Learning Equilibria in Games: A Linear Best Arm Identification Approach

Tesi di Laurea Magistrale in
Computer Engineering - Ingegneria Informatica

Author: **Edoardo Lamonaca**

Student ID: 920012
Advisor: Prof. Alberto Marchesi
Co-advisors: Francesco Trovò
Academic Year: 2021-22

Abstract

We deal with a problem in the field of empirical game theoretic analysis. Our problem is to find a max min strategy for the first player in a simulation based game. In simulation based game there is not an explicit description of the utilities of the players, so we need to use a game simulator that can be queried to obtain noisy estimates of the utilities. In particular in BAI setting (best arm identification) the main goal is to find the best arm at the end of the exploration phase. In our case we are working in a linear environment, this means that all the arms available to the players are connected in some way and pulling one of them gives some information about other arms other than pulled arms.

In our work we built three algorithms to find the best arm in a two player game. Our setting is modelled as a bi linear setting, making re-use some information about a pull of an arm to make a better estimation of other arms. In particular we built two algorithms Super-arm-g and Bilinear-g that pulls each time the best two arms according to the estimation of the rewards and a third one called GAM-g that it is exploiting more the property of a linear setting, pulling sub optimal arms to explore in a better way the environment. Super-arm-g tries to use the information of a pull to make a better estimation of the reward of all arms, but in order to do so it needs to transform the space of the setting at the beginning of the algorithm. This operation make the environment bigger, also to perform this transformation it needs more information than the other two algorithms. Other two uses the information coming from a pull only on a sub set of arms.

Keywords: BAI, simulation based game, linear setting

Abstract in lingua italiana

In questa tesi trattiamo un problema nel campo dell'analisi teorica di giochi. Il nostro problema riguarda trovare la strategia max min ottima in un gioco basato su una simulazione. Un gioco basato su una simulazione è un gioco in cui non abbiamo disponibile una esplicita descrizione dell'utilità dei giocatori, ma abbiamo bisogno di usare un simulatore che può essere interpellato per restituire un stima con del rumore dell'utilità. In particolare nello scenario di BAI (identificaione miglior braccio) l'obbiettivo è trovare il braccio migliore per il primo giocatore alla fine della così detta fase di esplorazione. Nel nostro caso specifico ci troviamo in uno scenario di ambiente linear, questo significa che tutte le braccia disponibili ai giocatori sono connesse in qualche modo e tirare una di esse da alcune informazioni riguardo la ricompensa delle altre braccia oltre alla ricompensa del braccio tirato.

Nel nostro lavoro abbiamo costruito tre algoritmi per trovare il braccio migliore da giocare in un gioco con solo due giocatori. Il nostro scenario è modellato con una relazione bilineare tra la ricompensa, le braccia e l'ambiente, rendendo possibile usare le informazioni riguardo il tirare un braccio per fare una stima migliore delle altre braccia. In particolare abbiamo costruito due algoritmi Super-arm-g e Bilinear-g che tirano ad ogni iterazione dell'algoritmo il biglior braccio ed il secondo migliore, secondo le loro stime della ricompensa associata ad ogni braccio e un terzo algoritmi GAM-g che sfrutta maggiormente le proprietà di uno scenario lineare, tirando braccia sub ottime per esplorare meglio la conformazione dell'ambiente. Super-arm-g cerca di usare le informazioni che provengono dal tirare un braccio per avere una migliore stima del premio di tutte le braccia, ma per poter fare ciò necessita di trasformare lo spazio dello scenario all'inizio dell'iterazioni dell'algoritmo. Questa operazione ingrandisce lo spazio matematico dello scenario, ed inoltre per poter eseguire questa trasformazione necessita di piu informazioni degli altri due algoritmi. Gli altri due usano le informazioni ricevute dal tirare un braccio per migliorare le stime soltanto di una parte delle altre braccia.

Contents

1 | Introduction

The area of the thesis is linear Bandit, and in particular his application on the field of simulation of games.

The area that studied games is called Game Theory and a usual game consist of multiple player (in our settings two opposite player). Each player has usually multiple action available to him and for each action he makes he gets a reward that depends of the decision every player did. Analyzing a game consist in trying to predict which action will play each agents and which it is the one that gives a better reward.

There are different strategy and analysis and strategy we can try to compute, and we are interested in finding the best response for player two to all action available to player one and then fining the action for player one that gives the best reward assuming player two is using the best response.

In our case we are using a simulation-based game which means that we are getting a simulation of a real game setting, including some noise. When we get the rewards from the same two actions the rewards could be a bit different due to the noise we are adding to emulate a real world scenario.

To be able to solve this type of problem many different algorithms can be designed.

However, they can be generally divided into two big types of groups.

The first one make a complete first estimate of the utilities of the whole game and then, in a second step, compute the best response and best action of the estimated game.

The second one design a 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.

We are implementing algorithms of the second category, in particular our model it is based on multi-armed bandit model and we focus our interest in the specific part of best arm identification problem.

The problem that we want to resolve is a game where one agent has $x \in X$ possible strategies and an opponent agent has $y \in Y$ other possibles strategies, traducing this in a bi linear bandit setting.

We make some assumption to make the problem more tractable, such as the linearity of the game, meaning that we think that the reward of different arms are somehow all connected.

The game that we are hypothesizing is a zero sum game, so the utility of player one is the opposite of the utility of player two. The reward for the two agents given a strategy x and a strategy y is $x^\top \theta y + \epsilon$, where ϵ is the noise and it is a random variable and $\theta \in \Theta$ is an unknown matrix that characterized a context.

In the linear bandit setting, there are some important property to keep in mind, such the linearity of the reward, so we can use the reward of all the sample used from all arms to estimate a better θ and have a better estimation of the mean rewards of each arms.

In our case, it is a bit more difficult because we do not have a linear setting, but a bi-linear one. As it will appear during the description of the papers studied there is not a lot of studies about bi linear settings. We adapted linear algorithms and consideration on our case, even if it is slightly different. So we make the problem formulation as close as a linear setting.

In chapter four, there are the most important paper from which we took inspiration and information, trying to highlight the main point of each papers and the pesudocode of them.

In chapter five there is a more rigorous formulation of the problem that we want to solve and the supposition we are making about the structure of the setting and the information needed.

In chapter six there are described the three algorithms that we adapted to the studied setting and the consideration about the three different algorithm.

In chapter seven, There are the description of the test made and the results and comments on them. There are two different types of experiments, one using only our three algorithm tested on armed bandit of dimension 3 and the comments on the results of the experiments.

Then there is an explanation of another game that we used to test ours algorithm. So there is the description of the game and how we trained the agents and changed the environment and after the results of the experiments with the result of the tests.

Finally, there are some general consideration about the whole problem and possible future directions.

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.

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.

The Games can be represented in different ways according to the type of situation that wants to be analyzed. These may range from coordination problems [4, 5, 10, 16], to resource allocation [3, 17] and commitment problems [9, 18].

The two most commonly adopted representations in literature are normal-form game and extensive-form game.

We are going to consider the extensive-form game.

By game in extensive form we intend the mathematical formalization of all relevant information about the game. This means in particular to specify:

- the initial setting
- all possible evolutions (i.e., all allowed moves at each possible situation)
- all final outcomes of a game, and the preferences of the players on them.

The extensive form is useful to analyze game in which players take action sequentially.

Games in extensive form usually are represented by trees. When a tree represents a game, the root of the tree corresponds to the initial position of the game, and every game position is represented by a vertex of the tree. The children of each vertex v are the vertices corresponding to the game positions that can be arrived at from v via one action.

In other words, the number of children of a vertex is equal to the number of possible actions in the game position corresponding to that vertex.

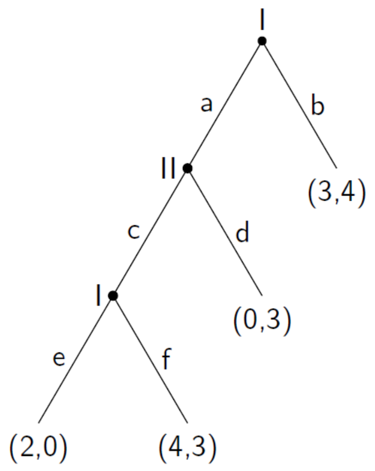

Figure 2.1: Tree representation of a game

2.1.1. Zero sum Games

A class of games we are going to use is zero-sum games: this means that there are two players and in every outcome of the game what one gets is the opposite of what gets the other one.

A two player zero sum game in strategic form is the triplet $(X; Y; f : X * Y \rightarrow R)$

X is the strategy space of Player 1, Y the strategy space of Player 2, $f(x; y)$ is what Player 1 gets from Player 2, when they play x, y respectively. Thus f is the utility function of Player 1, while for Player 2 the utility function g is $g = -f$.

2.1.2. Simulated-based games

Simulation-based games were firstly introduced by Vorobeychik and Wellman [28] 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 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). A simulation-based normal-form game $\tau = (N; S; u)$ 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;
- $u : S \to R_n$ is a vector of random variables representing players' stochastic utilities, whose realizations can be interpreted as outputs provided by the oracle O.

In particular, given a strategy profile $s \in S$, we denote with $u(s)$ the subvector of random variables corresponding to the players' utilities when the actions in s are played, with $ui(s)$ being the random variable representing the utility of player $i \in N$. With an abuse of notation, we define $u := E[u]$ as the expectation of u, with $u(s)$ and $u_i(s)$ assuming their respective meanings.

A possible example of simulation-based game is when $N = 1, 2$ and, for each strategy profile $s \in S$, the winning player (receiving a payoff of 1, instead of -1 when losing) is determined according to a Bernoulli distribution with parameter $\mu_s \in [0; 1]$ (assuming that these distributions are independent among each other). Formally, this can be modeled by letting, for every $s \in S$, $u1(s) \sim Be(\mu_s)$, while $u_2(s) = -u_1(s)$. 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 $u_i(s) \sim N(u_i(s); \lambda)$ for every $i \in N$ and $s \in S$, where $u : S \to R_n$ is the utility vector function of some underlying (true) normal-form game and $\lambda \in R_{>0}$ is the noise standard deviation.

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 [20] 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 [1].
- Unsupervised Learning: given unknown inputs, the goal is to learn a more efficient representation for them. The typical unsupervised learning problems are Clustering and Compression [23].
- 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 [22].

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 [15] 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, are able to output the of recommended arm. This task is called pure exploration problem (Bubeck et al [2]) 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. 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(μ), 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:
  - $A := a_1, a_2, ..., a_n$ is a set of N available arms;
  - R is a set of unknown distributions such that, at every time step t, $r_{i,t} \sim R(a_i)$ with $E[R(a_i)] = R(a_i)$.

Solving a MAB problem calls for algorithms that at each time step $t \in 1; ...; T$ 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.

Algorithm 2.1 Pure exploration problem

```
for $t = 1, 2, ..., T$ do
    Select one arm $a_i$
    Receive the reward from the environment $r_{i;t}$
    Estimate parameters and update information
end for
Output the best arm
```

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, dynamic pricing, network routing, and persuasion.

2.2.2. Best Arm Identification In Multi-Armed Bandit

In our work we consider only pure exploration problems, so when designing the algorithms we do not care about the cumulative reward, but we identify the best arm(s), given a maximum number of pulls T. 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 and how the arm to pull at each round is selected.

3 | State of the Art

In this chapter we review the current state of the art on simulation based game and on best arm identification on linear bandit.

The first computational studies on simulation-based games have been carried out by Vorobeychik et al, [28] who addressed the problem of solving a game by generalizing from some data a complete payoff-function representation as a standard regression problem. Their work has been also extended by Gatti and Restelli [12] 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. This work differ from us because we do not compute the Nash equilibria, but the best response.

Learning a game in the papers reported is done in two way: 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 or as a second way 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. In particular, Viqueira et al. [25, 26] 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. We are closer as approach to the second way to try to solve the problem.

More specifically we use an approach similar to the Fixed-budget setting. This means that we have a specified number of pulls, called budget, and we have to maximize the probability to find the best arm under the specified budget.

We would like to mention here the algorithm proposed by Audibert et al. [29], 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 [19]. in their work proposed an algorithm, called GP-SE, inspired by the one of Audibert et al. [29], but with an elimination rule that is suitably defined for the problem of identifying maxmin equilibrium 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.

We have to mention the linear bandit setting which is very close to our setting. In the linear bandit setting, instead of having N independent arms, each of the N arms is associated with a d-dimensional attribute vector, and the expected reward is the linear combination of its own attribute vector and an unknown global vector θ. Each pull of an arm reports a stochastic reward centered at its expectation, and the goal is to pull as few times as possible to identify the arm with the highest expected reward. Due to the linear structure of the problem, pulling each arm reveals information about the global vector θ and therefore, indirectly, about the expected reward of other arms. This setting is studied by at the beginning by Soare, Lazaric, Munos [21] introducing concept of static allocation strategy to understand which are the best distribution of arm to pull, knowing that pulling a sub optimal arm can still give valid information to find the best arm. This concept is a type of G-allocation that it is proven to be NP-Hard. Another way to use the information of the linearity was made by Chao, Tao, Blanco, Zhou [24] which a relaxation of the given G-allocation, making it more feasible. Then we have to mention a new approach by Degenne et all. [8] which avoid the need to solve an optimal design problem and provides some theoretical result on the comparison with the approaches that needs the solution of G-optimality.

4 | Background

In this section, we present the main papers on linear bandit problems, which will be used as building blocks for the following definition of our solution.

4.1. Best Arm Identification in Linear Bandits with Linear Dimension Dependency [24]

We are given a set of N arms $X = \{x_1, x_2, ..., x_n\}$. Each arm is associated with a d-dimensional attribute vector and use $x_i \in \mathbb{R}$ to denote the vector for the arm x_i. We assume $||x_i||_2 \leq 1$ for all arms. We also assume w.l.o.g. that the rank of X is exactly d.

There is an unknown global vector $\theta \in \mathbb{R}^d$ with $||\theta||_2 \leq L$. Each pull of arm x_i reports a stochastic reward $x_i^\top \theta + \epsilon$ where ϵ is an independent κ-subgaussian noise.

Given a $\delta \in (0, 1)$ the goal is to design an algorithm with as small query complexity Q as possible that, with probability at least $(1 - \delta)$, sequentially makes at most Q pulls of the arms, and then identifies the arm with the highest mean reward. We assume $X_{[1]}^\top \theta > X_{[i]}^\top \theta$ for $i > 1$ where $X_{[1]}$ denotes the arm in X whose mean reward is the i-th largest. We let $\Delta_i = X_{[1]}^\top - X_{[i]}^\top$ be the reward gap between arm $X_{[1]}$ and arm $X_{[i]}$ for $i > 2$.

4.1.1. Complexity

The algorithm of this paper with confidence of $(1 - \delta)$ has sample complexity of:

$$O\left(\frac{d}{\Delta_2^2}(\ln \delta^{-1} + \ln N + \ln \ln \Delta_2^{-1})\right).$$

For the fully data-dependent algorithm, for confidence parameter δ, the sample complexity is:

$$O\left(\sum_{i=2}^{d} \Delta_i^{-2}(\ln \delta^{-1} + \ln N + \ln \ln \Delta_2^{-1})\right).$$

Algorithm, the Estimator for θ

We define a procedure to estimate the underlying unknown θ given a set S of arms and n samples.

Algorithm 4.1 Estimator θ

```
Require: A set of $S$ arms, and $n$ samples.
  Let $y_1, y_2, ..., y_n$ be the $n$ samples acquired from $S$ according to the distribution $\lambda_S^*$.
  Pull arms $y_1, y_2, ..., y_n$ and suppose their corresponding reward are $r_1, r_2, ..., r_n$
  Compute $A \leftarrow n \sum_{x \in S} \lambda_S^*(x) x x^\top$ and $b \leftarrow \sum_{i=1}^{n} r_i y_i$.
  $\hat{\theta} \leftarrow A^{-1} b$.
  OUTPUT: The estimate $\hat{\theta}$
```

This algorithm collect the samples acquired and uses it to decides which arm to pull. Then it collect the rewards and compute the multiplication above to make an estimation of θ.

4.1.2. X-dependent Algorithm

This is an iterative algorithm that eliminates $(|S| - p)$ suboptimal arms from S with probability at least $(1 - \delta)$.

In practice, during the r-th iteration, $ELIMTIL_p$ uses $VECTOREST$ to get an $\frac{\epsilon_r}{2}$-close estimate of $x^\top \theta$ for all $x \in S_r$ with probability at least $1 - \delta_r$, and then discards all arms whose estimated mean rewards are ϵ_r worse that that of the highest estimate mean. $ELIMTIL_p$ continues until at most p arms remain. By letting ϵ_r and δ_r decrease exponentially, we are able to keep the best arm in the set of output arms with probability at least $(1 - \delta)$.

