + -1 \cdot 0 = \frac{1}{2}$, while looking at the lower bounds is: $\mathbb{E}[C] = -4 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} - 4 \cdot 0 = -\frac{3}{2}$;

- The expected value of player 2 for playing the pure strategy R, looking at the estimated utilities, is: $\mathbb{E}[R] = 1 \cdot \frac{1}{2} + 0 \cdot \frac{1}{2} + 2 \cdot 0 = \frac{1}{2}$, while looking at the lower bounds is: $\mathbb{E}[R] = 0 \cdot \frac{1}{2} - 2 \cdot \frac{1}{2} - 1 \cdot 0 = -\frac{2}{2}$.

As we said, the three values considering the estimated utilities are the same, but looking at the lower bounds are different. In particular, we have that $\mathbb{E}[C]$ is smaller $\mathbb{E}[L]$ and $\mathbb{E}[R]$ and so the algorithm, according to Equation (5.15), selects strategy C. At this point, we have 2 candidates strategy profiles, those reported in yellow in Figure 5.31, to which of them is assigned

		↓		
	L	C	R	
T	0,-1	0,-4	1,0	1/2
M	-2,-1	0,1	1,-2	1/2
B	-3,-4	0,-4	-4,-1	0

Figure 5.31: Lower bounds game.

a probability to be pulled equal to the probability with which they are played by player 1. Assume that the algorithm selects and pulls the strategy profile (T, C). At this point, as we expected, we have pulled 2 strategy profiles. Now we move to the samples prescribed by the second querying rule, the ones selected by Equation (5.16), that are the strategy profiles that represent the best pure deviations, looking at the upper bounds, of each player from the NE $\boldsymbol{x}$, assuming that the other player continue playing the mixed strategy prescribed by $\boldsymbol{x}$. Notice that, in this phase, we can select any of the strategy of each player, not only the ones played by him/her with positive probability in $\boldsymbol{x}$. Considering player 1, we have that:

- The expected value of player 1 for playing the pure strategy T, looking at the upper bounds, is: $\mathbb{E}[T] = 2 \cdot \frac{1}{6} + 4 \cdot \frac{2}{3} + 4 \cdot \frac{1}{6} = \frac{7}{2}$;

- The expected value of player 1 for playing the pure strategy M, looking at the upper bounds, is: $\mathbb{E}[M] = 2 \cdot \frac{1}{6} + 3 \cdot \frac{2}{3} + 5 \cdot \frac{1}{6} = \frac{19}{6}$;

- The expected value of player 1 for playing the pure strategy B, looking at the upper bounds, is: $\mathbb{E}[B] = 3 \cdot \frac{1}{6} + 6 \cdot \frac{2}{3} + 2 \cdot \frac{1}{6} = \frac{29}{6}$.

So, we have that the bigger expected value, looking at the upper bounds, is the one provided by strategy B and so the algorithm, according to Equation (5.16), selects it. At this point, we have 3 candidates strategy profiles, those reported in light blue in Figure 5.32, to which of them is assigned

		L	C	R
	T	2,1	4,0	3,2
	M	2,3	3,2	5,2
→	B	3,2	6,2	2,5
		1/6	2/3	1/6

Figure 5.32: Upper bounds game.

a probability of being pulled equal to the probability with which they are played by player 2. Assume that the algorithm selects and pulls the strategy profile (B, L). Considering player 2, we have that:

- The expected value of player 2 for playing the pure strategy L, looking at the upper bounds, is: $\mathbb{E}[L] = 1 \cdot \frac{1}{2} + 3 \cdot \frac{1}{2} + 2 \cdot 0 = \frac{4}{2}$;
- The expected value of player 2 for playing the pure strategy C, looking at the upper bounds, is: $\mathbb{E}[C] = 0 \cdot \frac{1}{2} + 2 \cdot \frac{1}{2} + 2 \cdot 0 = \frac{2}{2}$;
- The expected value of player 2 for playing the pure strategy R, looking at the upper bounds, is: $\mathbb{E}[R] = 2 \cdot \frac{1}{2} + 2 \cdot \frac{1}{2} + 5 \cdot 0 = \frac{4}{2}$.

So, we have that the bigger expected values, looking at the upper bounds, are the one provided by strategies L and R and so the algorithm, according to Equation (5.16), selects one between them. Assume it selects R, at this point, we have 2 candidates strategy profiles, those reported in yellow in Figure 5.33, to which of them is assigned a probability of being pulled equal

			↓	
	L	C	R	
T	2,1	4,0	3,2	1/2
M	2,3	3,2	5,2	1/2
B	3,2	6,2	2,5	0

Figure 5.33: Upper bounds game.

to the probability with which they are played by player 1. Assume that the

algorithm selects and pulls the strategy profile (M, R). At this point, as we expected, we have pulled other 2 strategy profiles, for a total of $2|N| = 4$ strategy profiles per round. Finally, the algorithm updates the bounds and check if the condition of Equation (5.18) is satisfied. We do not report here the computation since it is trivial but a little bit long and not particularly interesting.

Example 2: pure Nash case

We would also like to briefly show that, in the case in which the NE found is pure and not mixed, the algorithm works almost in the same way as our PNE-LUCB algorithm, except for the fact that the NE is pulled not 1 but $|N|$ times. In order to facilitate the comparison between the two, we use the same setting we used in the example relative to PNE-LUCB. So, we have that at a certain time step t, the utilities of the estimated game, denoted with $\hat{\boldsymbol{u}}^t(\boldsymbol{s})$, are the ones reported in Figure 5.34, while the lower and upper

	L	C	R
T	4,5	0,1	0,2
M	0,0	1,1	1,2
B	1,1	0,2	0,1

Figure 5.34: Estimated game.

bounds are, for simplicity, for every strategy profile $\boldsymbol{s} \in S$, $\hat{\boldsymbol{u}}^t(\boldsymbol{s}) \pm 2$. We can represent them as two games, whose utilities are defined by the values of the lower an upper bounds for each strategy profile $\boldsymbol{s} \in S$, as they are reported in Figure 5.35 and Figure 5.36 respectively. Firstly, our algorithm computes

	L	C	R
T	2,3	-2,-1	-2,0
M	-2,-2	-1,-1	-1,0
B	-1,-1	-2,0	-2,-1

Figure 5.35: Lower bounds game.

	L	C	R
T	6,7	2,3	2,4
M	2,2	3,3	3,4
B	3,3	2,4	2,3

Figure 5.36: Upper bounds game.

one mixed Nash equilibrium of the estimated game, that in our case would result to be pure, and that is (T, L). We report it also with the notation that we used for the mixed NE in Figure 5.37. At this point, considering one player at a time, the algorithm should select the strategy, among the ones played with positive probability by the players in the equilibrium, that is the one that, given that the other player continue playing the strategy prescribed

	L	C	R	
T	4,5	0,1	0,2	1
M	0,0	1,1	1,2	0
B	1,1	0,2	0,1	0
	1	0	0	

Figure 5.37: Estimated game.

by the NE, has the worst lower bound assuming the the considered player plays a pure strategy. Since the equilibrium is pure, both the players have only one possibility, that is to select (M, R), that as a consequence is pulled 2 times. Moving to the second type of sample, we need to look for the strategy profiles that represent the best pure deviations, looking at the upper bounds, of each player from the NE $\boldsymbol{x}$, assuming that the other player continue playing the same strategy prescribed by $\boldsymbol{x}$. Since both the players are playing pure strategies we should look at Equation (5.17). So, following the same steps that we described in the previous example, considering player 1, we have that:

- The expected value of player 1 for playing the pure strategy T, looking at the upper bounds, is: $\mathbb{E}[T] = 6 \cdot 1 + 2 \cdot 0 + 2 \cdot 0 = 6$;
- The expected value of player 1 for playing the pure strategy M, looking at the upper bounds, is: $\mathbb{E}[M] = 2 \cdot 1 + 3 \cdot 0 + 3 \cdot 0 = 2$;
- The expected value of player 1 for playing the pure strategy B, looking at the upper bounds, is: $\mathbb{E}[B] = 3 \cdot 1 + 2 \cdot 0 + 2 \cdot 1 = 3$.

So, recalling that we can not select the strategy T because it is the one that player 1 played in the NE, we have that the bigger expected value, looking at the upper bounds, is the one provided by strategy B and so the algorithm, according to Equation (5.17), selects it. At this point, differently from the previous example, since also the strategy of player 2 is pure, we have only one candidate strategy profile, that is the one reported in light blue in Figure 5.38, that clearly gets a probability of being pulled equal to 1 and so the algorithm pulls the strategy profile (B, L) (exactly as PNE-LUCB). Then, considering player 2, we have that:

- The expected value of player 2 for playing the pure strategy L, looking at the upper bounds, is: $\mathbb{E}[L] = 7 \cdot 1 + 2 \cdot 0 + 3 \cdot 0 = 7$;
- The expected value of player 2 for playing the pure strategy C, looking at the upper bounds, is: $\mathbb{E}[C] = 3 \cdot 1 + 3 \cdot 0 + 4 \cdot 0 = 3$;

		L	C	R
	T	6,7	2,3	2,4
	M	2,2	3,3	3,4
→	B	3,3	2,4	2,3
		1	0	0

Figure 5.38: Upper bounds game.

- The expected value of player 2 for playing the pure strategy R, looking at the upper bounds, is: $\mathbb{E}[R] = 4 \cdot 1 + 4 \cdot 0 + 3 \cdot 0 = 4$.

So, recalling that we can not select the strategy L because it is the one that player 2 played in the NE, we have that the bigger expected values, looking at the upper bounds, is the one provided by strategy R and so the algorithm, according to Equation (5.17), selects it. At this point, differently from the previous example, since also the strategy of player 1 is pure, we have only one candidate strategy profile, that is the one reported in yellow in Figure 5.39, that clearly gets a probability of being pulled equal to 1 and

			↓	
	L	C	R	
T	6,7	2,3	2,4	1
M	2,2	3,3	3,4	0
B	3,3	2,4	2,3	0

Figure 5.39: Upper bounds game.

so the algorithm pulls the strategy profile (T, R) (exactly as PNE-LUCB).