Algorithm 4.2 X-dependent Algorithm

```
Require: Arms set $S$ and confidence level $\delta$.
  Initialize $S_1 \leftarrow S, r \leftarrow 1$.
  while $|S_r| > p$ do
        Set $\epsilon_r \leftarrow \frac{1}{2^r}, \delta_r \leftarrow \frac{6}{\pi^2} \frac{\delta}{r^2}$.
        $\hat{\theta} \leftarrow VECTOREST\left(S, c_0 \frac{2+(6+\frac{\epsilon_r}{2} d)}{(\frac{\epsilon_r}{2}^2)} \ln \frac{5|S|}{\delta_r}\right)$.
        Select arm $a_r \leftarrow \arg\max_{x \in S_r} x^\top \hat{\theta}_r$.
        $S_{r+1} \leftarrow \frac{S_r}{\{x \in S_r | x^\top \hat{\theta}_r < x_{a_r}^\top \hat{\theta}_r - \epsilon_r\}}$.
        $r \leftarrow r + 1$.
  Output: Set $s_r$.
```

4.1.3. Y-dependent Algorithm

This algorithm compare the mean reward of two arms and instead of checking the confidence of intervals of the two arms, it turn to the confidence interval of $y^\top \theta$ (where $y = x - x'$) and check the corresponding confidence interval.

At the r-th iteration, the confidence interval of $y^\top \theta$ has length $Err_{\lambda_T^*}(y, l_r, \theta) \leq err_{\lambda_T^*}(y, l_r)$ where the latter quantity is an upper estimate of the first one and does not need the knowledge about θ in advance. At the r-th iteration, we use $\hat{\theta}_r$ as an estimate of θ.

Algorithm 4.3 Y-dependent Algorithm

```
Require: Set $S$ of active arms (the ones not yet eliminated), set $S \subseteq T$ of arms those
  can be pulled, and confidence level $\delta$.
  Initialize $S_1 \leftarrow S, r \leftarrow 1$.
  while while $|S_r| > p$ do do
        Set $\delta_r \leftarrow \frac{6}{\pi^2}, l_r \leftarrow 4c_0(2 + (6 + \frac{4}{1.1^r})d)(\frac{1.1^r}{4})^2$.
        $\hat{\theta}_r \leftarrow VECTOREST(T, l_r \ln(5\frac{|S|^2}{2\delta_r})$.
        Select arm $a_r \leftarrow argmax_{x \in S_r} x^\top \hat{\theta}_r$.
        $S_{r+1} \leftarrow \frac{S_r}{\{x \in S_R | (x_{a_r} - x)^T \hat{\theta}_r > min\{err_{\lambda_T^*}(x_{a_r} - x, l_r), \hat{Err}_{\lambda_T^*}(x_{a_r} - x, l_r)\}}$
        $r \leftarrow r + 1$.
  end while
  Output: Set $S_r$.
```

4.1.4. Adaptive Linear Best Arm, ALBA

Using the $Y - ELIMTIL$ algorithm as a subroutine, they introduce an algorithm that outputs the best arm with complexity depends on $\Delta_1, \Delta_2, \ldots, \Delta_d$. To achieve this goal, the algorithm runs in rounds. During each round r, it invokes $Y - ELIMTIL$ to identify the top-($\frac{d}{2^r}$) arms and discards the remaining arms. Authors called this algorithm $ALBA$:

Algorithm 4.4 ALBA

```
Require: Arms set $X$ and confidence level $\delta$.
  Initialize $S_0 \leftarrow X$.
  for $r \leftarrow 0$ to $\lfloor log_2 d \rfloor$ do do
    Set $\delta \leftarrow \frac{6}{\pi^2} \frac{\delta}{(r+1)^2}$.
    $S_{r+1} \leftarrow Y - ELIMTIL_{\lfloor \frac{d}{2^r} \rfloor}(S_r, X \cap span(S_r), \delta_r))$.
  end for
  Output: Arm in $S_{r+1}$
```

4.1.5. Consideration

This algorithms do something very similar to what we are trying to achieve. In this case there is a linear setting and are selected some direction in which explore the the sub space made from the arms and the vector θ. In our setting having different arms of player two it is more difficult to exploit the property made from the difference of value of arms x of player one, because different arms y generate different sub spaces. So it is not trivial to adapt this algorithm in our setting, but was very interesting the idea of considering the difference of two arms x and the elimination of the arms that we are sure that are not good enough, leaving only a sub sets of arms as pullable in th net iteration of the algorithms.

4.2. Gamification of Pure Exploration for Linear Bandits [8]

In this paper the authors investigate an active pure-exploration setting, that includes best-arm identification, in the context of linear stochastic bandits. They design the first asymptotically optimal algorithm for fixed confidence pure exploration in linear bandits.

They treat the problem as a two-player zero-sum game between the agent and the nature and they search for algorithms that are able to output a correct answer with high confidence to a given query using as few samples as possible.

4.2.1. Linear Bandit

The problem is a finite-arm linear bandit problem, where the collection of arms $A \subset \mathbb{R}^d$ is given with $|A| = A$ and spans R^d. We assume that $\forall a \in A, ||a|| \leq L$ where $||a||$ denotes the Euclidean norm of the vector a. The learning protocol goes as follows: for each round $1 \leq t \leq T$, the agent chooses an arm $a_t \in A$ and observes a noisy sample $Y_t = (\theta, a_t) + \eta_t$, where $\eta_t \sim N(0, \sigma^2)$ is conditionally independent from the past and θ is some unknown regression parameter. They assume $\sigma^2 = 1$.

4.2.2. Correct answer

We assume that θ belongs to some set $M \subset \mathbb{R}^d$ known to the agent. For each parameter a unique correct answer is given by the function $i^* : M \leftarrow I$ among the $I = |I|$ possible ones, and the answer is the arm with the largest mean.

For any answer $i \in I$ is defined the alternative to i, denoted by $\neg i$ the set of parameters where the answer i is not correct. $\neg i := \{\theta \in M : i \neq i^*(\theta)\}$

4.2.3. Design Matrix and Norm

w is the probability distribution to play the arms.

For any $w \in (\mathbb{R}^+)^A$ the design matrix is $V_w := \sum_{a \in A} w^a a a^\top$.

They define $||x||_V := \sqrt{x^\top V x}$ for $x \in \mathbb{R}^d$ and a symmetric positive matrix $V \in R^{dxd}$. Note that it is a norm only if V is positive definite.

It is denoted by $\sum_K$ the probability simplex of dimension $K - 1$ for all $K \geq 2$.

4.2.4. Sion's Minimax Theorem

The algorithm inverts the order of the move of the players. This is possible thanks to Sion's Minimax theorem [14] which is a generalization of the John Von Neumann minimax theorem [27]. The Sion's Minimax theorem says: Let X be a compact convex subset of a linear topological space and Y a convex subset of a linear topological space. If f is a real-valued function on $X * Y$ with:

- $f(x, \cdot)$ upper semicontinuous and quasi-concave on $Y, \forall x \in X$, and
- $f(\cdot, y)$ lower semicontinuous and quasi-convex on X, $\forall y \in Y$

then,

$$\min_{x \in X} \sup_{y \in Y} f(x, y) = \sup_{y \in Y} \min_{x \in X} f(x, y).$$

4.2.5. LinGame

This is one of the two algorithms presented in this paper. At the beginning it is described the stopping rule. Then in each iteration it chooses the best arm as answer with the information from the previous iteration. Then it proceeds to chose the arm to pull. It select the one that it is more far away than the cumulative suggestion made from Ada Hedge in previous iteration and the number of times each arm it is actually pulled from this algorithm. Then select the best response from the nature and collect the reward and add the new information to all the saved variables.

4.2.6. Algorithm

Algorithm 4.5 LinGame

```
for $t = 1, ...$ do
    Stopping Rule: $max_{i \in I} inf_{\lambda : i \in \neg i} \frac{1}{2} ||\theta_t^{\sim} - \lambda_i||^2_{V_{N_t}} > \beta(t, \delta)$ stop and return $i^*$
    Best answer: $i_t = i^*(\theta_{t-1}^{\sim})$
    Agent play: Get $w_t$ from $L_w^{i_t}$ and update $W_t = W_{t-1} + w_t$
    Response Nature: Best response for nature $\lambda_t \in argmin_{\lambda \in \neg i_t} ||\tilde{\theta}_{t-1} - \lambda||^2_{V_{w_t}}$
    Gains: Feed learner with $g_t(w) = \sum_{a \in A} \frac{w^a U_t^a}{2}$
    Pull: $a_t \in argmin_{a \in A} N_{t-1}^a - W_t^a$
end for
```

4.2.7. Estimation of theta

They fix a regularization parameter $\eta > 0$. The regularized least square estimator for the parameter $\theta \in M$ at time t is $\theta^{\sim} = (V_{N_t} + \eta I_d)^{-1} \sum_{s=1}^{t} Y_s a_s$ where I_d is the identity matrix. By convention $\tilde{\theta}_0 = 0$.

4.2.8. Stopping Rule

The stopping rule is:

$$max_{i \in I} inf_{\lambda : i \in \neg i} \frac{1}{2} ||\tilde{\theta}_t - \lambda_i||^2_{V_{N_t}} > \beta(t, \delta),$$

and it returns the index:

$$i_t^* \in argmax_{i \in I} inf_{\lambda_i \in \neg i} \frac{||\tilde{\theta}_t - \lambda_i||^2_{V_{N_t}}}{2}.$$

4.2.9. Saddle Frank-Wolfe heuristic for AB-design

They propose a variant of the Frank-Wolfe heuristic [11] that takes into account a count for each element in the transductive set B. The pseudo-code of our method is displayed after. Sanity check on various MAB instances shows the correctness of their heuristic, its convergence guarantee remains for the future work.

4.2.10. AdaHedge: learner for agent

The Frank-Wolfe heuristic has two different agent. The first agent (the one that chooses w) is implementing AdaHedge algorithm [7].

AdaHedge is a regret minimizing algorithm of the exponential weights family.

It select a distribution of probability of playing all the available arms that depends of the cumulative regret of each arm. It is used to select and ideal distribution of arms to play in order to explore as good as possible the sub spaced made from the θ and the arms.

4.2.11. Optimism

The saddle point argument would be correct for a known game, while the algorithm is confronted to a game depending on the unknown parameter They use the principle of Optimism in Face of Uncertainty. Given an estimate $\theta^{\sim}_{t-1}$ we compute upper bounds for the gain of the agent at θ, and feed these optimistic gains to the learner.

4.2.12. Tracking

In the algorithm, the agent plays weight vectors in a simplex. Since the bandit procedure allows only to pull one arm at each stage, the algorithm needs a procedure to transcribe weights into pulls. This is what they call tracking. The choice of arm is $a_{t+1} \in argmin_{a \in A} N^a_t - W^a_{t+1}$.

4.2.13. Consideration

This algorithm is quite close to our setting, even if it solves as the others a linear bandit problem. We adapted this algorithm to solve a bi linear settings as described in the section GAM-g algorithm (the third one). This not try to invert matrix or operation like that, but only consider the reward an the losses associated at each pull, making more adaptable in our case.

The result of this papers show how the algorithm it is good not only in theory, but also in practice. They also shown how the algorithm it is faster in the execution time, making feasible for us to compute the experiments and compare different algorithms.

4.3. Contextual Bandits with Linear Payoff Functions [6]

In this paper the authors analyze a polynomial-time algorithm for the contextual bandit problem with linear payoffs. The algorithm is simpler and more robust in practice than its precursor. They also give a lower bound showing that a regret of $\overline{O}(\sqrt{Td})$ cannot be improved, modulo logarithmic factors.

4.3.1. Problem Setting

Let T be the number of rounds and K the number of possible actions. Let $r_{t,a} \in [0, 1]$ be the reward of action a on round t. On each rpund t for each action a the learner observes K feature vectors $x_{t,a} \in \mathbb{R}^d$, with $||x_{t,a}|| \leq 1$, where $||.||$ denotes the l_2-norm. After observing the feature vectors the learner then selects an action a_t and receives reward r_{t,a_t}.

They operate under the linear realizability assumption: there exists an unknown weight vector $\theta^* \in \mathbb{R}^d$ with $||\theta^*|| \leq 1$ so that

$$E[r_{t,a}|x_{t,a}] = x_{t,a}^\top \theta^*$$

for all $t$and a. Hence,we assume that the $r_{t,a}$ are independent random variables with expectation $x_{t,a}^\top \theta^*$.

Let $x_{t,a}$ be the feature vector of action a at step t, and define the regret of an algorithm A to be

$$\sum_{t=1}^{T} r_{t,a_t^*} - \sum_{t=1}^{T} r_{t,a_t},$$

where $a_t^* = argmax_a x_{t,a}^\top \theta^*$ is the best action at step t according to θ^* and a_t is the action selected by A at step t.

4.3.2. LINUCB

LinUCB decomposes the feature vector of the current round into a linear combination of feature vectors seen on previous rounds and uses the computed coefficients and rewards on previous rounds to compute the expected reward on the current round.

4.3.3. Algorithm

4.3.4. Base LinUCB and SupLinUCB

While experiments show LinUCB is probably sufficient in practice, there is technical difficulty in analyzing it.

In their analysis, the aithors need the predicted set of rewards on the current round to be computed from a linear combination of rewards that are independent random variables, in order to apply the Azuma/Hoeffding inequality. However, LinUCB has the problem that

Algorithm 4.6 LinUCB

```
Require: $\alpha \in \mathbb{R}_+, K, d \in \mathbb{N}$
  $A \leftarrow I_d$ The $d$-by-$d$ identity matrix
  $b \leftarrow =_d$
  for $t = 1, 2, 3, ..., T$ do
    $\theta \leftarrow A^{-1} b$
    Observe $K$ features,$x_{t,1}, x_{t,2}, ..., x_{t,K} \in \mathbb{R}^d$
    for $a = 1, 2, ..., K$ do
      $p_{t,a} \leftarrow \theta_t^\top x_{t,a} + \alpha \sqrt{x_{t,a}^\top A^{-1} x_{t,a}}$ Compute upper confidence bound
    end for
    Choose action $a_t = \arg\max_a p_{t,a}$ with ties broken arbitrarily
    Observe payoff $r_t \in [0, 1]$
    $A \leftarrow A + x_{t,a_t} x_{t,a_t}^\top$
    $b \leftarrow b + x_{t,a_t} r_t$
  end for
```

predictions in later rounds are made using previous outcomes. To handle this problem, we modify the algorithm into BaseLinUCB which assumes statistical independence among the samples, and then use a master algorithm SupLinUCB to ensure the assumption holds. This technique is similar to the Lin-Rel/SupLinRel decomposition by Auer [2002].

The idea of SupLinUCB is similar of LinUCB, so it is not of great interest to show also that algorithms.

4.3.5. Consideration

This paper it is focus on finding the best arm while trying to reduce the regret of the learning phases. In our setting we are addressing a best arm identification problem, meaning that we do not care about the regret in the learning phases, but only the probability of choosing the wrong arm t the end of the of the learning phases. Still it is an important paper on the field of linear payoff bandits and the algorithms described above are quite similar to the first two we are presenting on this thesis.

4.4. Best-Arm Identification in Linear Bandits Marta Soare Alessandro Lazaric Rémi Munos

This paper formalize the pure exploration setting of a MAB problem introducing the liner dependency of the reward to the x of the pulled arm.

4.4.1. Setting

Let $X \subseteq \mathbb{R}^d$ be a finite set of arms, where $|X| = K$ and the l_2-norm of any arm $x \in X$ is upper bounded by L.

Given and unknown parameter $\theta^* \in \mathbb{R}$, they assume that each time an arm $x \in X$ is pulled a random reward r(x) is generated according to the linear model $r(x) = x^\top \theta + \epsilon$, where ϵ is a i.i.d. noise bounded in $[-\sigma, \sigma]$.

Arms are evaluated according to their expected reward $x^\top$ and we denote by $x^* = argmax_{x \in X} x^\top \theta^*$ the best arm in X. Also they use $\Pi(\theta) = argmax_{x \in X} x^\top \theta$ to refer to the best arm corresponding to an arbitrary parameter θ. Let $\Delta(x, x') = (x - x')^\top \theta^*$ be the value gap between two arms, then they denote by $\Delta(x) = \Delta(x^*, x)$ the gap of x w.r.t. the optimal arm and by $\Delta_{min} = min_{x \in X} \Delta x$ the minimum gap, where $\Delta_{min} > 0$.

They also introduce the sets $Y = \{y = x - x', \forall x, x' \in X\}$ and $Y^* = \{y = x^* - x, \forall x \in X\}$ containing all the directions obtained as the difference of two arms (or an arm and the optimal arm) and redefine accordingly the gap of a direction as $\Delta(y) = \Delta(x, x')$ whenever $y = x - x'$.