5.5 SE

We propose one algorithm to learn pure Nash equilibria in the fixed-budget setting: SE (Successive Elimination). This is inspired by the SR (Successive Rejects) algorithm proposed by Audibert et al. [1], which is based on the idea of iteratively removing arms (that in our case are represented by strategy profiles) that are probably not the best (that in our case means that they are not NEs) until only the desired solutions remain (the NEs). What characterizes our algorithm with respect to the original one is the elimination rule, that is specifically defined for the problem of identifying, if they exist, pure NEs in *n-player general-sum games*, and the way in which we define the set of arms.

The basic idea of our algorithm is similar to the one of LUCB-NG, that is, we want to solve, for every player $i \in N$, a set of sub-problems in which he/she has to identify the best pure strategy he/she can play against any possible strategy profile of the other players $\boldsymbol{s}_{-i} \in S_{-i}$, namely $s_i^*(\boldsymbol{s}_{-i})$. Also in this case, we do not have the true utility function $u_i(\boldsymbol{s})$ for every player and for every strategy profile, but only an estimate of it, $\hat{u}_i(\boldsymbol{s})$ and so, as a consequence, we look for an estimate of $s_i^*(\boldsymbol{s}_{-i})$, namely $\hat{s}_i^*(\boldsymbol{s}_{-i})$. However, the way in which we proceed with the selection of $\hat{s}_i^*(\boldsymbol{s}_{-i})$ is not based on lower and upper bounds, as it was for the algorithm in fixed-confidence setting, but the best strategy that a player can play against a strategy profile of the others is the last that survives to a process of successive eliminations based on the utilities estimated during the rounds of the algorithm and that we better explain in the following. Anyway, the goal is to identify, for every player $i \in N$, the set of strategy profiles that represent the selected best responses to all the possible strategy profiles of the other players. More precisely, for every player $i \in N$, we want to find:

$$A_i^* \coloneqq \{(\hat{s}_i^*(\boldsymbol{s}_{-i}), \boldsymbol{s}_{-i}) \quad \text{for every} \quad \boldsymbol{s}_{-i} \in S_{-i}\}. \tag{5.19}$$

Notice that the cardinality of A_i^*, for each player $i \in N$, is equal to the number of possible strategy profiles of the other players. At the end, we are able to say that the pure NEs in the game are those strategy profiles that can be found in the set A_i^* of every player $i \in N$, which means that they have been selected by each player to be a best response to a strategy profile of the others. More precisely, a strategy profile $\boldsymbol{s} \in S$ is a Nash equilibria if:

$$\boldsymbol{s} \in A_i^* \quad \text{for every} \quad i \in N. \tag{5.20}$$

At this point, we can explain why we said that our definition of the set of arms was particular and specific for this approach: in the previous algorithms we proposed in this work the concept of arm perfectly coincided with the one of strategy profile, in this one instead they are strongly correlated but not coincident. More precisely, we consider each strategy profile $\boldsymbol{s} \in S$ as it corresponds not to one but to n distinct arms, one for every player $i \in N$, namely $a_i(\boldsymbol{s})$. This way, assuming for simplicity that every player $i \in N$ has the same number of available strategies $|S_i| = k$, so that $|S| = k^n$, we have n distinct sets of arms, one for every player $i \in N$, let us call it A_i, with $|A_i| = k^n$, for a total of nk^n arms. This happens because, as we anticipated before, each player $i \in N$ has to solve his/her own set of sub-problems in order to find A_i^* and this is done through a successive elimination of the non-optimal arms until only the best one remains. Since

in this part of the algorithm the problems of each player are uncorrelated with respect to the problems of the others, a strategy profile (an arm) that has been eliminated by a player $i \in N$, could have not been eliminated by another player $j \in N, j \neq i$, or it could also be the solution of one of the sub-problem of player j, and so we can not globally eliminate a strategy profile but we can only remove it from the strategy profiles *seen* from the point of view of player i. For this reason, we model this fact assigning to each player his/her own set of arms, so that he/she can remove whatever he/she wants without influencing the other players.

The algorithm works by splitting the available time (the rounds T) into $P - k^{n-1}$ phases, where $P = k^n$. Notice that P represents the number of arms available to every player at the beginning of the algorithm, while k^{n-1} is equal to the number of arms that we want to have for every player at the end of the algorithm, that is it corresponds to the number of strategy profiles that we can obtain combining $n-1$ players with k strategies per player, that means the other players excluding the considered one, and so the number of best responses that we want to find for every player. At the end of each phase p, the algorithm discards, from the set of each player, A_i^p, the arm $a_i(\boldsymbol{s})$ that corresponds to the strategy profile $\boldsymbol{s}$ that has the lowest chance of being the best response to a strategy profile $\boldsymbol{s}_{-i} \in S_{-i}$ of the other players. Notice that, in fact, $P - k^{n-1}$ corresponds exactly to the difference between what we have and what we would like to have, that is it represents the number of strategy profiles (arms) that we want to remove from the set of each player in order to have the desired number of best responses. This way, assuming for example that we are searching the NEs in a game with 2 players and 3 actions per player, we divide the rounds into $3^2 - 3^{2-1} = 9 - 3 = 6$ phases, so that, at the end of each phase, we remove 1 arm from the set of each player, for a total of 6 arms removed for every player, remaining with 3 arms per player, corresponding exactly to the 3 best responses to the 3 strategies of the other player. Specifically, let us assume that the arms of each player can be divided into k^{n-1} subsets, one for each sub-problem that must be solved, containing the arms corresponding to the strategy profiles available to be selected as a best response to a strategy profiles of the others $\boldsymbol{s}_{-i} \in S_{-i}$. More precisely, for every player $i \in N$, for every strategy profile of the other players $\boldsymbol{s}_{-i} \in S_{-i}$, we can imagine to have:

$$A_i^p(\boldsymbol{s}_{-i}) \coloneqq \{a_i(\boldsymbol{s}) \in A_i^p | \boldsymbol{s} = (\cdot, \boldsymbol{s}_{-i})\}, \tag{5.21}$$

where A_i^p is the set of remaining arms for player $i \in N$ during phase p. At this point, the algorithm assigns to each arm $a_i(\boldsymbol{s}) \in A_i^p$, with $a_i(\boldsymbol{s})$ corresponding to the strategy profile $\boldsymbol{s} = (\cdot, \boldsymbol{s}_{-i})$, such that $|A_i^p(\boldsymbol{s}_{-i})| > 1$

(because otherwise it would mean that the sub-problem of finding a best response to $\boldsymbol{s}_{-i}$ has already been solved) a parameter $\Delta_i^p(a_i(\boldsymbol{s}))$ as follows:

$$\Delta_i^p(a_i(\boldsymbol{s})) \coloneqq \hat{u}_i^p(a_i(\boldsymbol{s})) - \max_{a_i(\boldsymbol{s}') \in A_i^p(\boldsymbol{s}_{-i})} \hat{u}_i^p(a_i(\boldsymbol{s}')), \tag{5.22}$$

where $\hat{u}_i^p(a_i(\boldsymbol{s}))$ represents an estimate for the value of the strategy profile corresponding to the arm $a_i(\boldsymbol{s})$, $u_i(\boldsymbol{s})$, built using the samples collected until the end of phase p. Notice that $\Delta_i^p(a_i(\boldsymbol{s}))$ represents the difference between the utility of the current arm and the utility of the best available arm for the same sub-problem, so if $\Delta_i^p(a_i(\boldsymbol{s})) < 0$ it means that there is another arm that is better for player i than the one we are considering, while if $\Delta_i^p(a_i(\boldsymbol{s})) == 0$ then it means that, currently, the arm that we are considering is the best for its sub-problem. Then, among all the available strategy profiles for which $\Delta_i^p(a_i(\boldsymbol{s}))$ has been computed, the algorithm selects the arm $a_i(\boldsymbol{s})$ to discard from the set A_i^p as follows:

$$a_i(\boldsymbol{s}) \coloneqq \min_{a_i(\boldsymbol{s}) \in A_i^p : A_i^p(\boldsymbol{s}_{-i}) > 1} \Delta_i^p(a_i(\boldsymbol{s})). \tag{5.23}$$

The idea is that when $\Delta_i^p(a_i(\boldsymbol{s})) < 0$ for an arm $a_i(\boldsymbol{s}) \in A_i^p$, then it means that for player i there is a chance to deviate to a better strategy profile with which he/she can obtain a greater utility. Then removing the arm $a_i(\boldsymbol{s}) \in A_i^p$ that has the lowest $\Delta_i^p(a_i(\boldsymbol{s}))$ means removing the strategy profile that is, according to the current information, the worst possible response for that player to a strategy profile of the others. At the end of the last phase, each A_i^p contains exactly the arms that we are looking for, that is $A_i^p \equiv A_i^*$ for every player $i \in N$. At this point, in order to identify the NE in the game, we only need to find the strategy profiles that have been selected by all the players to be a best response to some strategy profile of the other players. More precisely, at the end the algorithm outputs the set $\bar{E}$ defined as:

$$\bar{E} \coloneqq \{\boldsymbol{s} | a_i(\boldsymbol{s}) \in A_i^* \quad \text{for every} \quad i \in N\}. \tag{5.24}$$

Following Audibert et al. [1], the lenght of the phases have been chosen to obtain an optimal (up to a logarithmic factor) convergence rate. In particular, we let $\overline{log}(P) \coloneqq \frac{1}{2} + \sum_{i=2}^{P} \frac{1}{i}$ and $T_0 \coloneqq 0$, then for every phase $p \in \left\{1, \dots, P - k^{n-1}\right\}$:

$$T_p \coloneqq \left\lceil \frac{T - P}{\overline{log}(P)\,(P + 1 - p)} \right\rceil. \tag{5.25}$$

Then, during each phase p, the algorithm selects each available strategy profile in $A^p = \bigcup_{i \in N} A_i^p$ for $T_p - T_{p-1}$ rounds.