4.4.2. XY-Allocation strategy

For any fixed n define the XY-allocation as

$$x_n^{XY} = arg \overset{[}{x}_n]\min \overset{[}{y} \in Y]\max ||y||_{A_{xn}^{-1}}$$

Arms are pulled with the objective of increasing the accuracy of directions in Y instead of arms X. Let x_n be a fixed allocation sequence and $\tilde{\theta}_n$ its corresponding estimate for θ^*. If and arm $x \in X$ is such that

$$\exists x' \in X s.t. c ||x' - x||_{A_{xn}^{-1}} \sqrt{log_n(K^2/\delta)} < \Delta`_n(x', x)$$

than arm x is sub-optimal. Moreover if is true we say that x' dominates x. Then construct the set of potential arm by removing from X all the dominated arms. So the stopping condition of the algorithm is to check if the number of non dominated arms is equal to 1.

Using $\hat{\mathrm{X}}(x_n)$ we construct $\tilde{\mathrm{Y}}(x_n) = \{y = x - x'; x, x' \in \tilde{\mathrm{X}}(x_n)\}$, the set of directions along which the estimations of θ^* needs to be improved. Let $j \in \mathbb{N}$ be the index of a phase and n_j its corresponding length. We denote by $\hat{\mathrm{X}}_j$ the set of non-dominated arms constructed on the basis of the samples collected in the phase $j - 1$. This set is used to identify the directions in $\tilde{\mathrm{Y}}_j$. Formally in phase j we implement a XY-allocation restricted on $\hat{\mathrm{X}}_j$.

Once phase j is over, the OLS estimate $\tilde{\theta}^j$ is computed using the rewards observed within phase j and then is used to test the stopping condition. Whenever the stopping condition does not hold a new set $\tilde{\mathrm{X}}_{j+1}$ is constructed using the discarding condition and a new phase is started.

4.4.3. Consideration

While the settings of this paper is quite generic and could be used for our interest, the concept of dominated arms where not immediate to transpose in our setting, having some difficulties in removing an arm for player one only in some response of player two. We decided to not use any mechanism of deleting some arms, but still it was worth the consideration of the mechanism.

5 | Problem Formulation

We had two slightly different problem formulation. Both problems consist of finding the best strategy of the player one regardless of the strategy of player two. The difference consist in the information available to the algorithm while it is running. In one only the action available to player one are known in advance, in the other both action available to player one and two are known.

This it is quite a huge difference, because knowing the composition of the actions available to player two in addition to the structure of the game we are supposing make it possible to transform the structure of the problem in an equivalent problem with only one action, that is embedding the conjunction of a couple of action (x_i, y_j), one for each player.

To understand the difference between the two try to think of a game of two kids Anna the fist one and Bob the second one. They wants to fight with their imagination with their preferred animals. At the beginning each of the two kids chooses three animals. Then at each round they decide their animal and their father decides which animal will win between them. Because Anna is older, Bob can choose the animal to send to the fight after Anna and knowing Anna's choice. Now we have the two cases. In the first case Anna only know her animals and before each battle decides which one to send. Anna does not know the animals possessed by Bob and does not listen of the motivation of how the fight between those two animals go, but only the final result (win or loss). In the second case Anna can see which one are the three animals chosen by Bob and exploit this information, like maybe Bob likes ground animals, and Anna can exploit it like choosing a bird or a sea animals, whereas in the first setting Anna can only speculate on why some of her animals won a fight or lost one.

The point of this examples is that knowing some information about the strategy of the opposite player can lead to a faster learning of the best strategy for player one, whereas not knowing them lead to the need to have more sample to be sure that strategy is better than another one.

The setting of the problem consist of n arms $x \in X$ of dimension k available to player one, m arms $y \in Y$ of dimension k available of player two and a matrix M of dimension

$m * m$ that represent the setting of the scenario in which we are playing. The sets of arms X and Y are finite.

Elements of arms x and y are bounded in the $[0, 1]$, and also the l_2 norms of an arm x or y is upper bounded by L.

In the first formulation the number of the arm of player one are known and are also known all the value of the vectors $x \in X$. We also know the number of vectors available to player two and also the values of all vectors $y \in Y$.

In the second formulation we know the number of arms available to player one $x \in X$ and the number of arms available to player two $y \in Y$. We also know the composition of the arms $x \in X$, but not the composition of the arms $y \in Y$.

In both cases, we are trying to represent a zero sum games where the reward on a single action is modelled as $reward = x^\top * W * y + \epsilon$ where ϵ is a noise and is represented by a i.i.d. bounded in $[-0.1, 0.1]$. The algorithms we are considering choose at each time step an arm to play for player one and an arm to play for player two. Then we observe a reward and according to the observation and the implementation of the algorithms we choose at each time step which one we think is the best arm for player one.

The goal is to find an algorithm that find the best arm and keep choosing it as fast as possible. We are using in our experiments a time horizon of 800 to see if the algorithms is converging on choosing the best arm. We are also using as a metrics the error that the algorithms is doing at each time step in choosing the arm, seeing if this is converging to zero in the time horizon chosen.

6 | Formalization and Implementation

In this section we describe all the three different algorithm implemented. For the studied setting we will describe the information needed to run it properly, the algorithm and the papers from which it comes from, all the steps that the algorithm do to work and some consideration from each algorithm.

6.1. Super-arm-g algorithm

6.1.1. General Description

This algorithm transform the arms X and Y available to player one and two, to a single super arms Z that represent both. The transformation is made extending the dimension of the vector of the arm, adding in each position a different part of the multiplication of the two vectors placing in the first place the element $x_1 * y_1$, then $x_1 * y_2$ ... $x_n * y_n$ and so on. We indicate the super arms $z \in Z$. So the model of the reward from $r = x^\top * M * y + \mu$ transform into $r = z^\top * \theta + \mu$ where μ is a iid representing the noise. This transformation to not loose information has to increase the dimension of the arms Z, making it k^2 where k is the dimension of arms x and y. As well the matrix M is transformed in a vector θ that collects all the information encapsulated in the matrix, arranging the elements of the matrix in a vector in the same order of the product of the super arms, making so that $x^\top * M * y = z^\top * \theta$

After this transformation we use linear armed bandit algorithm such as LinUCB to solve the problem to find the best arm x for player one.

As I will discuss later we have to keep track of the arms from which we generated the super arms during the computation of the algorithm. The algorithm has to find the worst super arm associated at each arm $x \in X$ and then find the best super arm. This to keep the information that player two is playing the best response.

6.1.2. Information needed

This algorithm has all $x \in X$ (arms available to player one) and $y \in Y$ (arms available to player two) known. We need to know them both to create and known the composition of the super arms $z \in Z$ and run properly a linear bandit algorithm, that need to know usually the composition of the arms X that are available to player one. The setting of the world represented by a matrix is unknown as is unknown the θ after the transformation, this is the usual setting of a linear bandit problem.

6.1.3. Inspiration and papers

This algorithm uses a very basic algorithm to work that comes from the "Best-Arm Identification in Linear Bandits, Marta Soare Alessandro Lazaric Rémi Munos" [21], for the algorithm that is slightly modified to accomplish all the requirements, and it is using some ideas from "Learning Probably Approximately Correct Maximin Strategies in Simulation-Based Games with Infinite Strategy Spaces" [19] regarding the pulling of two arms at each iteration.

6.1.4. Making a super arm

Before starting the iteration of the algorithm, we take all the possible couples of x and y and we create a super-arm of dimension $k * k$ where k is the dimension of the normal arm. We make this super-arm by placing in the first place the element $x_1 * y_1$, then $x_1 * y_2$ ... $x_n * y_n$.

Super arms are called $z \in Z$ for the rest of this paragraphs.

So we do not use anymore the two sets of arm X and Y, but the new set of *superarms*. But we still need to track from where those super arms came from for two reasons. The first one is that after detecting the best super arm, we should be able to detect the best arm x and the best arm y. The second reason is that during the exploration phases not all the super arms are the same. We have to find the worst super arms from each set of super arms that correspond to each arms x, and then find the best super arms among them. This because we are representing a zero-sum games, so the gain of player two is the opposite of the gain of player one.

6.1.5. Estimate theta

To obtain an unbiased estimate of θ we use an ordinary least square (OLS) using the super arms. We have that $\tilde{\theta} = A^{-1} \cdot b$ where $A = \sum_t (z_t^T \cdot z_t) + I_d$ and $b = \sum_t z_t \cdot r_t$ where z_t and r_t are the arm played at time t and the reward associated.

We need to add the identity matrix because we have no control over the $z \in Z$ that could be dependent linearly making the matrix singular.

6.1.6. Media reward and upper and lower bound

For selecting the best arm we are using the media reward of arms x_t used at time t. Media reward is equal to $\mu = z_t \cdot r_t^\top$. In a similar way we calculate the upper and lower bound of the reward of each arm. The bound is equal to $bound = 2\sigma$.

6.1.7. Find worst y

For each arm $x \in X$ at each iteration we have to find the worst (for player one and so best for player two) $y \in Y$. In order to do so we use the mean reward of each super arms $z \in Z$ associated to that arm x and we select the smallest mean reward among them, and we select the associated y_i associated to that specific z_i super arm. Then we save for this iteration that specific y_i and z_i as worst opposite arm for x_i, and repeat this operation for each $x \in X$.

6.1.8. Find best X

Between all the super arms z selected in the phase before, the algorithm selects the best arm to play for player one. In this way we are going to reduce the uncertainty of the reward of the arm we are thinking is the best one.

6.1.9. Find second best

In order to understand if the arm selected in the previous point is the best one, we have to explore the others promising arms. To do so we select the best arm between the remaining ones (so not the one chosen in the previous point), but in this case we do not use the media reward, but the upper bound of all the super arms. In this way we are selecting the arm that has more potential, and probably it was not pulled enough.

6.1.10. Pull arms

After selecting the best first and second arm we pull them and save the reward and the number of the pulled arm. In this way we can use those information to make a better estimate of theta in the next iteration.

6.1.11. Pseudocode

Algorithm 6.1 Super-arm-g algorithm

```
Require: $X, Y$
  $z \leftarrow \forall x \in X, \forall y \in Y$ Computer the super arms
  Pulls all $z$ two times
  for $t = 1, 2, 3, ..., T$ do
    $A \leftarrow A + z_{t,a_t} z_{t,a_t}^\top$
    $b \leftarrow b + z_{t,a_t} r_t$
    $\theta \leftarrow A^{-1} b$ Compute estimation of $\theta$
    for all z $\in Z$ do
      Compute mean reward and upper bound
    end for
    for all $x \in X$ do
      find $z_i$ associated to $x$ with lowest mean reward
      $Low \leftarrow z_i$ save it
    end for
    for all $z \in Low$ do
      Find best arm to pull with mean reward and second best arm with upper bound
    end for
    Pull the two selected arms and observe the reward
  end for
```

6.1.12. Consideration

This algorithm has the advantage to use all the rewards and samples to make a better estimation of θ. In this way we need a small amount of samples to converge to the best arm and to have a good estimation of θ (even if it is not the objective we are pointing to). Also transforming the algorithm in a very similar way to a standard linear bandit setting make us use algorithms in literature that are well known to be good and with theoretical guarantee so that we can rely on them.

On the other hand, the "trick" we did to make the set of super arms Z made the subspace larger, probably making way more difficult to use this approach in case of very big arms x and y as partially analyzed in [13]. The size of the space becomes larger in the power of two and we have to take it in consideration when using this algorithm.

6.2. Bilinear-g algorithm

6.2.1. General Description

This algorithm tries to transform the bi-linear setting in a normal linear multi armed bandit one. To do so we have to remember that we are modelling the reward as $x * M * y^\top + \mu$. We can see this algorithm as a n algorithm where n is the number of arms y where the reward is equal to $r = x * (M * y^\top) + \mu = x * \theta * \mu$, so we can use a linear bandit problem algorithm to solve part of the problem and as

6.2.2. Inspiration and papers

This algorithm uses a very basic algorithm to work that comes from the "Best-Arm Identification in Linear Bandits, Marta Soare Alessandro Lazaric Rémi Munos" [21], for the algorithm that is slightly modified to accomplish all the requirements, and it is using some ideas from "Learning Probably Approximately Correct Maximin Strategies in Simulation-Based Games with Infinite Strategy Spaces" [19] regarding the pulling of two arms at each iteration.

In a sense this is quite similar to the idea to run m different linear bandit algorithm where m is the number of different arms y available to player two. This is a refined version of it that is balancing how many times an arm y_j is used and the number of sampling are not equally divided between them. But still the sampling that are used to estimate a θ_j associated to a single y_j can not be used to improve the estimation of a different θ_k associated to a different y_k. This is for sure a great limitation and it is the cost to keep the algorithm very simple and similar to the one cited before in the papers and without making a transformation of the given arms and spaces.

6.2.3. Information needed

In this model we know in advance the number of arms $x \in X$ available to player one and $y \in Y$ available to player two. But we know the composition only of the arms $x \in X$ available to player one, and we do not know the composition of the arms $y \in Y$ available to player two. The setting of the world represented by a matrix is unknown.

6.2.4. Estimate theta

In order to obtain an unbiased estimate of θ we use an ordinary least square (OLS) using the arms $x \in X$. We have that $\tilde{\theta} = A^{-1} * b$ where $A = \sum (x_t * x_t^\tau) + I_d$ and $b = \sum x_t * r_t$

where x_t and r_t are the arm played at time t and the reward associated. In this scenario we can not use all the pull to have a better estimate of θ, because when we use different y arms we have different θ associated, so we can not use the information all together, but we have to divide it. So we will generate a different estimate of θ for each arms $y \in Y$, and each arms has it his own umber of samples and rewards associated. Each y ha sample in common between different x pulled, but they not share samples between different $y \in Y$. It is quite evident that in this way we need a substantial number of samples for each arms y in order to have a decent estimate of the θ associated.

6.2.5. Media reward and upper and lower bound

For selecting the best arm we are using the media reward of arms x_t used at time t. Media reward is equal to $\mu = x_t * r_t^\top$. In a similar way we calculate the upper and lower bound of the reward of each arm. The bound is equal to $bound = 2\sigma$. Finally, the upper bound is the mean reward plus the bound.

6.2.6. Find worst y

For each arm $x \in X$ at each iteration we have to find the worst (for player one and so best for player two) $y \in Y$. In order to do so we use the θ estimated of each arms $y \in Y$ and we make the multiplication between θ and the arm x to determine which y generate a lower reward.

6.2.7. Find best X

Now we have selected for each arm x the best response y and it is mean reward. In order to determine the best x we simply select the x with the mean reward higher between the best response. We have to be aware that selecting a specific x_i we are also selecting a specific y_j associated to it this round, that could be different from each other arms x, making quite less impactful in the sense of making a better estimation of *theta* that needed to be explored more.

6.2.8. Find second best

In order to understand if the arm selected in the previous point is the best one, we have to explore the others promising arms. To do so we select the best arm between the remaining ones (so not the one chosen in the previous point), but in this case we do not use the media reward, but the upper bound of rewards. In this way we are selecting the arm that has more potential, and probably it was not pulled enough. So we do now use the upper bound of the reward associated to each pair (arm x, arm y) that we are taking into consideration.

6.2.9. Pull arms

After selecting the best first and second arm we pull them and save the rewards and the numbers of the pulled arms. In this way we can use those information to make a better estimate of theta in the next iteration, using both the information observed from both pulls. We save the reward generated and the arms used in this iteration, saving the information in a separate structure for each arms y pulled.
In the reward we are observing there is some noise that can change the reward we are observing each time from the same couple of arms selected.

6.2.10. Pseudocode

Algorithm 6.2 Bilinear-g algorithm

```
Require: $X$ arms available to player one, $m$ number of arms to player two
  Pulls all $x$ two times with all the $y \in Y$ available
  for $t = 1, 2, 3, ..., T$ do
    for $y \in Y$ do
      $A_i \leftarrow A_i + x_{t,a_t} x_{t,a_t}^\top$
      $b_i \leftarrow b_i + x_{t,a_t} r_t$
      $\theta_i \leftarrow A^{-1} b$ Compute estimation of $\theta$
    end for
    for all x $\in Z$ do
      Compute mean reward and upper bound
    end for
    for all $x \in X$ do
      find $\theta_i$ that gives the lowest mean reward associated to $x_i$
      $Low \leftarrow \theta_i$ save it
    end for
    for all couple $(\theta, x_i) \in Low$ do
      Find best arm to pull with mean reward and second best arm with upper bound
    end for
    Pull the two selected arms and observe the reward
  end for
```

6.2.11. Consideration

This algorithm uses the minimum information required to work using only the number of arms X and Y available, the dimension of the arms and the composition of only the arms X available to player one. This make this algorithm more versatile, but probably needs more samples to explore in a proper way all the possible option and making more difficult to making a very precise estimate of the best θ. But on the other hand is very simple to implement and scales well with the dimension of the arms not adding more dimension to them making faster to invert the matrix to make an estimation of θ compared to the previous algorithm. The way we are trying to deal with the problem bring some drawback, we can not use all the samples to make a better estimation of the $theta$, but we have to separate them into smallest batch, one for each different arm $y \in Y$. So in comparison to the other algorithm with the same number of pulls we should have a less precise estimate and performance. But in this way we need way less information to run the algorithm, making it more feasible to be used in a real world application.

6.3. GAM-g algorithm

6.3.1. General Description

This algorithm is different from the other two because it is based on another type of linear bandit algorithm.
In this case the algorithm is built to explore as much as possible the rewards of all the arms to make sure the arm chosen is right. For how it is built we should have a worse estimation of theta, but a better result in the end to find the best arm.

6.3.2. Inspiration and papers

This algorithm is using the basis of the Gamification of Pure Exploration for Linear Bandits of Remy Degenne et all [8]. As in the previous case the algorithm is meant to work with a linear bandit environment, in our case we have a bi linear one. But the original algorithm is working with more θ and we made them with all the combination of the w and the given $y \in Y$ of players two.

6.3.3. Information needed

In this model we know in advance the number of arms $x \in X$ available to player one and $y \in Y$ available to player two. But we know the composition only of the arms $x \in X$ available to player one, and we do not know the composition of the arms $y \in Y$ available to player two. The setting of the world represented by a matrix is unknown.

6.3.4. Design Matrix and Norm

w is the probability distribution to play the arms.

For any $w \in (\mathbb{R}^+)^A$ the design matrix is $V_w := \sum_{a \in A} w^a a a^\top$.

They define $||x||_V := \sqrt{x^\top V x}$ for $x \in \mathbb{R}^d$ and a symmetric positive matrix $V \in R^{dxd}$. Note that it is a norm only if V is positive definite.

It is denoted by $\sum_K$ the probability simplex of dimension $K - 1$ for all $K \geq 2$.

6.3.5. Sion's Minimax Theorem

The algorithm inverts the order of the move of the players. This is possible thanks to Sion's Minimax theorem which is a generalization of the John Von Neumann minimax theorem. The Sion's Minimax theorem says:

Let X be a compact convex subset of a linear topological space and Y a convex subset of a linear topological space. If f is a real-valued function on $X * Y$ with:

- $f(x, \cdot)$ upper semicontinuous and quasi-concave on $Y, \forall x \in X$;
- $f(\cdot, y)$ lower semicontinuous and quasi-convex on X, $\forall y \in Y$

then,

$$\min_{x\in X} \sup_{y\in Y} f(x,y) = \sup_{y\in Y} \min_{x\in X} f(x,y).$$

6.3.6. Estimation of theta

There is a regularization parameter $\eta > 0$. The regularized least square estimator for the parameter $\theta \in M$ at time t is $\tilde{\theta} = (V_{N_t} + \eta I_d)^{-1} \sum_{s=1}^{t} Y_s a_s$ where I_d is the identity matrix. By convention $\tilde{\theta}_0 = 0$.

6.3.7. AdaHedge [7]

AdaHedge is a part of the algorithm that is responsable to find the next arm to play. Adahedge [7] is an algorithm that is adaptive to the sum of the squared L_∞ norm of the losses, without any prior information on the range of the losses. It is of the class of follow the regularized leader, this means that the algorithm at each time step it will play the minimizer of the sum of the past losses plus a time-varying regularization.

In the Hedge setting, prediction proceeds in rounds. At the start of each round $t = 1, 2, ...$, a learner has to decide on a weight vector $w_t = (w_{t,1}, ..., w_{t,K}) \in R$ K over K "experts". Each weight $w_{t,k}$ is required to be non negative, and the sum of the weights should be 1. Nature then reveals a K-dimensional vector containing the losses of the experts '$l_t = (l_{t,1}, ..., l_{t,K}) \in R$ K. Learner's loss is the dot product $h_t = w_t$'t , which can be interpreted as the expected loss if Learner uses a mixed strategy and chooses expert k with probability $w_{t,k}$. We denote aggregates of per-trial quantities by their capital letter, and vectors are in bold face. Thus, $L_{t,k} = l_{1,k} + ... + l_{t,k}$ denotes the cumulative loss of expert k after t rounds, and $H_t = h_1 + ... + h_t$ is Learner's cumulative loss (the Hedge loss).

We define the Hedge or exponential weights strategy as the choice of weights $w_{t,k} = w_{1,k} e{-}\eta Lt{-}1, kZ_t$ where $w_1 = (1/K, \ldots, 1/K)$ is the uniform distribution, $Zt = w_1 e^{-\eta L_{t-1}}$ is a normalizing constant, and $\eta \in (0, \infty)$ is a parameter of the algorithm called the learning rate. If $\eta = 1$ and one imagines $L_{t-1,k}$ to be the negative log-likelihood of a sequence of observations, then $w_{t,k}$ is the Bayesian posterior probability of expert k and Z_t is the marginal likelihood of the observations. Like in Bayesian inference, the weights are updated multiplicatively, i.e., $w_{t+1,k} \propto w_{t,k}e^{-\eta l_{t,k}}$. The loss incurred by Hedge in round t is $h_t = w_t\ l_t$, the cumulative Hedge loss is $H_t = h_1 + \ldots + h_t$, and our goal is to obtain a good bound on HT.

6.3.8. Arm to play

At each step the Adahedge gives a probability to play a given arm to all the available arm. But in our case we have to chose only one arm to pull and we can not take fragmental reward. So we try to compensate this taking track of two vector: a vector that saves the sums of all the distribution of probability to play an arm, and another vector that saves the actual number of times was played in the past. At each iteration we choose to play the arm which has the biggest difference between the numbers of times should have been played and the actual number of times it was played.

6.3.9. Pull arms

We pull the arms chosen, we take the reward associated and save the reward and the number of the arm pulled by incrementing by one in the vector that is saving the number of pulling for each arms. Then we make the complement of the reward (in this way if an arm is good has a low number associated and if it is bad has a big number associated) and we feed the AdaHedge algorithm with it. In order to work properly AdaHedge has to consider the reward as a loss and not as a gain so the complementary of the reward is a good way to feed the algorithm, but to be able to do so we should have an estimation of the maximum values of the reward possible. This is quite simple if we keep all the value of the arms in the range of $[0, 1]$ and the same for the matrix. We also have to normalize the error, because AdaHedge works better if the error is always between $[0, 1]$. So to make it possible we make an estimation of the biggest value of the rewards, take a slightly bigger number and use it to compute the error $error = (ChosenValue - reward)/ChosenValue$.

6.3.10. Algorithm

6.3.11. Consideration

This algorithm is the more complex of the three and it is based on the idea of trying to explore as much as possible all the arms. In this way it should be more robust to noises and errors due to different arms with a reward very close to the best arm.

Algorithm 6.3 GAM-g algorithm

```
Require: : $x \in X$ arm available to player one
  for $t = 1, ...T$ do
    Agent play: Get $w_t$ from $L_w^{i_t}$ and update $W_t = W_{t-1} + w_t$
    Response Nature: Best response for nature $y_i \in Y$
    Gains: Feed learner with $g_t(w) = \sum_{a \in A} \frac{w^a U_t^a}{2}$
    Pull: $a_t \in argmin_{a \in A} N_{t-1}^a - W_t^a$
  end for
```

But it has some disadvantages, using as a metrics the number of steps it is the slower of the three and if we compute the loss of arm pulled during the exploring phases it is for sure the worst one because it still pulls arm sub optimal.

On the other hand if we consider as a metrics the speed of the algorithms, it is the faster on the three, this is due to the fact that it it the only algorithm that do not require to invert a matrix, that it is the most expensive operation of the others two algorithms.

The last consideration make it very versatile to use in case we want to try it on very large arm vectors, but we do not have any experiments on that regards.

7 | Experiments

This section is divided in two parts. In the first one we are showing the results of the first group of experiments that consist on the test of the three basics algorithms using as arms a 3 dimension vector. The dimension is quite small to be able to run multiple experiments in a reasonable time.

The second type of experiments utilize a real simulation game. In this part we are going to describe the chosen game, the setting of the environments and how we trained the agents. In the end we are going to describe how the experiments are ran and the results.

7.1. Vectors

In this section we describe the main experiments we ran and the results. The experiments are ran using all the three algorithm on the same sets of data, that consist of four arms of dimension three for player one, three arms of dimension three for player two and a matrix 3 by 3 for the environment. All the arms and the matrix are generated by a random generator of numbers in the rage $[0, 1]$ in this way the result do not explode in big numbers and we can keep a maximum threshold to run properly the third algorithm. We had 18 different experiment with the setting described above, here we analyze 8 of them.The not reported experiment are quite similar to the fourth one (as you will see later, all three algorithms choose the correct arm from the beginning and the error for the reward goes smoothly to zero). All experiments are ran on the time spawn of 800 and are repeated 50 times to try to reduce the variance of a single run. The graphs that we are going to show represent the percentage of times each algorithms ha selected the correct answer in each time steps and the error it is making in that specific time step choosing that specific arm instead of the best answer. Before running the algorithm, we pulled each arm two times, in order to make the algorithms work an make possible the estimation of the θ and the rewards of each arms.

7.1.1. Experiment one

This is the result of the majority of the experiments. We have 10 more experiments with a similar trend in the error and in the probability of finding the best arm.

Specifically those are the value of the arms available to player one:

First arm player one: $[0.8285874164841791, 0.3204776292471202, 0.9167182338155841]$

Second arm player one: $[0.12329131472862598, 0.17120593541128581, 0.3568780650352187]$

Third arm player one: $[0.1245065255563258, 0.620603147326825, 0.12507361420907293]$

Fourth arm player one: $[0.05800837578412954, 0.28624549598882365, 0.9694699828038708]$

This is the value of the matrix that is representing the environment:

$$\begin{bmatrix} 0.570627679293725 & 0.6544789765162096 & 0.3093380096758692 \\ 0.8902058387476882 & 0.08743732065752974 & 0.4591299543963444 \\ 0.7876284760248763 & 0.897208002570129 & 0.4786878357904021 \end{bmatrix}$$

Those are the arms available to player two:

First arm player two: $[0.8762234180936752, 0.2961640430028448, 0.5681266047173104]$

Second arm player two: $[0.09935972822382311, 0.7196206478864489, 0.327357699381271]$

Third arm player two: $[0.5411335813585807, 0.5753941013510298, 0.7952410436811846]$

Here we can see the different reward of each arm of player one combined with each arm of player two:

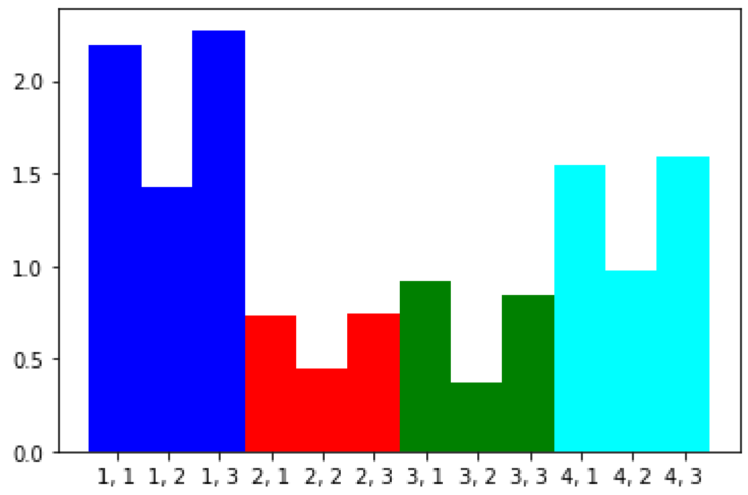

Figure 7.1: Reward of each couple (arm player one, arm player two)

As we can see arm one for player one has higher reward than the others two, also the minimum reward for arm one is higher than the maximum reward of arm two and three.

The result for the probability over the 50 experiments to find the best arm at each time step from 0 to 800 is quite simple: always correct. We have to remember that before the start of the algorithms we pull each combination of the arms (12 in this case) twice to be able to compute the estimation of theta and other variables needed for some of the three algorithm. And after seeing how far better is the best arm from the others it is not surprising the result of the experiment. As said before, in other 11 experiment the result is the same as this experiment. And also in those cases one arm is clearly better than the others.

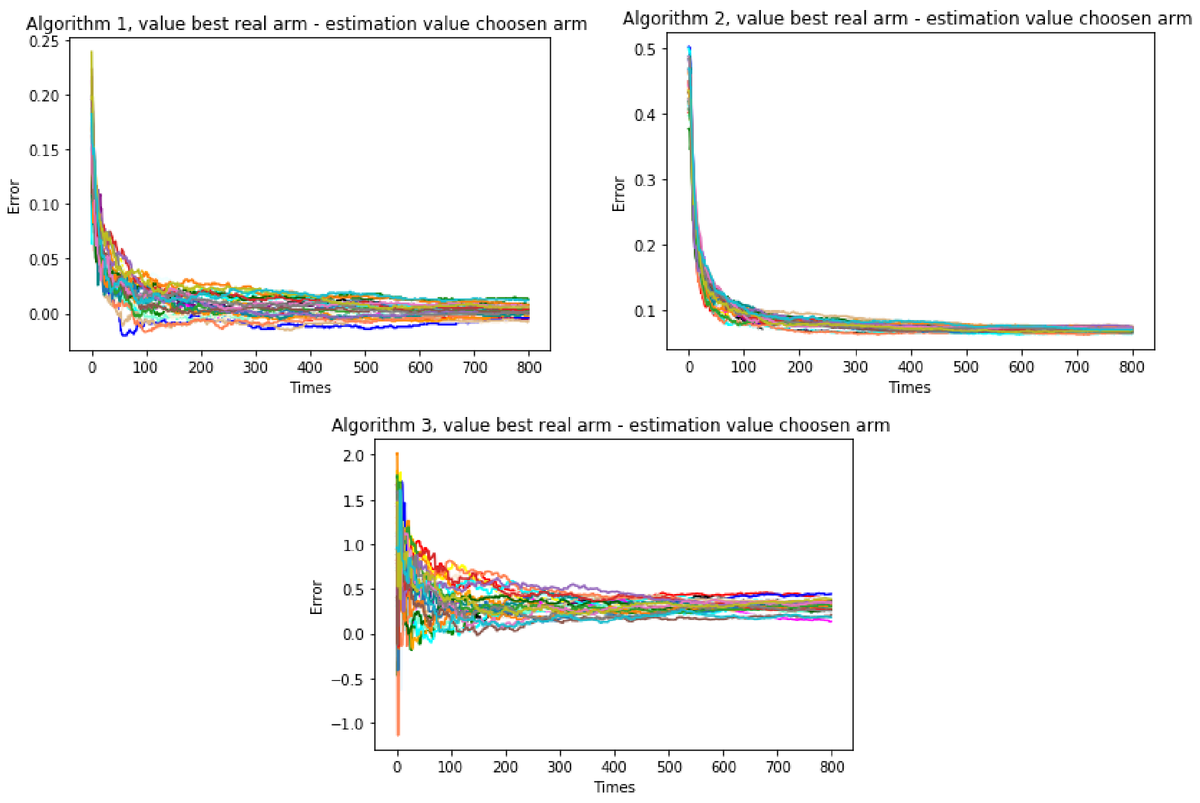

Figure 7.2: Error during time of the three algorithms. 1) Super-arm-g 2) Bilinear-g 3)GAM-g

Here we can see how all three algorithms converges to and error of 0 very fast. This behaviour is shared with others 11 experiments. We can also see how the first algorithm (super-arm-g) is the fastest and it is not surprising due to the fast that it is the only one that knows more information at the beginning. In fact knows the composition of the arms available to player two.

In figure 7.3 you can see the error of the three algorithm in the same plot. We can notice as GAM-g has more uncertainty while the other two have a pretty low error and uncertainty. Still all three algorithms converge to a number close to 0.