5.5.1 Pseudocode

In the following, we provide the pseudocode of the algorithm we have just described in order to better clarify its functioning. Notice that we use only one procedure, that has been already explained for the previous algorithms, so we do not report here again the description.

Algorithm 6 SE

Input: budget T
Output: $\bar{E}$

Initialize $t \leftarrow 0$, $T_0 \leftarrow 0$, $\bar{E} \leftarrow \emptyset$, $P \leftarrow k^n$
Initialize $A_i^1 \leftarrow S \quad \forall i \in N$
for $p = 1, 2, \ldots, P - k^{n-1}$ **do**
    $T_p \leftarrow \left\lceil (T - P)/(\overline{log}(P)\,(P + 1 - p)) \right\rceil$ ▷ Equation (5.25)
    for $T_p - T_{p-1}$ *times* **do**
        pullAllStrategyProfiles(A^p)
    end for
    for $i \in N$ **do**
        for $a_i(\boldsymbol{s}) \in A_i^p | A_i^p(\boldsymbol{s}_{-i}) > 1$ **do**
            $\Delta_i^p(a_i(\boldsymbol{s})) \leftarrow \hat{u}_i^p(a_i(\boldsymbol{s})) - \max_{a_i(\boldsymbol{s}') \in A_i^p(\boldsymbol{s}_{-i})} \hat{u}_i^p(a_i(\boldsymbol{s}'))$ ▷ Equation (5.22)
        end for
        $a_i(\boldsymbol{s}) \leftarrow \arg\min_{a_i(\boldsymbol{s}) \in A_i^p : A_i^p(\boldsymbol{s}_{-i}) > 1} \Delta_i^p(a_i(\boldsymbol{s}))$ ▷ Equation (5.23)
        $A_i^{p+1} \leftarrow A_i^p - a_i(\boldsymbol{s})$ ▷ Remove strategy profile $\boldsymbol{s}$ for player i
    end for
end for
$\bar{E} \coloneqq \{\boldsymbol{s} | a_i(\boldsymbol{s}) \in A_i^* \quad \text{for every} \quad i \in N\}$ ▷ Equation (5.24)
return $\bar{E}$

5.5.2 Example

Assume that we are running SE to learn a game and that, for simplicity, the utilities of the estimated game after the first general pull are the ones reported in Figure 5.40 and that they do not change during the example.

	L	C	R
T	0,2	4,1	2,3
M	4,3	1,1	3,2
B	3,4	2,3	0,1

Figure 5.40: Estimated game.

(Clearly it is not what happens in reality, otherwise the pulls done during the algorithm would be useless, but now we want only to show and better clarify the steps done by the algorithm to reach the final solution.) As we anticipated before, if we have a game with 2 players and 3 actions per players the rounds of the algorithm are divided into 6 phases. In the following, we show what happens during each phase.

Phase 1

First of all, the algorithm computes the $\Delta_i^p(a_i(\boldsymbol{s}))$ as prescribed by Equation (5.22), for every available arm (all at the beginning) for every player separately. The deltas of player 1 are those reported in light blue in Figure 5.41, while those of player 2 are reported in orange in Figure 5.42. Then,

	L	C	R
T	0,2 (-4)	4,1 (0)	2,3 (-1)
M	4,3 (0)	1,1 (-3)	3,2 (0)
B	3,4 (-1)	2,3 (-2)	0,1 (-3)

Figure 5.41: Arms of player 1.

	L	C	R
T	0,2 (-1)	4,1 (-2)	2,3 (0)
M	4,3 (0)	1,1 (-2)	3,2 (-1)
B	3,4 (0)	2,3 (-1)	0,1 (-3)

Figure 5.42: Arms of player 2.

always for each player separately, the algorithm selects and remove the arm that has the minimum $\Delta_i^p(\cdot)$. For player 1 the arm eliminated is the one that corresponds to the strategy profile (T, L), with a $\Delta_1^1(a_1(T, L)) = -4$, while for player 2 the arm eliminated is the one that corresponds to the strategy profile (B, R), with a $\Delta_2^1(a_2(B, R)) = -3$. In the case in which two or more arms have the same $\Delta_i^p(\cdot)$, the algorithm selects one among them at random to be removed. The arms eliminated are coloured in grey during the following phases.

Phase 2

	L	C	R
T	0,2	4,1 (0)	2,3 (-1)
M	4,3 (0)	1,1 (-3)	3,2 (0)
B	3,4 (-1)	2,3 (-2)	0,1 (-3)

Figure 5.43: Arms of player 1.

	L	C	R
T	0,2 (-1)	4,1 (-2)	2,3 (0)
M	4,3 (0)	1,1 (-2)	3,2 (-1)
B	3,4 (0)	2,3 (-1)	0,1

Figure 5.44: Arms of player 2.

For player 1, the arm eliminated at the end of this phase is the one that corresponds to the strategy profile (M, C), with a $\Delta_1^2(a_1(M, C)) = -3$, while for player 2, the arm eliminated is the one that corresponds to the strategy profile (T, C), with a $\Delta_2^2(a_2(T, C)) = -2$.

Phase 3

	L	C	R
T	0,2	4,1 (0)	2,3 (-1)
M	4,3 (0)	1,1	3,2 (0)
B	3,4 (-1)	2,3 (-2)	0,1 (-3)

Figure 5.45: Arms of player 1.

	L	C	R
T	0,2 (-1)	4,1	2,3 (0)
M	4,3 (0)	1,1 (-2)	3,2 (-1)
B	3,4 (0)	2,3 (-1)	0,1

Figure 5.46: Arms of player 2.

For player 1, the arm eliminated at the end of this phase is the one that corresponds to the strategy profile (B, R), with a $\Delta_1^3(a_1(B, R)) = -3$, while for player 2, the arm eliminated is the one that corresponds to the strategy profile (M, C), with a $\Delta_2^3(a_2(M, C)) = -2$.

Phase 4

	L	C	R
T	0,2	4,1 (0)	2,3 (-1)
M	4,3 (0)	1,1	3,2 (0)
B	3,4 (-1)	2,3 (-2)	0,1

Figure 5.47: Arms of player 1.

	L	C	R
T	0,2 (-1)	4,1	2,3 (0)
M	4,3 (0)	1,1	3,2 (-1)
B	3,4 (0)	2,3 (-1)	0,1

Figure 5.48: Arms of player 2.

For player 1, the arm eliminated at the end of this phase is the one that corresponds to the strategy profile (B, C), with a $\Delta_1^4(a_1(B, C)) = -2$, while for player 2, the arm eliminated is the one that corresponds to the strategy profile (T, L), with a $\Delta_2^4(a_2(T, L)) = -1$.

Phase 5

	L	C	R
T	0,2	4,1	2,3 (-1)
M	4,3 (0)	1,1	3,2 (0)
B	3,4 (-1)	2,3	0,1

Figure 5.49: Arms of player 1.

	L	C	R
T	0,2	4,1	2,3
M	4,3 (0)	1,1	3,2 (-1)
B	3,4 (0)	2,3 (-1)	0,1

Figure 5.50: Arms of player 2.

Notice that the arm of player 1 corresponding to the strategy profile (T, C), that is $a_1(T, C)$, is coloured in green in Figure 5.49 and its $\Delta_1^5(a_1(T, C))$ has not been computed. This happens because it is the only available response left for player 1 to the strategy C of player 2, so it is considered the best response and, as a consequence, the strategy that he/she would play in that case and so it can not be eliminated. The same happens for the arm $a_2(T, R)$ of player 2, that is the only response left to the strategy T of player 1. At this point we can imagine what will the final situation be like: in the figure of each player there will be all the cells colored in grey except for 3 of them, that will be the solution of the sub-problems that each player must solve. Anyway, for player 1, the arm eliminated at the end of this phase is the one that corresponds to the strategy profile (B, L), with a $\Delta_1^5(a_1(B, L)) = -1$, while for player 2, the arm eliminated is the one that corresponds to the strategy profile (B, C), with a $\Delta_2^5(a_2(B, C)) = -1$.

Phase 6

	L	C	R
T	0,2	4,1	2,3 (-1)
M	4,3	1,1	3,2 (0)
B	3,4	2,3	0,1

Figure 5.51: Arms of player 1.

	L	C	R
T	0,2	4,1	2,3
M	4,3 (0)	1,1	3,2 (-1)
B	3,4	2,3	0,1

Figure 5.52: Arms of player 2.

Finally, for player 1, the arm eliminated at the end of this last phase is the one that corresponds to the strategy profile (T, R), with a $\Delta_1^6(a_1(T, R)) = -1$, while for player 2, the arm eliminated is the one that corresponds to the strategy profile (M, R), with a $\Delta_2^6(a_2(M, R)) = -1$. So the final situation we obtained is the one reported in Figure 5.53 and Figure 5.54. At this point the algorithm has ended and the strategy profiles that it outputs inside the set $\bar{E}$ are the ones that have been selected by both the players, that in our

	L	C	R
T	0,2	4,1	2,3
M	4,3	1,1	3,2
B	3,4	2,3	0,1

Figure 5.53: Arms of player 1.

	L	C	R
T	0,2	4,1	2,3
M	4,3	1,1	3,2
B	3,4	2,3	0,1

Figure 5.54: Arms of player 2.

example is equivalent to say that they are colored in green. In this case, the only pure NE in the game is represented by the strategy profile (M, L) because it is the only that has been selected by both the players.

Chapter 6

Experiments

In this chapter, we report the results of the experiments we have done to verify the correctness and the performances of our algorithms. We experimentally evaluate the three algorithms designed to find pure NEs, that are PNE-LUCB, LUCB-NG, and SE, against each others on randomly generated games of different sizes and different number of players. Then, we experimentally evaluate the algorithm to find a mixed NE, MNE-LUCB, on games randomly generated with the constraint of not having any pure NEs.

6.1 Experimental setting

In what follows we describe the way in which the games have been generated to handle the different tests, the type of bounds that have been used over the utilities of the estimated games for the algorithms in the fixed-confidence setting, and the parameters that have been taken into consideration to evaluate the performances of the algorithms.