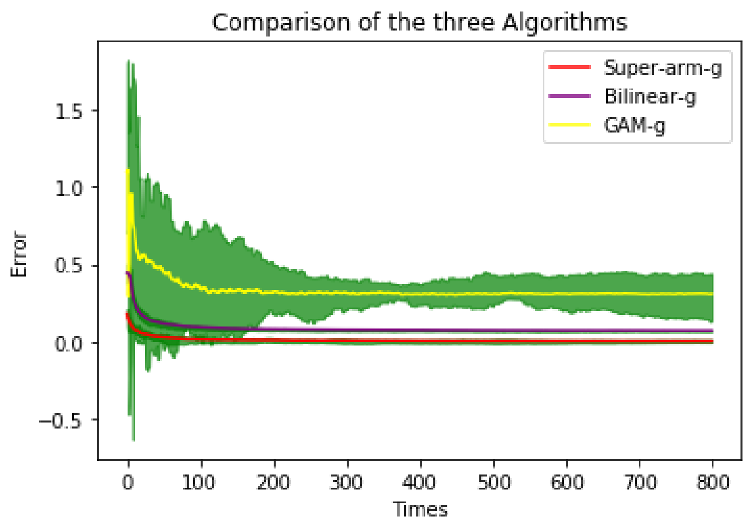

Figure 7.3: Mean error of the three algorithms and max min interval of each algorithm

7.1.2. Experiment two

Those are the value of the arms available to player one:

First arm player one: [0.6539936932649052, 0.49234274001921385, 0.9073937998343392]

Second arm player one: [0.14679488906671767, 0.7429628522780226, 0.5576982329690424]

Third arm player one: [0.8930164433455009, 0.8780462992936544, 0.9014394768882377]

Fourth arm player one: [0.2539102367562567, 0.6194135715644297, 0.3121193794381716]

This is the value of the matrix that is representing the environment:

$$\begin{bmatrix} 0.87302404451816763 & 0.21159403045606273 & 0.28918565354173686 \\ 0.5189626386885673 & 0.4849994006831272 & 0.9629953198577507 \\ 0.17670422753928094 & 0.2532554211216236 & 0.9072025363404456 \end{bmatrix}$$

Those are the arms available to player two:

First arm player two: [0.18050836984570318, 0.6401278349403822, 0.48281685719384904]

Second arm player two: [0.5890871013415527, 0.26856705274431947, 0.17129466008906424]

Third arm player two: [0.549236398976063, 0.4311610694683382, 0.9076721344539929]

Here we can see the different reward of each arm of player one combined with each arm of player two:

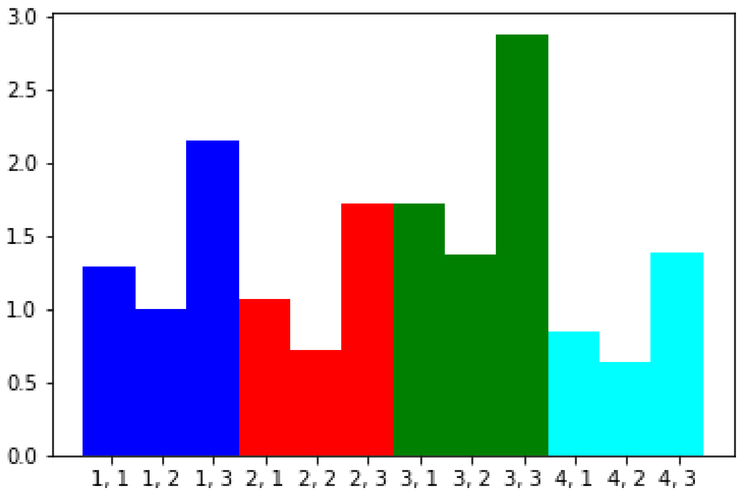

Figure 7.4: Reward of each couple (arm player one, arm player two)

It is quite clear that arm 3 is the best choice for player one, while arm 2 is the best option for player two. Still min reward of the best arm for player one is not above the max reward of the other arm for player one. All three algorithms from the beginning of the iteration choose the best arm.

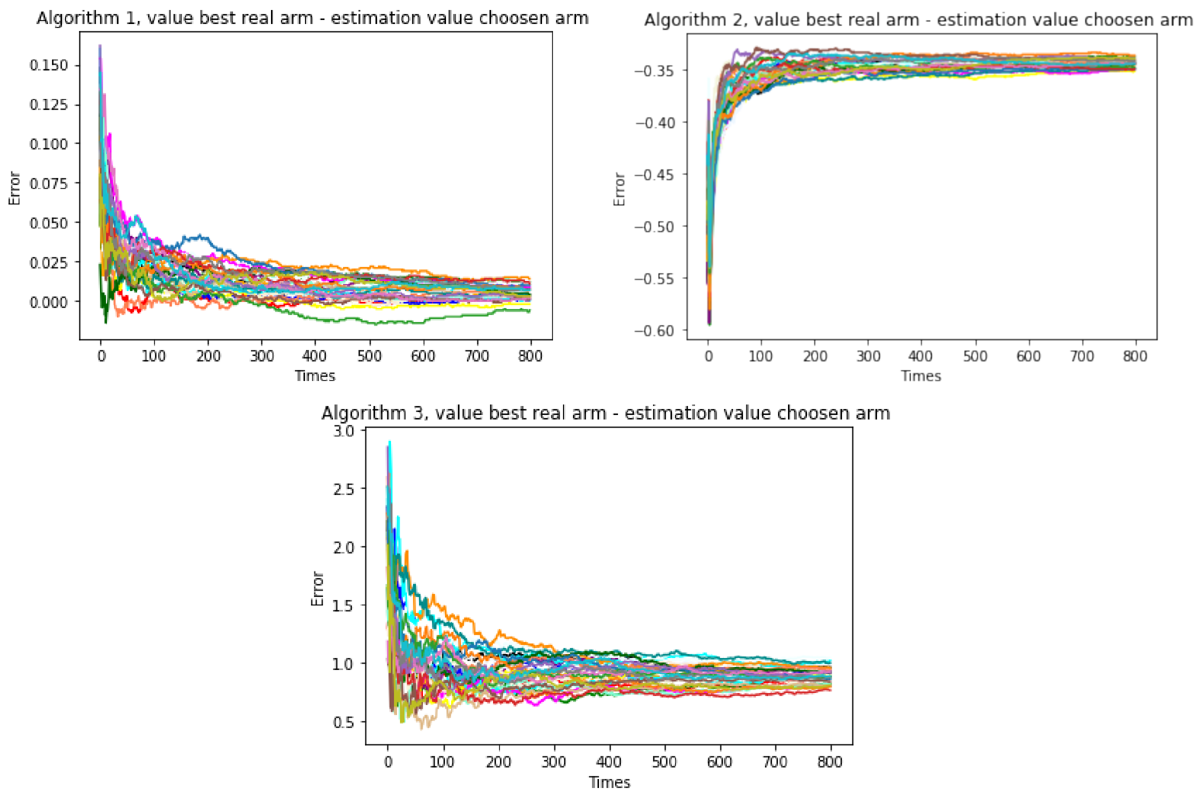

Figure 7.5: Error during time of the three algorithms. 1) Super-arm-g 2) Bilinear-g 3)GAM-g

As we can see in figure 7.5 and 7.6 the estimation of the reward for algorithm three (GAM-g) is not very good and converge to a value a bit less than 1. Algorithm 2 (Bilinear-g) makes a bit of a overestimation of the reward while algorithm one (Super-arm-g) has a fairly good estimation from the beginning, and again it is not surprising since it is the algorithm with more information.

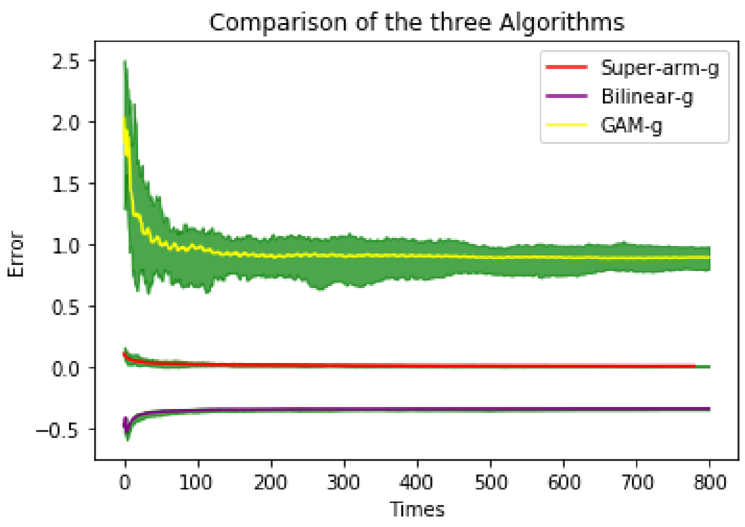

Figure 7.6: Mean error of the three algorithms and max min interval of each algorithm

7.1.3. Experiment three

Those are the value of the arms available to player one:

First arm player one: $[0.559387845693711, 0.030592908648924944, 0.2602060725485499]$

Second arm player one: $[0.8726665974325338, 0.5560948238820561, 0.6097581768365081]$

Third arm player one: $[0.10013745928984152, 0.5112722762337724, 0.24118477460825727]$

Fourth arm player one: $[0.23939263641098318, 0.1583657496121632, 0.4930573887885642]$

This is the value of the matrix that is representing the environment:

$$\begin{bmatrix} 0.4487646935680837 & 0.7281095467996116 & 0.8342454475052776 \\ 0.9563963230397624 & 0.7493372132409527 & 0.7860515446124889 \\ 0.31535516141192754 & 0.6279740002955373 & 0.9276369320718432 \end{bmatrix}$$

Those are the arms available to player two:

First arm player two: $[0.2736098621053312, 0.25135138415013714, 0.5633984772761889]$

Second arm player two: $[0.9342597204077243, 0.856543770581789, 0.9465716163540253]$

Third arm player two: $[0.4145019168719918, 0.6076227550290373, 0.4356584182755976]$

Here we can see the different reward of each arm of player one combined with each arm of player two:

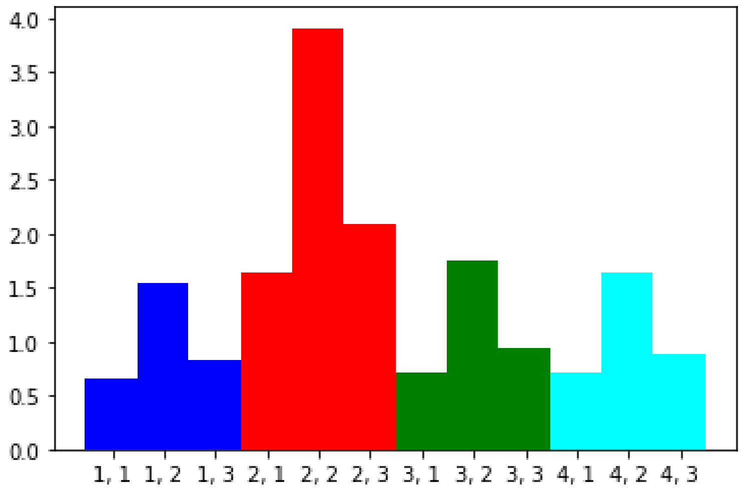

Figure 7.7: Reward of each couple (arm player one, arm player two)

It is quite clear that arm 2 is the best choice for player one, while the others arm are very similar in performances. We can notice that arm one and three for player two are quite similar, adding probably some problem in finding an equilibrium. As we can notice from figure 7.8 the third algorithm (GAM-g) in finding the best arm. Still it finds after 100 times the correct answer in all experiments. Figure for the other two algorithms are not reported since they are always correct from the beginning.

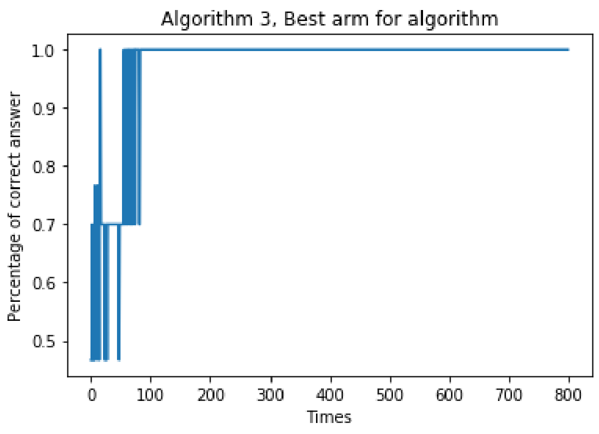

Figure 7.8: Percentage of choosing best arm at each time step GAM-g

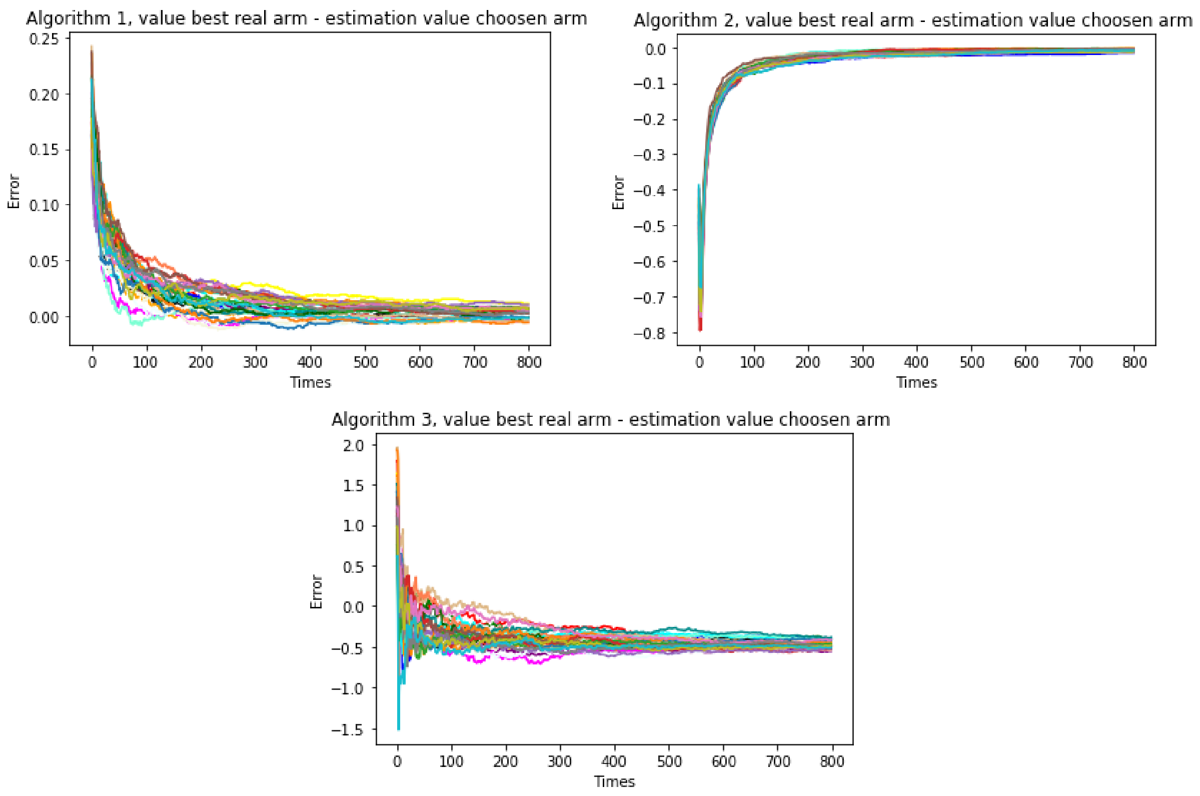

Figure 7.9: Error during time of the three algorithms. 1) Super-arm-g 2) Bilinear-g 3)GAM-g

As we can see in figure 7.9 and 7.10 the estimation of the reward for algorithm three (GAM-g) is very good and converge to 0.

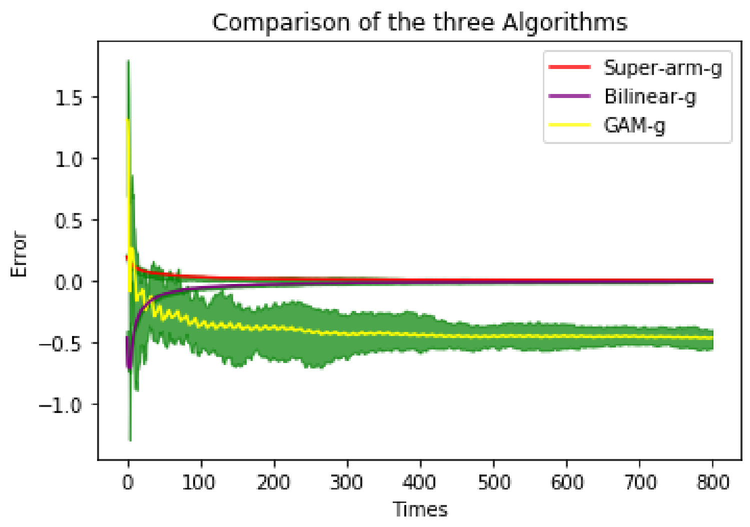

Figure 7.10: Mean error of the three algorithms and max min interval of each algorithm

7.1.4. Experiment four

Those are the value of the arms available to player one:

First arm player one: $[0.26404238493533333, 0.1619612217622196, 0.6246053878659477]$

Second arm player one: $[0.7465741715346589, 0.061099245088389376, 0.2662259155112924]$

Third arm player one: $[0.9409195196945171, 0.8173711850216506, 0.14235450114879167]$

Fourth arm player one: $[0.6060940205761899, 0.40093675510616444, 0.28020334213618725]$

This is the value of the matrix that is representing the environment:

$$\begin{bmatrix} 8 & 1 & 6 \\ 0 & 0.5 & 10 \\ 1 & 4 & 2 \end{bmatrix}$$

Those are the arms available to player two:

First arm player two: $[0.7114237342739692, 0.053097447451999624, 0.7724254052002553]$

Second arm player two: $[0.27858763107952933, 0.41750758315774095, 0.5799484244817303]$

Third arm player two: $[0.45235760695948235, 0.09025751959471473, 0.6250558612477992]$

Here we can see the different reward of each arm of player one combined with each arm of player two:

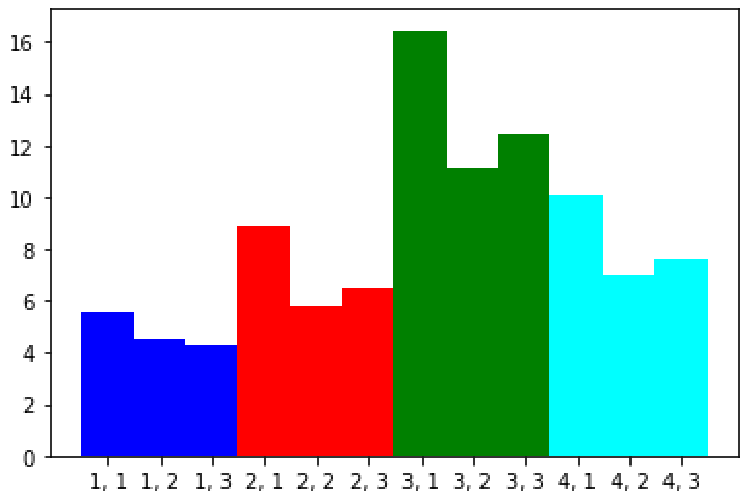

Figure 7.11: Reward of each couple (arm player one, arm player two)

As shown in figure 7.11 arm three is the best arm for player one, while arm two and three are very close for player two. All three algorithms do not find any difficulties in finding the best arm since the beginning. so we will not show the figure of the percentages of choosing the best arm since it is a straight line on 1.

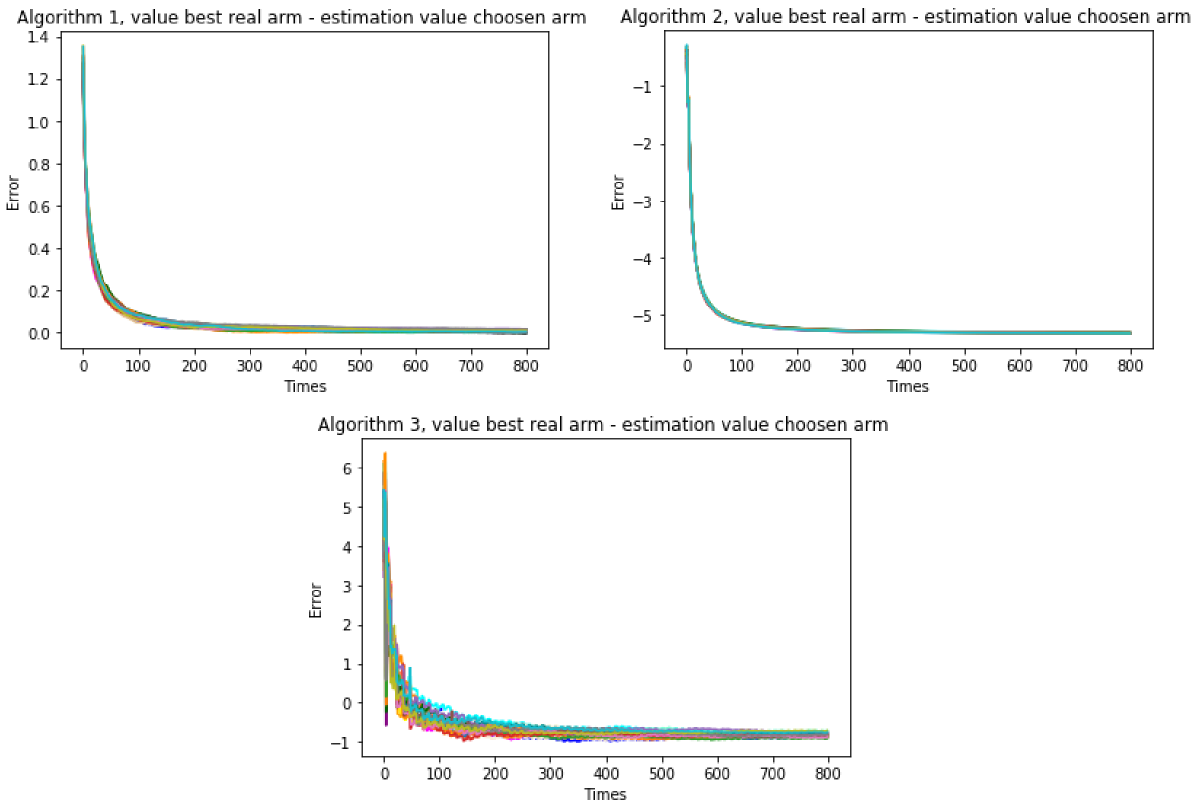

Figure 7.12: Error during time of the three algorithms. 1) Super-arm-g 2) Bilinear-g 3)GAM-g

As we can see in figure 7.12 and 7.13 the estimation of the reward for algorithms one (Super-arm-g) and three (GAM-g) is very good and converge both to 0 quite fast. On the other hand algorithm two (Bilinear-g) has a pretty large over estimation of the reward of all the arms. This over estimation do not make the algorithm choose a different best arm, but it is important to notice that.

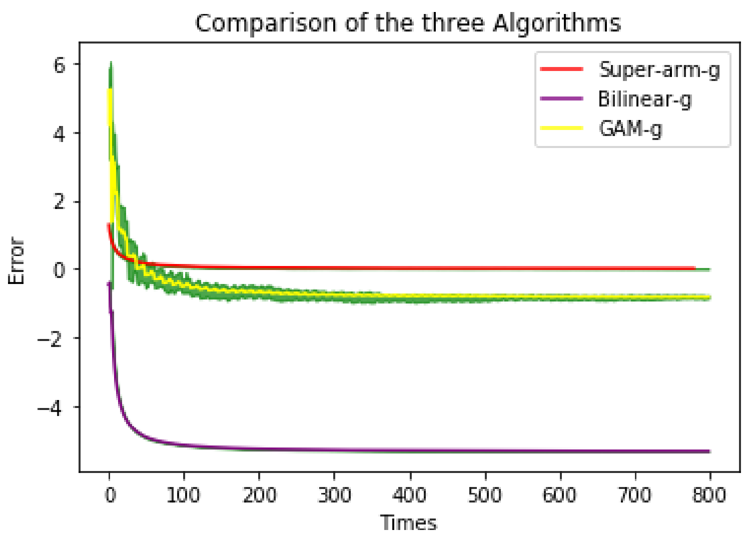

Figure 7.13: Mean error of the three algorithms and max min interval of each algorithm

7.1.5. Experiment five

Those are the value of the arms available to player one:

First arm player one: [0.7400797306398084, 0.576547832665651, 0.7349830663582422]

Second arm player one: [0.345067231955826, 0.9156681583876762, 0.6823912524346282]

Third arm player one: [0.5638993306859795, 0.8748338736453104, 0.18485633496104792]

Fourth arm player one: [0.673719565597578, 0.1725870996908213, 0.7472676319688508]

This is the value of the matrix that is representing the environment:

$$\begin{bmatrix} 0.10491443939828526 & 0.6793030919801211 & 0.7194757379073787 \\ 0.13546896362398142 & 0.12173364585448296 & 0.6991816425464136 \\ 0.7898135475171382 & 0.21428911809375328 & 0.7623852971358278 \end{bmatrix}$$

Those are the arms available to player two:

First arm player two: [0.7233068388545373, 0.8898790946755905, 0.45671865937762546]

Second arm player two: [0.36975365980964037, 0.5782920189315391, 0.3785203504783837]

Third arm player two: [0.8700421843916122, 0.08522705600354452, 0.8195182497977863]

Here we can see the different reward of each arm of player one combined with each arm of player two:

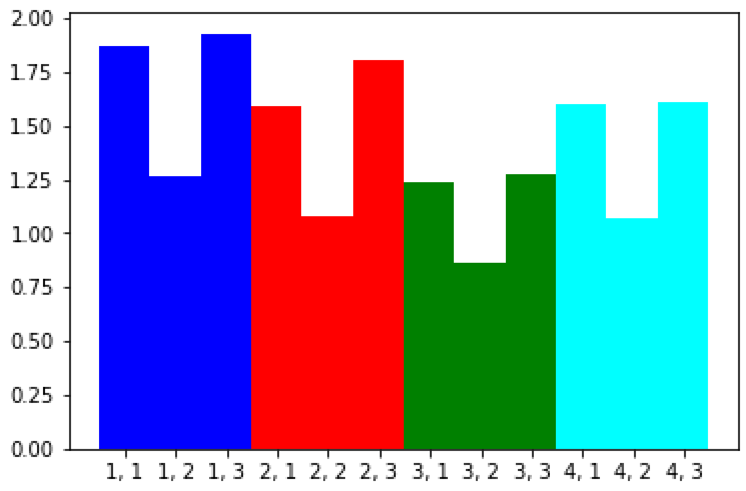

Figure 7.14: Reward of each couple (arm player one, arm player two)

We can notice from Figure 7.14 as the min reward of all four arms for player one has a very similar reward, especially arm 1 that is the best arm and 2. As we can notice from figure 7.15 all three algorithms finds some difficulties in finding the best arm. All three still have a percentage quite high from the beginning and stays above 0.7/1.0 which is a good result and all three converges to the optimal arm in the end.

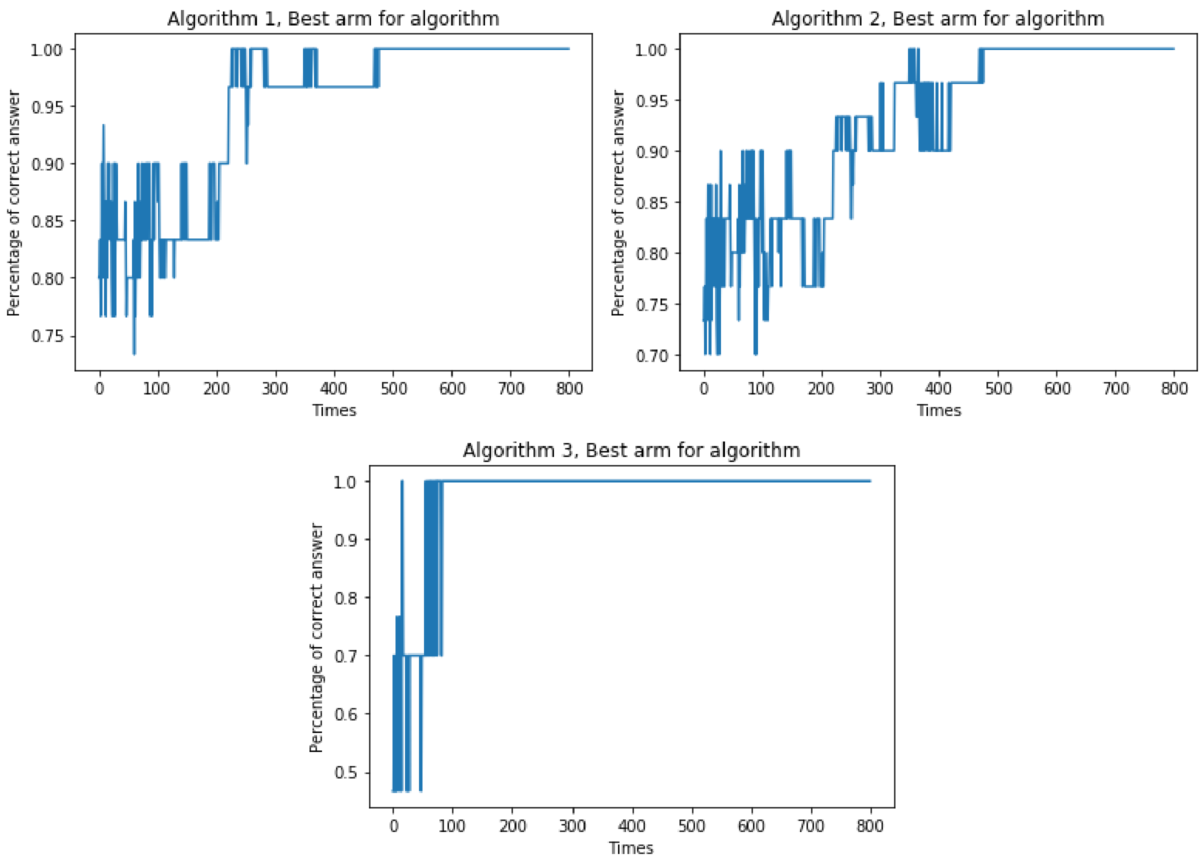

Figure 7.15: Percentage of choosing best arm at each time step 1) Super-arm-g 2) Bilinear-g 3)GAM-g

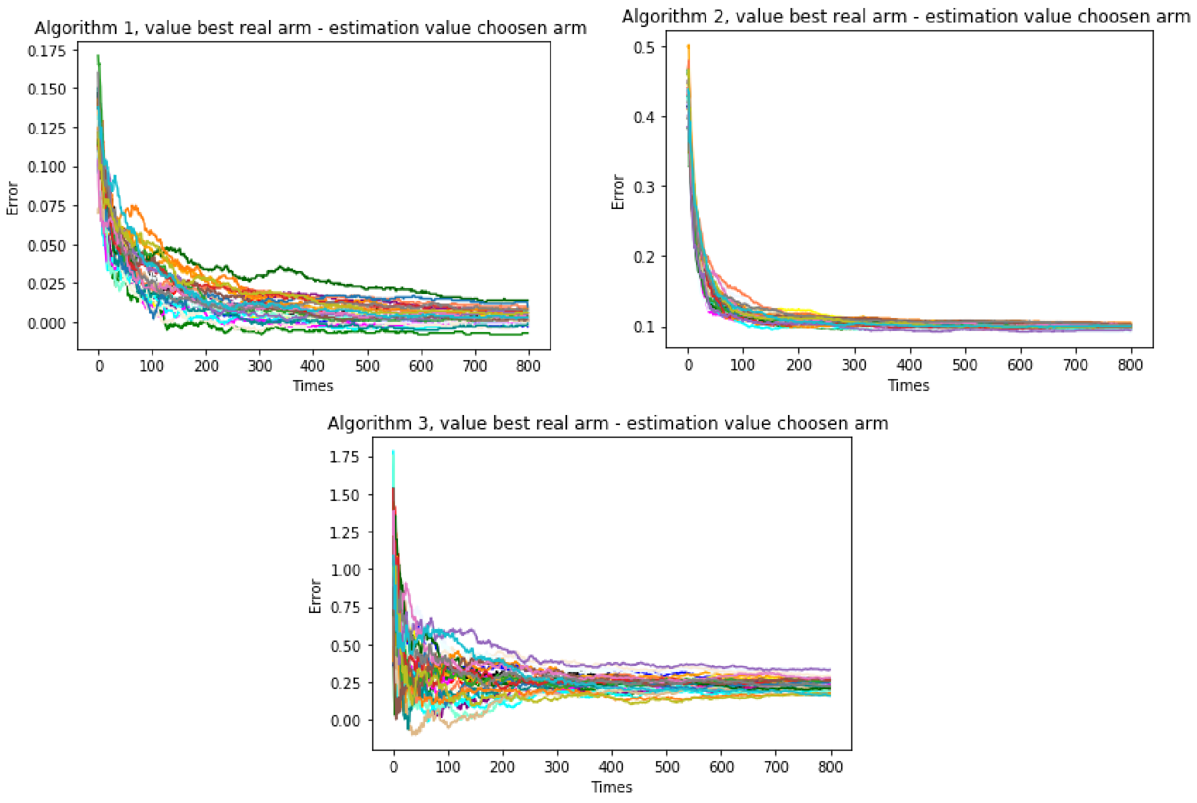

Figure 7.16: Error during time of the three algorithms. 1) Super-arm-g 2) Bilinear-g 3)GAM-g

As we can see in figure 7.16 and 7.17 the estimation of the reward for all three algorithm is quite correct even if they struggle to understand which is the best arm. This make sense since the second best arm is very close as a reward to the best arm.