6.1.1 Data generation

The utilities of all the games considered for the experiments are generated from independent Gaussian random variables. In particular, the utility of a player $i \in N$ at a strategy profile $\boldsymbol{s} \in S$ is a sample from the distribution $\tilde{u}_i(\boldsymbol{s}) \sim \mathcal{N}(0,1)$. The results of the queries to the oracle are obtained adding to the utility of the considered strategy profile a Gaussian noise $e_t \sim \mathcal{N}(0,\lambda)$, where $\lambda = 0,05$. We analyzed games of different sizes, with a different number of players and with a different number of pure NEs. In particular, we generated games with $n = 2$ players and $|S_i| \in \{3,5,7,10,15\}$ actions per player, and games with $n = 3$ players and $|S_i| \in \{3,5\}$ actions per player. In the following we organize and comment the games dividing them

according to the number of players n, and to the number of actions available to each player i, $|S_i|$. We generated 60 independent game instances (except for the case in which $n = 2$ and $|S_i| = 15$, where after 15 games the results were statistically significant). Of these 60 games, we made sure that 20 did not have any pure NE, 20 had exactly one pure NE and 20 had two or more pure NEs (for the case in which $n = 2$ and $|S_i| = 15$ we have only 5 games per type). For each possible combination of algorithm and game instance, we averaged the results of each algorithm over 5 runs. We provided the empirical means and standard deviation of the considered performance indexes.

6.1.2 Bound definition

For the algorithms in the fixed-confidence setting that require the computation of the lower and upper bounds we have adopted Hoeffding-type confidence bounds. Given a strategy profile $\boldsymbol{s} \in S$, and the estimated utility of a player $i \in N$ at $\boldsymbol{s}$, $\hat{u}_i(\boldsymbol{s})$, the Hoeffding-type bounds are defined as:

$$L_i^t(\boldsymbol{s}) \coloneqq \hat{u}_i(\boldsymbol{s}) - \sqrt{\frac{\beta(t,\delta)}{2N_{\boldsymbol{s}}(t)}} \quad \text{and} \quad U_i^t(\boldsymbol{s}) \coloneqq \hat{u}_i(\boldsymbol{s}) + \sqrt{\frac{\beta(t,\delta)}{2N_{\boldsymbol{s}}(t)}}, \tag{6.1}$$

where $\beta(t,\delta)$ is some exploration rate that we specify in what follows, t is the current round, and $N_{\boldsymbol{s}}(t)$ is the number of the times the strategy profile $\boldsymbol{s}$ has been pulled until time t. For the algorithm PNE-LUCB and MNE-LUCB the exploration term has been set as for the M-LUCB algorithm by Garivier et al. [14]:

$$\beta(t,\delta) \coloneqq \log\Big(\frac{\log(t)+1}{\delta}\Big). \tag{6.2}$$

On the other hand, for the algorithm LUCB-NG the exploration term has been set similarly to what has been done for the LUCB-G algorithm by Zhou et al. [47]. Notice that, since in this case the approach was to consider the identification of the best response of each player $i \in N$ to each strategy profile of the other players S_{-i} as independent problems, also the bounds are computed independently from each other. This means that if a player i selects to pull a strategy profile $\boldsymbol{s}$, even if also the utility of the others players at that strategy profile is sampled and updated, they do not count this in the numbers of queries performed at strategy profile $\boldsymbol{s}$. As a consequence, in the bounds we use $t_{\boldsymbol{s}_{-i}}$ instead of the number of rounds t, where $t_{\boldsymbol{s}_{-i}}$ is the number of times player i has selected to solve the sub-problem $\boldsymbol{s}_{-i}$, and $N_{\boldsymbol{s}}(t_{\boldsymbol{s}_{-i}})$ instead of $N_{\boldsymbol{s}}(t)$, where $N_{\boldsymbol{s}}(t_{\boldsymbol{s}_{-i}})$ is the number of times arm $\boldsymbol{s}$ has been selected by player i during the resolution of the sub-problem $\boldsymbol{s}_{-i}$.

Then, the Hoeffding-type bounds of a player i over the utility of a strategy profile $\boldsymbol{s}$ becomes:

$$L_i^t(\boldsymbol{s}) \coloneqq \hat{u}_i(\boldsymbol{s}) - \sqrt{\frac{\beta'(t_{\boldsymbol{s}_{-i}}, \delta)}{2N_{\boldsymbol{s}}(t_{\boldsymbol{s}_{-i}})}} \quad \text{and} \quad U_i^t(\boldsymbol{s}) \coloneqq \hat{u}_i(\boldsymbol{s}) + \sqrt{\frac{\beta'(t_{\boldsymbol{s}_{-i}}, \delta)}{2N_{\boldsymbol{s}}(t_{\boldsymbol{s}_{-i}})}}, \quad (6.3)$$

and, denoting with m the number of actions available to each player, and with n the number of players, the exploration rate is defined as:

$$\beta'(t_{\boldsymbol{s}_{-i}}, \delta) \coloneqq \log\Big(\frac{m^n t_{\boldsymbol{s}_{-i}}^4}{4\delta}\Big). \quad (6.4)$$

6.1.3 Performance indexes

The performance indexes that are used to evaluate the algorithms are:

- $\bar{B}$: average of the number of pulls used by the algorithm to meet a termination condition (if it has not exceeded the budget B);
- $\sigma(B)$: standard deviation of the number of pulls used by the algorithm to meet a termination condition (if it has not exceeded the budget B);
- $\sigma^2(B)$: variance of the number of pulls used by the algorithm to meet a termination condition (if it has not exceeded the budget B);
- $\%limit$: Percentage of times the algorithm has exceeded the budget before meeting a termination condition;
- $\%oneNE$: Percentage of times the algorithm found at least one pure NE of the game;
- $\%allNEs$: Percentage of times the algorithm found all the pure NEs of the game;
- $\%fakeNE$: Percentage of times the algorithm returned a strategy profile that is not a pure NE of the game;
- $\bar{\epsilon}$: average of the error performed by the algorithm in the case in which a strategy profile that is not a NE of the game is returned. The error for a strategy profile $\boldsymbol{s} \in S$ returned by an algorithm is defined as:

$$\epsilon(\boldsymbol{s}) = \max_{i \in N} \max_{s_i' \in S_i} \Big(u_i(s_i', \boldsymbol{s}_{-i}) - u_i(\boldsymbol{s})\Big), \quad (6.5)$$

  where s_i' represents a deviation of player i with respect to the action he/she is playing in the strategy profile $\boldsymbol{s}$.

For instance, assume that we are learning a game with 2 players, and that the algorithm returns the strategy profile $\boldsymbol{s} \in S$ that is not a NE. Then it means that for player 1, or for player 2, or for both of them there is the possibility to change the action they are playing and to deviate to a strategy profile that guarantees them a better utility.

Notice that in those settings in which these indexed are not well defined, we did not report their values in the results. More specifically, in the case in which we look for pure NEs of games that does not have any NE the parameters $\%oneNE$ and $\%allNEs$ are not reported, in the case in which we look for a mixed NE the parameters $\%oneNE$, $\%allNEs$, and $\%fakeNE$ are not reported, and, finally, in the case in which we analyze the algorithms SE alone, the parameters $\bar{B}$, $\sigma(B)$, and $\sigma^2(B)$ are not reported since they are uniquely defined by the budget.

Finally, notice that the results reported in the tables that follow are obtained averaging the performance indexes computed for the resolution of all the games of each type, *i.e.*, combination of number of players n and number of actions per player $|S_i|$.

6.2 Results

In what follows, we report the results of the experiments, divided into four sections, according to the number of players considered and to the type of NEs searched. Unless otherwise specified, all the tests have been done requiring $\delta = 0.2$, $\epsilon = 0$, and fixing a budget of $B = 250000$ pulls. Notice that in the tables we denote with:

- *PNE-LUCB*: the PNE-LUCB algorithm that stops after finding one Nash equilibrium;
- *PNE-LUCB (All)*: the PNE-LUCB that looks for all the Nash equilibria;
- *LUCB-NG*: the LUCB-NG algorithm that stops after finding one Nash equilibrium;
- *LUCB-NG (All)*: the LUCB-NG algorithm that looks for all the Nash equilibria;
- *SE*: the SE algorithm;
- *MNE-LUCB*: the MNE-LUCB algorithm.

6.2.1 Looking for pure NEs in 2-player games

We report here the results obtained with games with $n = 2$. In Appendix A.1 we reported the results differentiating w.r.t. the number of NEs the game has.

Table 6.1: Games with $n = 2$ and $|S_i| = 3$ (Total).

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	9,746.85	26,276.18	46,836.12	76,629.02	$2.5 \cdot 10^5$
$\sigma(B)$	2,322.77	5,997	5,847.44	10,875.78	0
$\sigma^2(B)$	$5.4 \cdot 10^6$	$3.6 \cdot 10^7$	$3.42 \cdot 10^7$	$1.18 \cdot 10^8$	0
$\%limit$	11.67	18.33	29.67	66.67	0
$\%oneNE$	100	100	100	100	100
$\%allNEs$	62.26	100	58.14	100	100
$\%fakeNE$	0	0	0	0	0
$\bar{\epsilon}$	X	X	X	X	X

Table 6.2: Games with $n = 2$ and $|S_i| = 5$ (Total).

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	22,547.98	48,745.37	54,191.67	$2.3 \cdot 10^5$	$2.5 \cdot 10^5$
$\sigma(B)$	5,483.16	6,954.75	7,378.09	0	0
$\sigma^2(B)$	$3.01 \cdot 10^7$	$4.84 \cdot 10^7$	$5.44 \cdot 10^7$	0	0
$\%limit$	10	21.67	45	98.33	0
$\%oneNE$	100	100	100	100	100
$\%allNEs$	62.96	100	48.48	100	100
$\%fakeNE$	0	0	0	0	1
$\bar{\epsilon}$	X	X	X	X	$5.87 \cdot 10^{-4}$

Table 6.3: Games with $n = 2$ and $|S_i| = 7$ (Total).