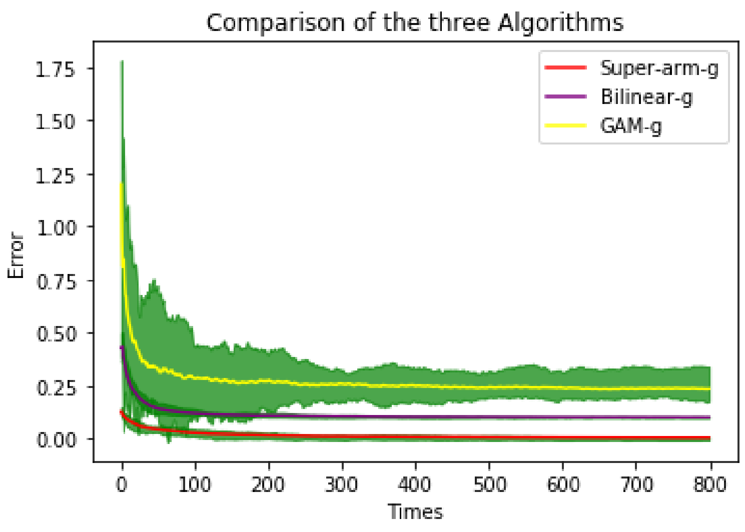

Figure 7.17: Mean error of the three algorithms and max min interval of each algorithm

7.1.6. Experiment six

Those are the value of the arms available to player one:

First arm player one: $[0.3783790770067742, 0.04509920943528589, 0.2206567345322612]$

Second arm player one: $[0.8374862507164998, 0.3402368919038926, 0.6487244129750209]$

Third arm player one: $[0.006492839558645 88, 0.15139404302157267, 0.5186768417088721]$

Fourth arm player one: $[0.46950197256209913, 0.4174602853527538, 0.776731380215685]$

This is the value of the matrix that is representing the environment:

$$\begin{bmatrix} 0.24999736016010754 & 0.42679444804934 22 & 0.42104221154416 91 \\ 0.7482545665289171 & 0.3400437268893406 & 0.88659085460408 \\ 0.17531194853943322 & 0.29346489179836066 & 0.7548200811931902 \end{bmatrix}$$

Those are the arms available to player two:

First arm player two: $[0.4697619288466751, 0.8157025115607986, 0.9769181335032701]$

Second arm player two: $[0.7932210066433859, 0.5270393574027046, 0.055965226188414596]$

Third arm player two: $[0.5494841043947376, 0.5352256221973702, 0.9645188661509865]$

Here we can see the different reward of each arm of player one combined with each arm of player two:

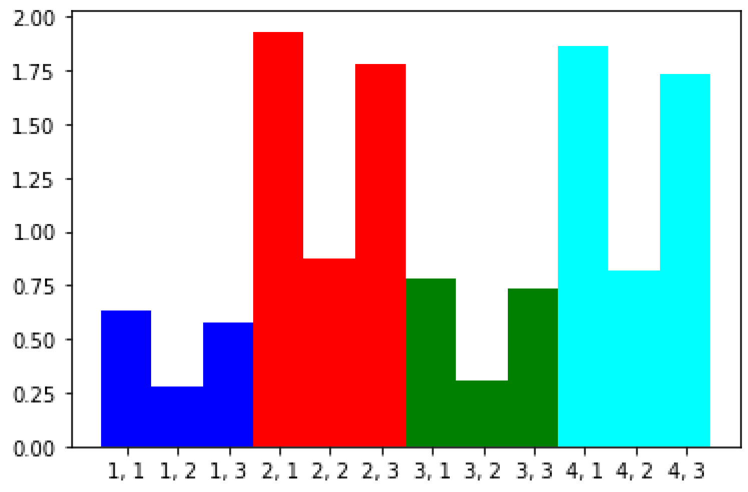

Figure 7.18: Reward of each couple (arm player one, arm player two)

We can notice from Figure 7.18 as the best arm 4 for player one and the arm 2 are very close in reward. So we should expect some type of error in the percentage of choosing the best arm, but a low error in the difference between the value of the best arm and the estimation of the chosen arm at each time step.

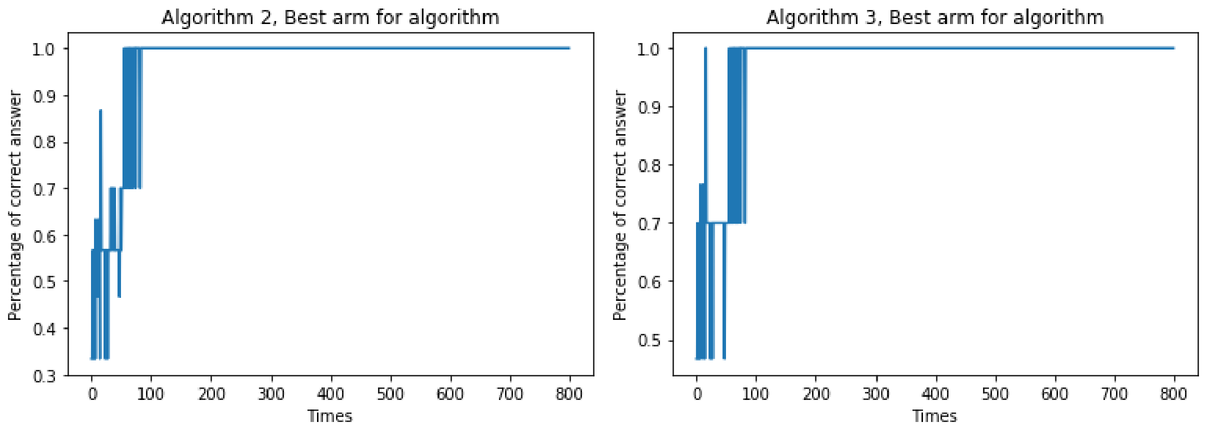

Figure 7.19: Percentage of choosing best arm at each time step 2) Bilinear-g 3)GAM-g

We can see in Figure 7.19 as in algorithm Bilinear-g and GAM-g there are some trouble in finding the exact arm at the beginning, but after 100 iteration both start to have a high percentage of success and about around 150 the percentage goes to 1.0. Algorithm Super-arm-g have a percentage of success of 1.0 from the beginning.

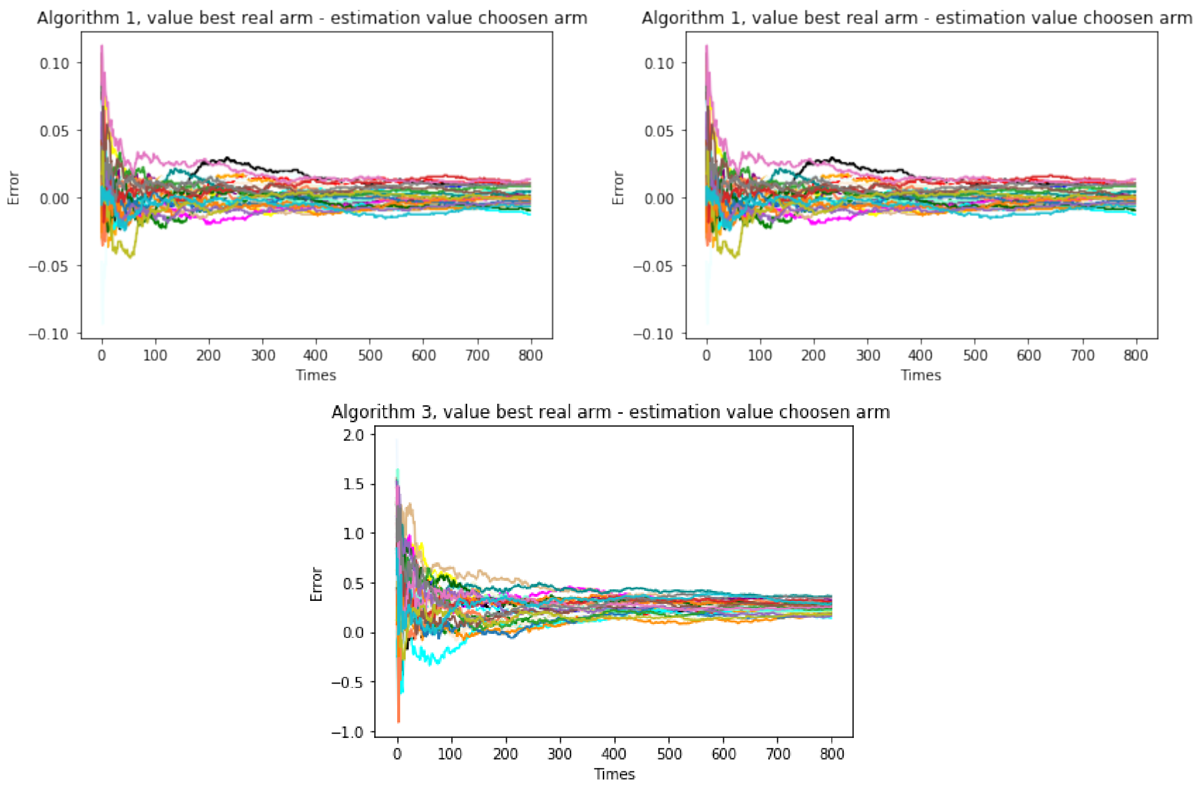

Figure 7.20: Error during time of the three algorithms. 1) Super-arm-g 2) Bilinear-g 3)GAM-g

As we can see in figure 7.19 and 7.20 the estimation of the reward for all three algorithm is quite correct even if they struggle to understand which is the best arm. This make sense since the second best arm is very close as a reward to the best arm.

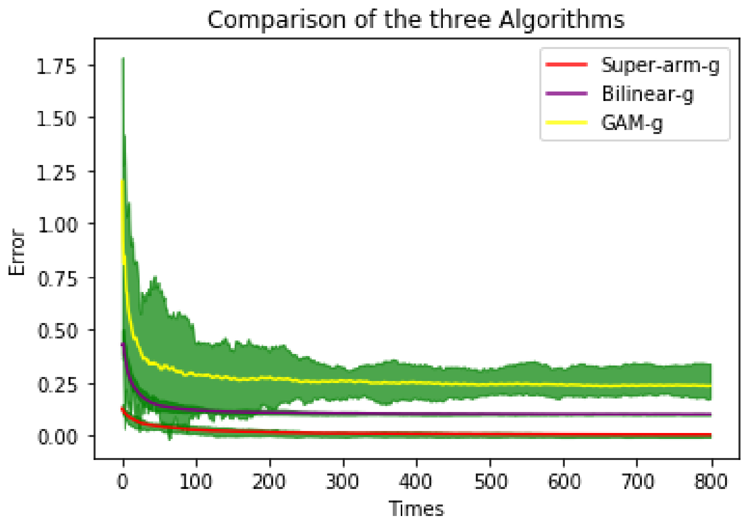

Figure 7.21: Mean error of the three algorithms and max min interval of each algorithm

7.1.7. Experiment seven

Those are the value of the arms available to player one:

First arm player one: $[0.34323564827916286, 0.9024510983869803, 0.9345356261390722]$

Second arm player one: $[0.6126493651904283, 0.710929664774286, 0.06606059076429249]$

Third arm player one: $[0.926562397890643, 0.13450860002936138, 0.06930100202896383]$

Fourth arm player one: $[0.24382941773891098, 0.5239976778455409, 0.9411197869692752]$

This is the value of the matrix that is representing the environment:

$$\begin{bmatrix} 0.54460658248501 & 0.12226479152463576 & 0.04771085220907545 \\ 0.7437617158394552 & 0.6133099286844137 & 0.05940883176853251 \\ 0.43099931915931433 & 0.27857358639017915 & 0.01961790785741513 \end{bmatrix}$$

Those are the arms available to player two:

First arm player two: $[0.5157625488939386, 0.602027945582136, 0.407083088446776]$

Second arm player two: $[0.9257268390783879, 0.41324896612767725, 0.16661333614624652]$

Third arm player two: $[0.4420073353654981, 0.33265944053850793, 0.8144820231217136]$

Here we can see the different reward of each arm of player one combined with each arm of player two:

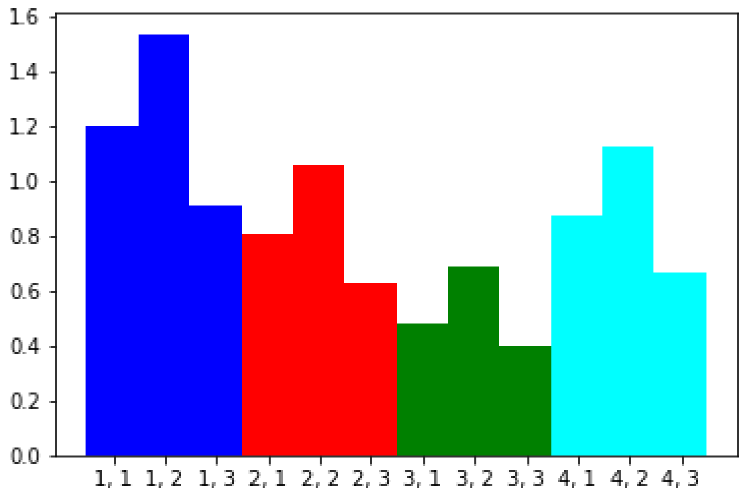

Figure 7.22: Reward of each couple (arm player one, arm player two)

We can notice from Figure 7.22 as the best arm is 1 for player one and the second best arm is 4. The difference of reward is not big, but they are not very close either.

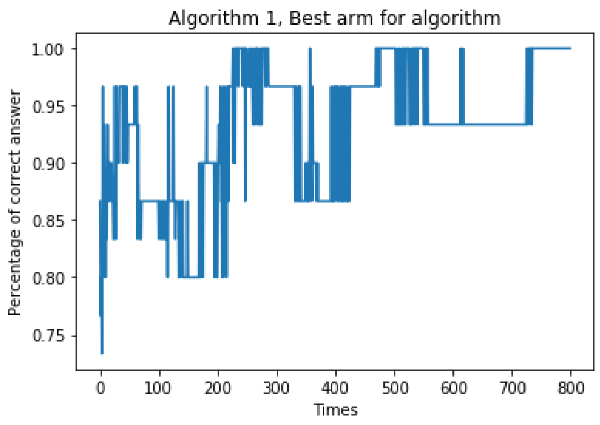

Figure 7.23: Percentage of choosing best arm at each time step 2) Bilinear-g 3)GAM-g

We can see in Figure 7.23 as Super-arm-g have some trouble in finding the exact arm at the beginning and also at the end of the iteration, but the percentage of success is quite high, being above 0.85. This means that there are some cases in which the convergence is hard, but those cases are not frequent. Other two algorithms have a percentage of success of 1.0 from the beginning.

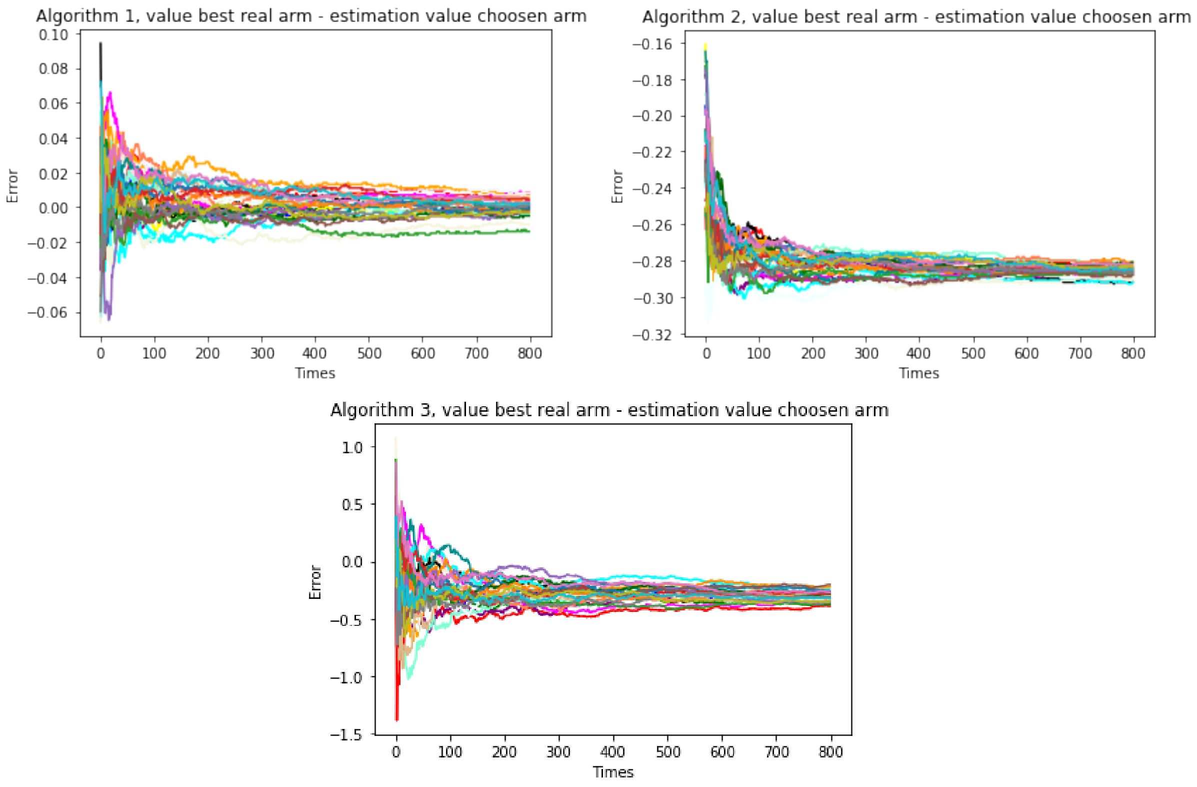

Figure 7.24: Error during time of the three algorithms. 1) Super-arm-g 2) Bilinear-g 3)GAM-g

As we can see in figure 7.24 and 7.25 the estimation of the reward for algorithms Super-arm-g and GAM-g are quite correct even if Super-arm-g struggle to understand which is the best arm. The estimation for Bilinear-g is a bit an over estimation in the regard if the reward.