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	41,683.98	75,688.75	95,641.33	X	$2.5 \cdot 10^5$
$\sigma(B)$	7,346.54	9,777.85	13,277.94	X	0
$\sigma^2(B)$	$5.4 \cdot 10^7$	$9.56 \cdot 10^7$	$1.76 \cdot 10^8$	X	0
$\%limit$	16.67	38.33	61.67	100	0
$\%oneNE$	100	100	100	X	100
$\%allNEs$	60	100	30.43	X	100
$\%fakeNE$	0	0	0	X	0.33
$\bar{\epsilon}$	X	X	X	X	$2.12 \cdot 10^{-3}$

Table 6.4: Games with $n = 2$ and $|S_i| = 10$ (Total).

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	45,609.17	93,162.46	$1.25 \cdot 10^5$	X	$2.5 \cdot 10^5$
$\sigma(B)$	6,977.67	10,305.42	16,554.09	X	0
$\sigma^2(B)$	$4.87 \cdot 10^7$	$1.06 \cdot 10^8$	$2.74 \cdot 10^8$	X	0
$\%limit$	25	43.33	75.17	100	0
$\%oneNE$	100	100	100	X	100
$\%allNEs$	57.78	100	33.33	X	100
$\%fakeNE$	0	0	0	X	0
$\bar{\epsilon}$	X	X	X	X	X

Table 6.5: Games with $n = 2$ and $|S_i| = 15$ (Total).

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	75,239.2	X	$1.55 \cdot 10^5$	X	$2.5 \cdot 10^5$
$\sigma(B)$	18,598.04	X	19,591.03	X	0
$\sigma^2(B)$	$3.46 \cdot 10^8$	X	$3.84 \cdot 10^8$	X	0
$\%limit$	6.67	X	70.67	X	0
$\%oneNE$	100	X	100	X	100
$\%allNEs$	64.29	X	20	X	100
$\%fakeNE$	0	X	0	X	1.33
$\bar{\epsilon}$	X	X	X	X	$5.14 \cdot 10^{-3}$

Looking at Tables 6.1-6.5, if we focus only on the algorithms in the fixed-confidence setting, it is clear that the algorithm PNE-LUCB outperforms the LUCB-NG algorithm w.r.t. any performance metric analysed: in the case in which they are looking for a single NE the mean, the standard deviation and the variance in the number of pulls are always smaller and it exceeds the budget a smaller number of times. Anyway, in the case in which both terminate, they both return a true NE the 100% of the times. Also in the case in which the two algorithms are looking for all the NEs PNE-LUCB outperforms LUCB-NG: as before, the mean, the standard deviation and the variance in the number of pulls are always smaller and it exceeds the budget a smaller number of times. Once again, in the case in which both terminate, they return all the true NE the 100% of the times. However, notice that when the number of actions increases, it happens that they exceed the given budget. In our experiments, this starts to happen the 100% of the times for LUCB-NG from $|S_i| = 7$ on, while for PNE-LUCB only when $|S_i| = 15$. Moreover, notice that none of them, in none of the cases, ever return a strategy profile that is not a NE. For what concerns the only algorithm in the fixed-budget setting, SE, it happens that it returns a strategy profile that is not a NE, but only with a probability less than 2%,

while it always returns all the true NEs. Looking at the graph provided in Figure 6.1 we can have an even clearer overview of the comparison between the algorithms PNE-LUCB and LUCB-NG. More specifically, we can see

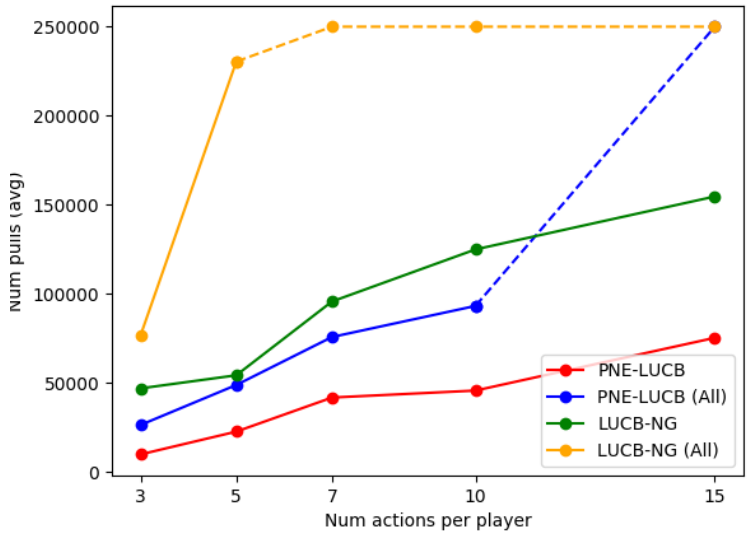

Figure 6.1: Average number of pulls with respect to the number of actions available to each player. (The dashed lines represent game dimensions for which the algorithm in question exceed the budget the 100% *of the times.)*

that both for PNE-LUCB and LUCB-NG, in the case in which they are looking for a single NE, the average number of pulls required to meet the termination condition seems to grow linearly w.r.t the number of actions per player, while, in the case in which they are looking for all the NEs in a game, the growth seems to be much faster, especially for LUCB-NG.

Since the SE algorithm is a limited-budget algorithm, and therefore it uses all the available budget, we analyze its performances as we gradually reduce it. We report in Tables 6.6 and 6.7 the results obtained only with games with $n = 2$ and $|S_i| = \{5, 10\}$, since the trends obtained with all the different dimensions are pretty similar. The complete results are reported in Appendix A.1. The performance of the SE algorithm remains excellent even when reducing the budget, in fact it finds one or all NEs more than 97% of the times, and even when the returned NE is incorrect, the corresponding ϵ is small, *i.e.*, smaller than 10^{-2}.

Table 6.6: Games with $n = 2$ and $|S_i| = 5$.

B	200000.0	150000.0	100000.0	50000.0	25000.0	12500.0	6250.0
$\%oneNE$	100	100	100	100	100	100	100
$\%allNEs$	100	100	100	100	100	100	100
$\%fakeNE$	1.33	1	1.33	2.33	1.67	0.67	1
$\bar{\epsilon}$	$5.87 \cdot 10^{-4}$	$5.87 \cdot 10^{-4}$	$5.87 \cdot 10^{-4}$	$5.87 \cdot 10^{-4}$	$5.87 \cdot 10^{-4}$	$9.59 \cdot 10^{-4}$	$5.87 \cdot 10^{-4}$

Table 6.7: Games with $n = 2$ and $|S_i| = 10$.

B	200000.0	150000.0	100000.0	50000.0	25000.0	12500.0	6250.0
$\%oneNE$	100	100	100	100	100	100	98.67
$\%allNEs$	99.33	99.67	99.33	99	99.67	99.67	97.33
$\%fakeNE$	0.33	0	0.33	0.33	1	1	1.67
$\bar{\epsilon}$	$3.86 \cdot 10^{-3}$	X	$3.86 \cdot 10^{-3}$	$8.27 \cdot 10^{-3}$	$6.06 \cdot 10^{-3}$	$8.27 \cdot 10^{-3}$	$9.27 \cdot 10^{-3}$

6.2.2 Looking for pure NEs in 3-player games

We report here the results obtained with games with $n = 3$. In Appendix A.2 we reported the results differentiating w.r.t. the number of NEs the game has. Looking at Tables 6.8 and 6.9, if we focus only on the algorithms

Table 6.8: Games with $n = 3$ and $|S_i| = 3$ (Total).

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	36,785.33	63,227.73	$1.12 \cdot 10^5$	X	$2.5 \cdot 10^5$
$\sigma(B)$	6,785.76	10,101.46	14,475.22	X	0
$\sigma^2(B)$	$4.6 \cdot 10^7$	$1.02 \cdot 10^8$	$2.1 \cdot 10^8$	X	0
$\%limit$	16.67	36.67	60.33	100	0
$\%oneNE$	100	100	100	X	99.67
$\%allNEs$	60	100	38.46	X	99.67
$\%fakeNE$	0	0	0	X	0.33
$\bar{\epsilon}$	X	X	X	X	$1.69 \cdot 10^{-3}$

Table 6.9: Games with $n = 3$ and $|S_i| = 5$ (Total).

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	62,378.25	$1.29 \cdot 10^5$	$1.31 \cdot 10^5$	X	$2.5 \cdot 10^5$
$\sigma(B)$	8,781.33	9,728.79	16,557.28	X	0
$\sigma^2(B)$	$7.71 \cdot 10^7$	$9.46 \cdot 10^7$	$2.74 \cdot 10^8$	X	0
$\%limit$	25	50	75.67	100	0
$\%oneNE$	100	100	100	X	100
$\%allNEs$	60	100	47.06	X	99.33
$\%fakeNE$	0	0	0	X	0.67
$\bar{\epsilon}$	X	X	X	X	$1.75 \cdot 10^{-3}$

in the fixed-confidence setting, it is even clearer that the algorithm PNE-LUCB outperforms the LUCB-NG algorithm w.r.t. any performance metric

analyzed: in the case in which they are looking for a single NE the mean, the standard deviation and the variance in the number of pulls are always smaller and it exceeds the budget a much smaller number of times. Also in this setting, when they both terminate, then they return a true NE the 100% of the times. Notice that in the case in which they are looking for all the NEs the LUCB-NG algorithm exceeds the budget the 100% of the times, and so the comparison is not even needed. Finally, for what concerns the only algorithm in the fixed-budget setting, SE, notice that, unlike the case in which $n = 2$, in this case it happens that it does not find some true NEs, however this happens less than 1% of the times. Finally, also for these games we analyze the performances of the algorithm SE alone, gradually reducing its budget. The results are reported in Table 6.10 and 6.11. We can draw

Table 6.10: Games with $n = 3$ and $|S_i| = 3$.

B	200000.0	150000.0	100000.0	50000.0	25000.0	12500.0	6250.0
$\%oneNE$	99.67	99.67	100	99	99	99	98.33
$\%allNEs$	99.67	99.67	100	99	99	99	98.33
$\%fakeNE$	0.33	0.33	0	0.67	0.33	2.33	2
$\bar{\epsilon}$	$1.69 \cdot 10^{-3}$	$1.69 \cdot 10^{-3}$	X	$1.69 \cdot 10^{-3}$	$1.69 \cdot 10^{-3}$	$4.86 \cdot 10^{-3}$	$4.17 \cdot 10^{-3}$

Table 6.11: Games with $n = 3$ and $|S_i| = 5$.

B	200000.0	150000.0	100000.0	50000.0	25000.0	12500.0	6250.0
$\%oneNE$	100	100	100	100	100	100	99.67
$\%allNEs$	99.67	99.67	99.33	99.33	99	99.33	98.33
$\%fakeNE$	0.67	1.33	1	1.67	1.67	2	2
$\bar{\epsilon}$	$1.75 \cdot 10^{-3}$	$1.75 \cdot 10^{-3}$	$5.4 \cdot 10^{-3}$	$5.29 \cdot 10^{-3}$	$5.29 \cdot 10^{-3}$	$3.57 \cdot 10^{-3}$	$5.29 \cdot 10^{-3}$

the same conclusions that we drew for the previous case when $n = 2$: even when reducing the budget, the performance of the algorithm remains good, it always returns one or all the NEs more than the 98% of the times, and even when the returned NE is incorrect, the corresponding ϵ is smaller, *i.e.*, smaller than 10^{-2}.

6.2.3 Looking for a mixed NE

We report here the results obtained both with games with $n = 2$ and $n = 3$ obtained by the algorithm MNE-LUCB. Notice that the games we considered are the ones among those used for the previous tests that we are sure to not have pure NEs. Moreover, to test this algorithm, we required $\epsilon = 0.1$ and not 0, because that would have mean to pretend that the algorithm outputs a perfect mixed NE of the game, which is unfeasible having an estimate

of the utilities and not the real ones. Finally, notice that we have not compared MNE-LUCB against any other algorithm because, to the best of our knowledge, it is the only available method to find mixed NEs in simulation-based games. The results are reported in Tables 6.12 and 6.13.

We can see that, even if the requirement was only to have $\epsilon < 0.1$, the

Table 6.12: Games with $n = 2$.

$\lvert S_i \rvert$	3	5	7	10	15
$\bar{B}$	67,356.73	$1.31 \cdot 10^5$	$1.5 \cdot 10^5$	$1.51 \cdot 10^5$	$2.21 \cdot 10^5$
$\sigma(B)$	4,423.91	10,846.06	8,672.59	6,439.22	17,054
$\sigma^2(B)$	$1.96 \cdot 10^7$	$1.18 \cdot 10^8$	$7.52 \cdot 10^7$	$4.15 \cdot 10^7$	$2.91 \cdot 10^8$
$\%limit$	15	22	26	24	76
$\bar{\epsilon}$	$3.39 \cdot 10^{-4}$	$2.2 \cdot 10^{-3}$	$2.05 \cdot 10^{-4}$	$2.12 \cdot 10^{-4}$	$2.35 \cdot 10^{-4}$

Table 6.13: Games with $n = 3$.

$\lvert S_i \rvert$	3	5
$\bar{B}$	$1.63 \cdot 10^5$	$2.01 \cdot 10^5$
$\sigma(B)$	6,815.9	8,880.77
$\sigma^2(B)$	$4.65 \cdot 10^7$	$7.89 \cdot 10^7$
$\%limit$	25	43
$\bar{\epsilon}$	$1.8 \cdot 10^{-4}$	$1.44 \cdot 10^{-4}$

algorithm always return equilibria with a much lower value, *i.e.*, smaller than $3 \cdot 10^{-3}$. Moreover, for the case in which $n = 2$, up to the size of games with $|S_i| = 10$, the percentage of the times that it exceeds the budget is quite small, *i.e.*, less than 26%. It becomes higher for the case in which $|S_i| = 15$, because the dimension of the space increases a lot and so it would need an higher budget to meet the termination condition.

6.3 Comments

With regard only to the search of pure NEs in the fixed-confidence setting, PNE-LUCB proves to have better performances than LUCB-NG, especially when the goal is to find all the pure NEs of the game. This is probably due to the less strict termination condition of PNE-LUCB with respect to the one of LUCB-NG. Indeed, to fulfill the termination condition, PNE-LUCB needs to determine for every strategy profile whether it is an NE or not, while LUCB-NG needs to find the best response of every player to every strategy profile of the others, which is a more complex and resource-intensive problem. Moreover, the bounds adopted by PNE-LUCB take into

consideration all the queries performed to the simulator, while the ones adopted by LUCB-NG consider, for every sub-problem separately, only the queries performed to solve it, thus not using all the available information. In fact, when a player selects a strategy profile to be queried, the utilities of all the players at that strategy profile are sampled, but this is not taken into account. However, when we include also SE in this comparison, it results to be the best performing one, reaching an high precision even with a reduced number of available queries (budget). Finally, regarding the only algorithm designed to find a mixed NE, MNE-LUCB, we have no other algorithms against which compare it, but we could safely state that its performance are satisfactory, considering that it returns a good approximation of the true mixed NE in most of the cases when the game dimension does not increase too much.

Chapter 7

Conclusion and Future works

In this work, we presented and evaluated four algorithms to learn games for which no analytical description of the utilities was available. Indeed, simulation-based games model those settings in which only a game specified by means of a game simulator that we could query to obtain noisy estimates of them is available. However, running the simulation during the game is not a feasible solution, so we had to design algorithms able to iteratively sample players' stochastic utilities using some ad-hoc querying rules, and, then, output some strategy profiles which represent good candidates for being (approximate) equilibria. In particular, we addressed the problem of learning *Nash Equilibria* of *n-player general sum* games by introducing a *multi-armed bandit* model and considering two possible settings: *fixed-confidence* setting and *fixed-budget* setting. In the former setting we proposed 3 algorithms: PNE-LUCB and LUCB-NG, designed to find one or all the pure NEs of a game under analysis, and MNE-LUCB, designed to find a mixed NE of a game under analysis. In the latter setting, we proposed the use of the SE algorithm, designed to find all the pure NEs of a game under analysis. Finally, we experimentally evaluated all our algorithms on games with utilities generated from independent Gaussian random variables. We discovered that for the task of learning pure NE, PNE-LUCB had better performances than LUCB-NG, but, worse than SE, which turned out be the best in this task. Regarding the only algorithm to learn a mixed NE in the fixed confidence setting, we observed that it returns a good approximation of the real mixed NEs in most cases.

7.1 Future works

In future, we could extend our work along different directions. First of all, we aim to provide the theoretical analysis and to prove the δ-PAC properties of the second algorithm designed to fine one or more pure NEs in the fixed-confidence setting, LUCB-NG, of the algorithm designed to find a mixed NE in the fixed-confidence setting, MNE-LUCB, and of the algorithm designed to find all the pure NEs in the fixed-budget setting, SE. Moreover, we would like to study the possibility of applying our techniques to solve problems of *empirical mechanism design* [39]. Finally, we could think about tackling the problem of learning types of solution concepts different from the one of Nash Equilibria, *i.e.*, we could try and design algorithms to learn *Stackelberg Equilibria* in *n-players general-sum* games.

Bibliography

[1] Jean-Yves Audibert, Sébastien Bubeck, and Remi Munos. Best arm identification in multi-armed bandits. In *COLT 2010 - The 23rd Conference on Learning Theory*, pages 41–53, 11 2010.

[2] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256, May 2002. ISSN 0885-6125. doi: 10.1023/A:1013689704352.

[3] Nicola Basilico and Nicola Gatti. Strategic guard placement for optimal response toalarms in security games. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1481–1482. Citeseer, 2014.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

[5] L. Bisi, G. De Nittis, F. Trovò, M. Restelli, and N. Gatti. Regret minimization algorithms for the followers behaviour identification in leadership games. In *Proceedings of the Conference on Uncertainty in Artificial (UAI)*, pages 1–10, 2017.

[6] Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration for multi-armed bandit problems, 2010.

[7] Neil Burch, Martin Schmid, Matej Moravcik, Dustin Morrill, and Michael Bowling. Aivat: A new variance reduction technique for agent evaluation in imperfect information games. In *AAAI*, pages 949–956, 2018.

[8] Matteo Castiglioni, Alberto Marchesi, and Nicola Gatti. Be a leader or become a follower: The strategy to commit to with multiple leaders. In *IJCAI*, pages 123–129, 2019.

[9] Matteo Castiglioni, Alberto Marchesi, Nicola Gatti, and Stefano Coniglio. Leadership in singleton congestion games: What is hard and what is easy. *Artificial Intelligence*, 277:103177, 2019.

[10] Matteo Castiglioni, Andrea Celli, Alberto Marchesi, and Nicola Gatti. Online bayesian persuasion. *Advances in Neural Information Processing Systems*, 33, 2020.

[11] Sofia Ceppi, Nicola Gatti, Giorgio Patrini, and Marco Rocco. Local search techniques for computing equilibria in two-player general-sum strategic-form games. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 1469–1470, 2010.

[12] Stefano Coniglio, Nicola Gatti, and Alberto Marchesi. Computing a pessimistic stackelberg equilibrium with multiple followers: The mixed-pure case. *Algorithmica*, 82(5):1189–1238, 2020.

[13] Gabriele Farina, Andrea Celli, Nicola Gatti, and Tuomas Sandholm. Ex ante coordination and collusion in zero-sum multi-player extensive-form games. In *Conference on Neural Information Processing Systems (NIPS)*, 2018.

[14] Aurélien Garivier, Emilie Kaufmann, and Wouter M. Koolen. Maximin action identification: A new bandit framework for games. *ArXiv*, abs/1602.04676, 2016.

[15] N. Gatti and Marcello Restelli. Equilibrium approximation in simulation-based extensive-form games. In *AAMAS*, 2011.

[16] Nicola Gatti, Alessandro Lazaric, Marco Rocco, and Francesco Trovò. Truthful learning mechanisms for multi-slot sponsored search auctions with externalities. *ARTIF INTELL*, 227:93–139, 2015.

[17] Shivaram Kalyanakrishnan, Ambuj Tewari, P. Auer, and P. Stone. Pac subset selection in stochastic multi-armed bandits. In *ICML*, 2012.

[18] Emilie Kaufmann, Olivier Cappé, and Aurélien Garivier. On the complexity of best-arm identification in multi-armed bandit models. *Journal of Machine Learning Research*, 17(1):1–42, 2016.

[19] T.L Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22, 1985. ISSN 0196-8858. doi: https://doi.org/10.1016/0196-8858(85)90002-8.

[20] Shuai Li, Baoxiang Wang, Shengyu Zhang, and Wei Chen. Contextual combinatorial cascading bandits. In *International conference on machine learning*, pages 1245–1253. PMLR, 2016.

[21] A. Marchesi, F. Trovò, and N. Gatti. Learning probably approximately correct maximin strategies in simulation-based games with infinite strategy spaces. In *AAMAS*, 2020.

[22] Alberto Marchesi, Matteo Castiglioni, and Nicola Gatti. Leadership in congestion games: Multiple user classes and non-singleton actions. In *IJCAI*, pages 485–491, 2019.

[23] Alberto Marchesi, Gabriele Farina, Christian Kroer, Nicola Gatti, and Tuomas Sandholm. Quasi-perfect stackelberg equilibrium. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2117–2124, 2019.

[24] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997. ISBN 978-0-07-042807-2.

[25] John Nash. Non-cooperative games. *Annals of mathematics*, pages 286–295, 1951.

[26] John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950. ISSN 0027-8424.

[27] Alessandro Nuara, Francesco Trovo, Nicola Gatti, and Marcello Restelli. A combinatorial-bandit algorithm for the online joint bid/budget optimization of pay-per-click advertising campaigns. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[28] Alessandro Nuara, Francesco Trovò, Dominic Crippa, Nicola Gatti, and Marcello Restelli. Driving exploration by maximum distribution in gaussian process bandits. In *International Conference on Autonomous Agents and MultiAgent Systems*, pages 948–956, 2020.

[29] Alessandro Nuara, Francesco Trovò, Nicola Gatti, and Marcello Restelli. Online joint bid/daily budget optimization of internet advertising campaigns. *arXiv preprint arXiv:2003.01452*, 2020.

[30] David Salomon. *Data Compression: The Complete Reference*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 1846286026.

[31] Samuel Sokota, Caleb Ho, and Bryce Wiedenbeck. Learning deviation payoffs in simulation-based games. In *AAAI*, 2019.

[32] N. Srinivas, A. Krause, S.M. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1015–1022, 2010.

[33] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

[34] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA, 2005. ISBN 0321321367.

[35] Francesco Trovò, Stefano Paladino, Marcello Restelli, and Nicola Gatti. Improving multi-armed bandit algorithms in online pricing settings. *International Journal of Approximate Reasoning*, 98:196–235, 2018.

[36] Francesco Trovo, Stefano Paladino, Marcello Restelli, and Nicola Gatti. Sliding-window thompson sampling for non-stationary settings. *Journal of Artificial Intelligence Research*, 68:311–364, 2020.

[37] Leslie Valiant. *Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World*. Basic Books, Inc., 2013.

[38] Enrique Areyan Viqueira, Cyrus Cousins, Eli Upfal, and Amy Greenwald. Learning equilibria of simulation-based games. *ArXiv*, abs/1905.13379, 2019.

[39] Enrique Areyan Viqueira, Cyrus Cousins, Yasser Mohammad, and Amy Greenwald. Empirical mechanism design: Designing mechanisms from data. In Ryan P. Adams and Vibhav Gogate, editors, *Proceedings of Machine Learning Research*, volume 115, pages 1094–1104, Tel Aviv, Israel, 22–25 Jul 2020. PMLR.

[40] H. von Stackelberg. *Marktform und Gleichgewicht*. Springer, Vienna, 1934.

[41] Bernhard von Stengel. Computing equilibria for two-person games. In R.J. Aumann and S. Hart, editors, *Handbook of Game Theory with Economic Applications*, volume 3, chapter 45, pages 1723–1759. Elsevier, 1 edition, 2002.

[42] Y. Vorobeychik and M. Wellman. Stochastic search methods for nash equilibrium approximation in simulation-based games. In *AAMAS*, pages 1055–1062, 2008.

[43] Y. Vorobeychik, Michael P. Wellman, and Satinder Singh. Learning payoff functions in infinite games. *Machine Learning*, 67:145–168, 2007.

[44] W. Walsh, D. Parkes, and R. Das. Choosing samples to compute heuristic-strategy nash equilibrium. In *AMEC*, 2003.

[45] B. Wiedenbeck, Fengjun Yang, and Michael P. Wellman. A regression approach for modeling games with many symmetric players. In *AAAI*, 2018.

[46] C.K.I. Williams and C.E. Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[47] Yichi Zhou, J. Li, and J. Zhu. Identify the nash equilibrium in static games with random payoffs. In *ICML*, 2017.

Appendix A

Additional experimental results

A.1 Looking for pure NEs in 2-player games

Table A.1: Games with $n = 2$, $|S_i| = 3$ and no Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	16,010.33	16,010.33	54,214.6	54,214.6	$2.5 \cdot 10^5$
$\sigma(B)$	6,913.67	6,913.67	9,165.49	9,165.49	0
$\sigma^2(B)$	$4.78 \cdot 10^7$	$4.78 \cdot 10^7$	$8.4 \cdot 10^7$	$8.4 \cdot 10^7$	0
$\%limit$	30	30	60	60	0
$\%fakeNE$	0	0	0	0	0
$\bar{\epsilon}$	X	X	X	X	X

Table A.2: Games with $n = 2$, $|S_i| = 3$ and one Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	8,938.54	30,050.34	57,497.45	95,881.85	$2.5 \cdot 10^5$
$\sigma(B)$	2,962.39	11,913.32	10,316.11	21,517.74	0
$\sigma^2(B)$	$8.78 \cdot 10^6$	$1.42 \cdot 10^8$	$1.06 \cdot 10^8$	$4.63 \cdot 10^8$	0
$\%limit$	5	15	15	60	0
$\%oneNE$	100	100	100	100	100
$\%fakeNE$	0	0	0	0	0
$\bar{\epsilon}$	X	X	X	X	X

Table A.3: Games with $n = 2$, $|S_i| = 3$ and more than one Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	6,130.3	30,696.23	33,487.77	82,952.2	$2.5 \cdot 10^5$
$\sigma(B)$	2,118.62	10,228.07	8,148.81	17,957.07	0
$\sigma^2(B)$	$4.49 \cdot 10^6$	$1.05 \cdot 10^8$	$6.64 \cdot 10^7$	$3.22 \cdot 10^8$	0
$\%limit$	0	10	14	80	0
$\%oneNE$	100	100	100	100	100
$\%allNEs$	0	100	0	100	100
$\%fakeNE$	0	0	0	0	0
$\bar{\epsilon}$	X	X	X	X	X

Table A.4: Games with $n = 2$, $|S_i| = 5$ and no Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	52,850.92	52,850.92	X	X	$2.5 \cdot 10^5$
$\sigma(B)$	15,126.22	15,126.22	X	X	0
$\sigma^2(B)$	$2.29 \cdot 10^8$	$2.29 \cdot 10^8$	X	X	0
$\%limit$	20	20	100	100	0
$\%fakeNE$	0	0	X	X	3
$\bar{\epsilon}$	X	X	X	X	$5.87 \cdot 10^{-4}$

Table A.5: Games with $n = 2$, $|S_i| = 5$ and one Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	8,618.02	36,414.09	53,747.06	X	$2.5 \cdot 10^5$
$\sigma(B)$	3,189.37	10,266.95	10,503.39	X	0
$\sigma^2(B)$	$1.02 \cdot 10^7$	$1.05 \cdot 10^8$	$1.1 \cdot 10^8$	X	0
$\%limit$	10	15	20	100	0
$\%oneNE$	100	100	100	X	100
$\%fakeNE$	0	0	0	X	0
$\bar{\epsilon}$	X	X	X	X	X

Table A.6: Games with $n = 2$, $|S_i| = 5$ and more than one Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	10,842.6	59,027	54,610.13	$2.3 \cdot 10^5$	$2.5 \cdot 10^5$
$\sigma(B)$	3,491.09	8,291.37	10,362.4	0	0
$\sigma^2(B)$	$1.22 \cdot 10^7$	$6.87 \cdot 10^7$	$1.07 \cdot 10^8$	0	0
$\%limit$	0	30	15	95	0
$\%oneNE$	100	100	100	100	100
$\%allNEs$	0	100	0	100	100
$\%fakeNE$	0	0	0	0	0
$\bar{\epsilon}$	X	X	X	X	X

Table A.7: Games with $n = 2$, $|S_i| = 7$ and no Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	50,727.61	50,727.61	X	X	$2.5 \cdot 10^5$
$\sigma(B)$	9,327.07	9,327.07	X	X	0
$\sigma^2(B)$	$8.7 \cdot 10^7$	$8.7 \cdot 10^7$	X	X	0
$\%limit$	30	30	100	100	0
$\%fakeNE$	0	0	X	X	1
$\bar{\epsilon}$	X	X	X	X	$2.12 \cdot 10^{-3}$

Table A.8: Games with $n = 2$, $|S_i| = 7$ and one Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	65,641.76	91,350.29	66,760.66	X	$2.5 \cdot 10^5$
$\sigma(B)$	18,624.02	18,020	28,953.16	X	0
$\sigma^2(B)$	$3.47 \cdot 10^8$	$3.25 \cdot 10^8$	$8.38 \cdot 10^8$	X	0
$\%limit$	20	30	65	100	0
$\%oneNE$	100	100	100	X	100
$\%fakeNE$	0	0	0	X	0
$\bar{\epsilon}$	X	X	X	X	X

Table A.9: Games with $n = 2$, $|S_i| = 7$ and more than one Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	16,187.23	90,154.8	$1.08 \cdot 10^5$	X	$2.5 \cdot 10^5$
$\sigma(B)$	3,792.49	21,125.36	13,079.62	X	0
$\sigma^2(B)$	$1.44 \cdot 10^7$	$4.46 \cdot 10^8$	$1.71 \cdot 10^8$	X	0
$\%limit$	0	55	20	100	0
$\%oneNE$	100	100	100	X	100
$\%allNEs$	0	100	0	X	100
$\%fakeNE$	0	0	0	X	0
$\bar{\epsilon}$	X	X	X	X	X

Table A.10: Games with $n = 2$, $|S_i| = 10$ and no Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	79,464.85	79,464.85	X	X	$2.5 \cdot 10^5$
$\sigma(B)$	16,887.7	16,887.7	X	X	0
$\sigma^2(B)$	$2.85 \cdot 10^8$	$2.85 \cdot 10^8$	X	X	0
$\%limit$	45	45	100	100	0
$\%fakeNE$	0	0	X	X	0
$\bar{\epsilon}$	X	X	X	X	X

Table A.11: Games with $n = 2$, $|S_i| = 10$ and one Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	47,447.64	$1.03 \cdot 10^5$	95,303.1	X	$2.5 \cdot 10^5$
$\sigma(B)$	12,956.33	19,611.46	25,618.73	X	0
$\sigma^2(B)$	$1.68 \cdot 10^8$	$3.85 \cdot 10^8$	$6.56 \cdot 10^8$	X	0
$\%limit$	25	45	70	100	0
$\%oneNE$	100	100	100	X	100
$\%fakeNE$	0	0	0	X	0
$\bar{\epsilon}$	X	X	X	X	X

Table A.12: Games with $n = 2$, $|S_i| = 10$ and more than one Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	24,557.08	96,681.6	$1.4 \cdot 10^5$	X	$2.5 \cdot 10^5$
$\sigma(B)$	3,808.01	16,338.71	19,944	X	0
$\sigma^2(B)$	$1.45 \cdot 10^7$	$2.67 \cdot 10^8$	$3.98 \cdot 10^8$	X	0
$\%limit$	5	40	55.5	100	0
$\%oneNE$	100	100	100	X	100
$\%allNEs$	0	100	0	X	100
$\%fakeNE$	0	0	0	X	0
$\bar{\epsilon}$	X	X	X	X	X

Table A.13: Games with $n = 2$, $|S_i| = 15$ and no Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	$1.39 \cdot 10^5$	X	X	X	$2.5 \cdot 10^5$
$\sigma(B)$	25,732.39	X	X	X	0
$\sigma^2(B)$	$6.62 \cdot 10^8$	X	X	X	0
$\%limit$	20	X	100	X	0
$\%fakeNE$	0	X	X	X	0
$\bar{\epsilon}$	X	X	X	X	X

Table A.14: Games with $n = 2$, $|S_i| = 15$ and one Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	78,171.28	X	$2.07 \cdot 10^5$	X	$2.5 \cdot 10^5$
$\sigma(B)$	31,511.11	X	0	X	0
$\sigma^2(B)$	$9.93 \cdot 10^8$	X	0	X	0
$\%limit$	0	X	80	X	0
$\%oneNE$	100	X	100	X	100
$\%fakeNE$	0	X	0	X	0
$\bar{\epsilon}$	X	X	X	X	X

Table A.15: Games with $n = 2$, $|S_i| = 15$ and more than one Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	21,134.56	X	$1.41 \cdot 10^5$	X	$2.5 \cdot 10^5$
$\sigma(B)$	7,292.9	X	19,555.22	X	0
$\sigma^2(B)$	$5.32 \cdot 10^7$	X	$3.82 \cdot 10^8$	X	0
$\%limit$	0	X	32	X	0
$\%oneNE$	100	X	100	X	100
$\%allNEs$	0	X	0	X	100
$\%fakeNE$	0	X	0	X	4
$\bar{\epsilon}$	X	X	X	X	$5.14 \cdot 10^{-3}$

Table A.16: Games with $n = 2$ and $|S_i| = 3$.

B	200000.0	150000.0	100000.0	50000.0	25000.0	12500.0	6250.0
$\%oneNE$	100	100	100	100	99.67	100	100
$\%allNEs$	100	100	100	100	99.67	100	99.67
$\%fakeNE$	0.33	0.67	0.67	0	1	0.67	1
$\bar{\epsilon}$	$9.29 \cdot 10^{-4}$	$9.29 \cdot 10^{-4}$	$9.29 \cdot 10^{-4}$	X	$2.65 \cdot 10^{-3}$	$9.29 \cdot 10^{-4}$	$4.76 \cdot 10^{-3}$

Table A.17: Games with $n = 2$ and $|S_i| = 7$.

B	200000.0	150000.0	100000.0	50000.0	25000.0	12500.0	6250.0
$\%oneNE$	100	99.33	99.67	99.67	99.67	100	99
$\%allNEs$	99.67	99.33	99.33	99.67	99.33	99.33	98
$\%fakeNE$	0.67	0	0.33	1	1	1.33	2
$\bar{\epsilon}$	$2.26 \cdot 10^{-3}$	X	$2.41 \cdot 10^{-3}$	$2.12 \cdot 10^{-3}$	$2.26 \cdot 10^{-3}$	$2.26 \cdot 10^{-3}$	$4.52 \cdot 10^{-3}$

Table A.18: Games with $n = 2$ and $|S_i| = 15$.

B	200000.0	150000.0	100000.0	50000.0	25000.0	12500.0	6250.0
$\%oneNE$	100	100	100	100	100	100	100
$\%allNEs$	100	100	100	100	100	100	100
$\%fakeNE$	0	1.33	1.33	1.33	2.67	0	4
$\bar{\epsilon}$	X	$5.14 \cdot 10^{-3}$	$5.14 \cdot 10^{-3}$	$5.14 \cdot 10^{-3}$	$1.09 \cdot 10^{-2}$	X	$1.67 \cdot 10^{-2}$

A.2 Looking for pure NEs in 3-player games

Table A.19: Games with $n = 3$, $|S_i| = 3$ and no Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	66,476.11	66,476.11	X	X	$2.5 \cdot 10^5$
$\sigma(B)$	16,459.91	16,459.91	X	X	0
$\sigma^2(B)$	$2.71 \cdot 10^8$	$2.71 \cdot 10^8$	X	X	0
$\%limit$	25	25	100	100	0
$\%fakeNE$	0	0	X	X	0
$\bar{\epsilon}$	X	X	X	X	X

Table A.20: Games with $n = 3$, $|S_i| = 3$ and one Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	19,875.52	33,532.49	$1.04 \cdot 10^5$	X	$2.5 \cdot 10^5$
$\sigma(B)$	5,104.72	4,316.26	23,136.36	X	0
$\sigma^2(B)$	$2.61 \cdot 10^7$	$1.86 \cdot 10^7$	$5.35 \cdot 10^8$	X	0
$\%limit$	25	35	50	100	0
$\%oneNE$	100	100	100	X	99
$\%fakeNE$	0	0	0	X	1
$\bar{\epsilon}$	X	X	X	X	$1.69 \cdot 10^{-3}$

Table A.21: Games with $n = 3$, $|S_i| = 3$ and more than one Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	27,199.6	96,958.96	$1.17 \cdot 10^5$	X	$2.5 \cdot 10^5$
$\sigma(B)$	8,506.54	24,543.4	18,431.6	X	0
$\sigma^2(B)$	$7.24 \cdot 10^7$	$6.02 \cdot 10^8$	$3.4 \cdot 10^8$	X	0
$\%limit$	0	50	31	100	0
$\%oneNE$	100	100	100	X	100
$\%allNEs$	0	100	0	X	100
$\%fakeNE$	0	0	0	X	0
$\bar{\epsilon}$	X	X	X	X	X

Table A.22: Games with $n = 3$, $|S_i| = 5$ and no Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	$1.2 \cdot 10^5$	$1.2 \cdot 10^5$	X	X	$2.5 \cdot 10^5$
$\sigma(B)$	13,771.43	13,771.43	X	X	0
$\sigma^2(B)$	$1.9 \cdot 10^8$	$1.9 \cdot 10^8$	X	X	0
$\%limit$	30	30	100	100	0
$\%fakeNE$	0	0	X	X	0
$\bar{\epsilon}$	X	X	X	X	X

Table A.23: Games with $n = 3$, $|S_i| = 5$ and one Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	37,651.54	$1.41 \cdot 10^5$	92,116.4	X	$2.5 \cdot 10^5$
$\sigma(B)$	13,851.37	20,580.23	16,447.77	X	0
$\sigma^2(B)$	$1.92 \cdot 10^8$	$4.24 \cdot 10^8$	$2.71 \cdot 10^8$	X	0
$\%limit$	35	55	60	100	0
$\%oneNE$	100	100	100	X	100
$\%fakeNE$	0	0	0	X	0
$\bar{\epsilon}$	X	X	X	X	X

Table A.24: Games with $n = 3$, $|S_i| = 5$ and more than one Nash.

Algorithm	PNE-LUCB	PNE-LUCB (All)	LUCB-NG	LUCB-NG (All)	SE
$\bar{B}$	35,366.84	$1.3 \cdot 10^5$	$1.66 \cdot 10^5$	X	$2.5 \cdot 10^5$
$\sigma(B)$	7,633.7	15,177.09	21,947.44	X	0
$\sigma^2(B)$	$5.83 \cdot 10^7$	$2.3 \cdot 10^8$	$4.82 \cdot 10^8$	X	0
$\%limit$	10	65	67	100	0
$\%oneNE$	100	100	100	X	100
$\%allNEs$	0	100	0	X	98
$\%fakeNE$	0	0	0	X	2
$\bar{\epsilon}$	X	X	X	X	$1.75 \cdot 10^{-3}$