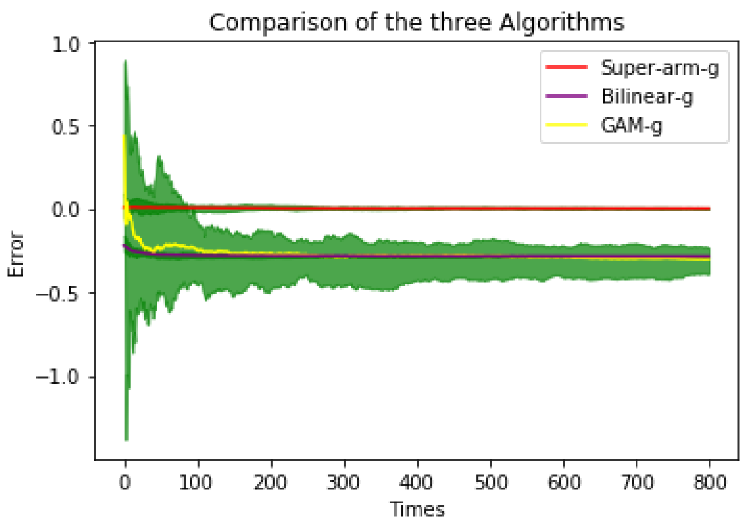

Figure 7.25: Mean error of the three algorithms and max min interval of each algorithm

7.1.8. Experiment eight

Those are the value of the arms available to player one:

First arm player one: [0.9871376736199118, 0.688121853533161, 0.5235809151333978]

Second arm player one: [0.03481376451873974, 0.34685413538499243, 0.05591468294799695]

Third arm player one: [0.8484788173613923, 0.9117151239295301, 0.43862231513255734]

Fourth arm player one: [0.6311689207509968, 0.08306431034585382, 0.49042361557796177]

This is the value of the matrix that is representing the environment:

$$\begin{bmatrix} 0.5899011009128065 & 0.5657184902992989 & 0.6054951466017203 \\ 0.5899011009128065 & 0.5657184902992989 & 0.6054951466017203 \\ 0.4974849256226015 & 0.7843065267705409 & 0.12423891479137728 \end{bmatrix}$$

Those are the arms available to player two:

First arm player two: [0.8705524572652754, 0.0619707940889308, 0.6618039107464511]

Second arm player two: [0.29313747095524345, 0.7147227382224571, 0.43880766191059795]

Third arm player two: [0.3976778652509695, 0.20770310609484777, 0.791678366091227]

Here we can see the different reward of each arm of player one combined with each arm of player two:

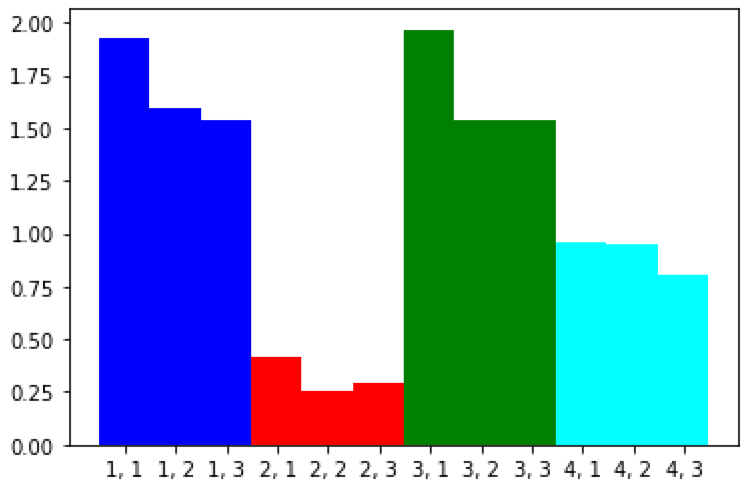

Figure 7.26: Reward of each couple (arm player one, arm player two)

We can notice from Figure 7.26 as the arm one and the arm three are very close in the rewards. It is important to notice that in arm one the best response for player two is the arm 3, while for arm three of player one the best response of player two is arm two. We can see in Figure 7.27 that all three algorithms struggle to have a 1.0 of correct answer, but it is important to notice that after few samples (50), all three algorithms have more than 0.8 percentage of correct arm chosen.

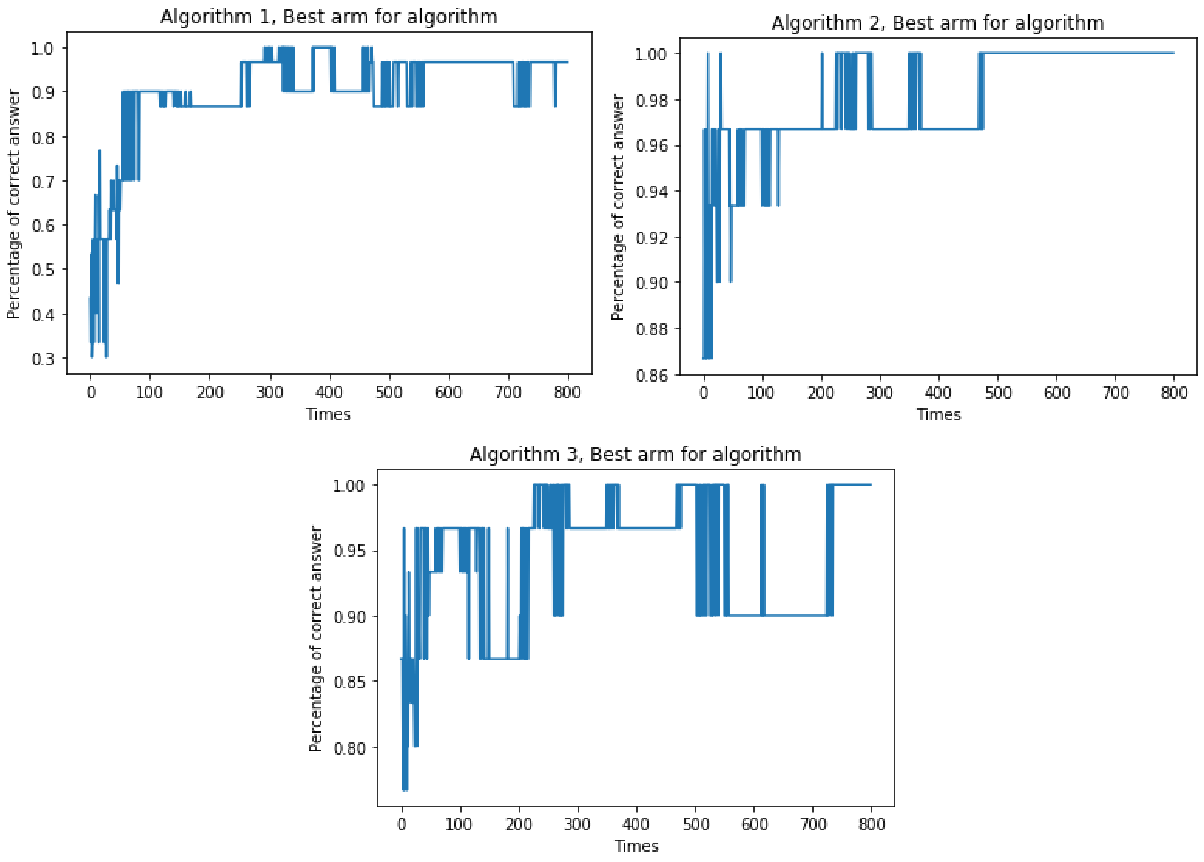

Figure 7.27: Percentage of choosing best arm at each time step 1) Super-arm-g 2) Bilinear-g 3)GAM-g

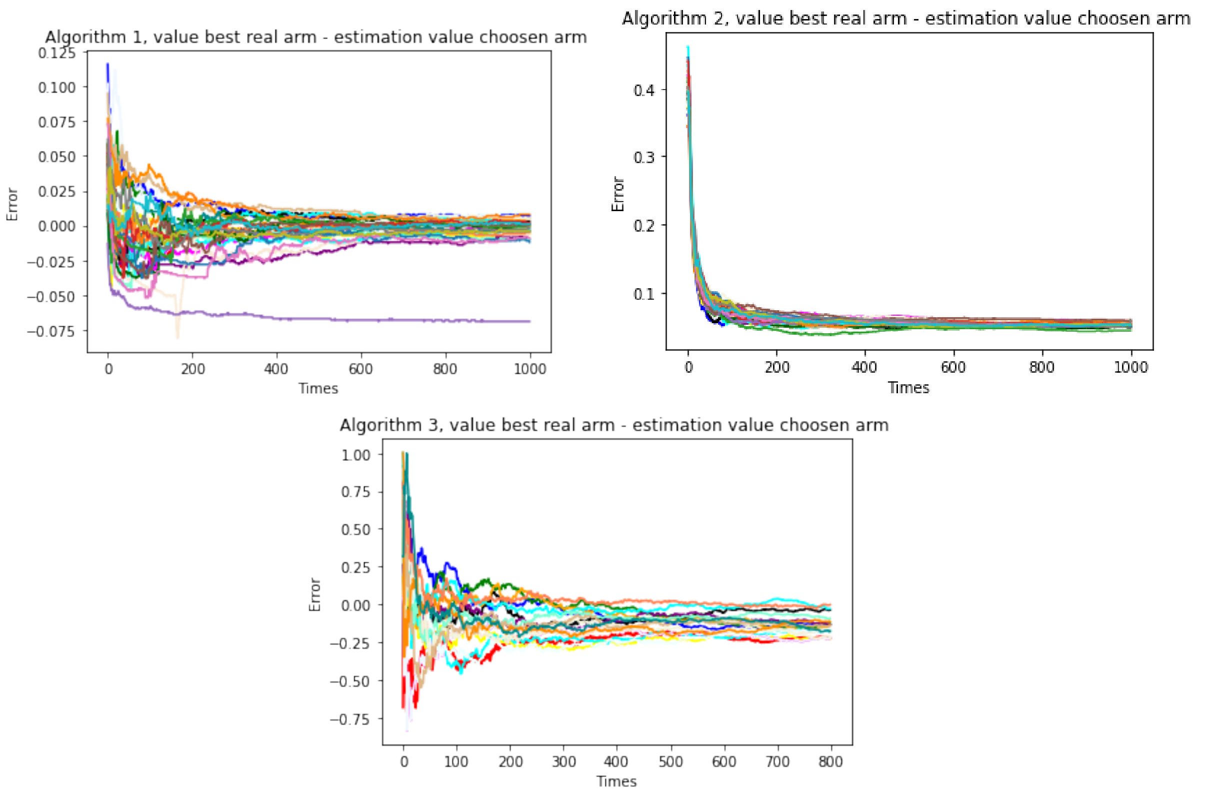

Figure 7.28: Error during time of the three algorithms. 1) Super-arm-g 2) Bilinear-g 3)GAM-g

As we can see in figure 7.28 and 7.29 the estimation of the reward for all three algorithm is quite correct even if they struggle to understand which is the best arm. This make sense since the second best arm is very close as a reward to the best arm.

7.1.9. Consideration

As expected algorithms one perform overall better than the other two. It has a better percentage of success and a better estimation of the reward of all the arms. But we have to consider that we tested with relatively small arms and still the time needed to compute this algorithm was way more than to computer the other two algorithms. This is because as the algorithm 2, we need to invert a matrix, and the size of the matrix depend on the length of the arms we are using (and we are using arms that are on the power of two of the others arms used in the others two algorithm).

Algorithm 3 (GAM-g algoritm) has overall the worse performance of the three. But we have to make some consideration, as it is intended to explore as well as possible the space of the matrix. This means that it keeps pulling arms that are not optimal to try to understand better the matrix, meaning that probably it is making a better estimation

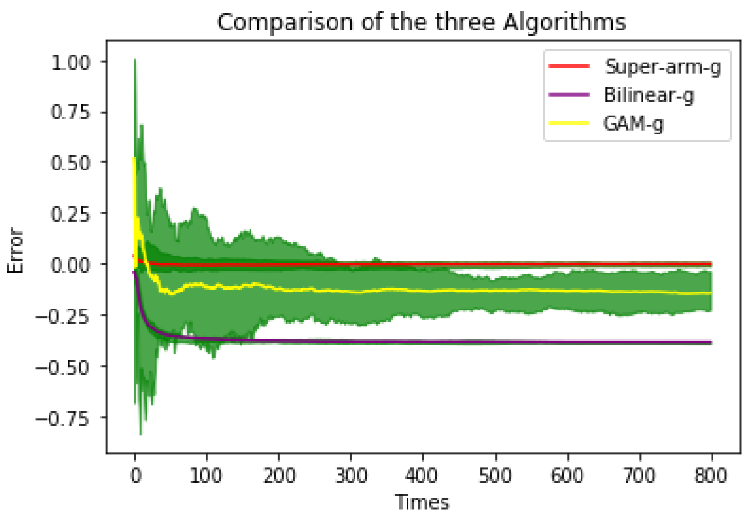

Figure 7.29: Mean error of the three algorithms and max min interval of each algorithm

of all the rewards, also of the sub optimal arms. Also we have to consider that it is the fastest experiments, this is due to the fast that it is not inverting any matrix to make the estimation of θ. Even if we are not showing the time needed to perform each iteration of the algorithms, we can at least say that this algorithm is faster, and it is a consideration to keep in mind.

Algorithm 2 (Bilinear-g algorithm) has overall a good performance even if it is not the best algorithm not in precision neither in speed of the algorithm.

In the end all three algorithm have a good performances on this type of samples, having some trouble when the reward of the best arm and a different arm is very close.

8 | Conclusions and future developments

We tackle the problem of finding the best arm identification in a linear multi armed bandit setting. Our model represent a two player zero sum game making the setting a bi linear setting in which each player has multiple strategy available to them. We are using a simulation game in which we get a reward after choosing the arm to play for the two players with some noise and we try to select the best arm to play for player one in a max min strategy. We address the problem with three slightly different approach trying to maximize the number of times the algorithm select the best arm for player one at the end of the exploration phase. Super-arm-g algorithm is the only one that needs the strategy available to player two in addition to the strategy available to player one, making it the algorithm with more information. The others two instead needs only needs the strategy available to player one to work. We tested the three algorithm with arms of 3-dimension generated at random and we confront them using as a benchmark the percentage of success in finding the best arm at the end of the exploration phase, the speed in selecting always the best arm and the error between the reward of the best arm and the estimation of reward of the selected arm in each time step. We saw how Super-arm-g algorithm performs better than other two algorithm, but it was natural due to the fact that it had more information than the other two algorithms. We saw that all the three algorithm are quite robust and overall performed well on the experiments.

In future experiments it is essential to try them out with more arms available to player one and two and also with arm and matrix of higher dimension. Probably it is interesting to have a serious comparison of the three algorithm regarding the time of execution trying to understand how much it is viable to use the Super-arm algorithm in higher dimension.

Bibliography

[1] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] S. Bubeck, R. Munos, and G. Stoltz. Pure Exploration for Multi-Armed Bandit Problems. working paper or preprint, June 2010. URL `https://hal.archives-ouvertes.fr/hal-00257454`.

[3] M. Castiglioni, A. Marchesi, N. Gatti, and S. Coniglio. Leadership in singleton congestion games: What is hard and what is easy. *Artificial Intelligence*, 277:103177, 2019.

[4] A. Celli, A. Marchesi, T. Bianchi, and N. Gatti. Learning to correlate in multi-player general-sum sequential games. *Advances in Neural Information Processing Systems*, 32, 2019.

[5] A. Celli, A. Marchesi, G. Farina, and N. Gatti. No-regret learning dynamics for extensive-form correlated equilibrium. *Advances in Neural Information Processing Systems*, 33:7722–7732, 2020.

[6] W. Chu, L. Li, L. Reyzin, and R. Schapire. Contextual bandits with linear payoff functions. In G. Gordon, D. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 208–214, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL `https://proceedings.mlr.press/v15/chu11a.html`.

[7] S. de Rooij, T. van Erven, P. D. Grünwald, and W. M. Koolen. Follow the leader if you can, hedge if you must. *Journal of Machine Learning Research*, 15(37):1281–1316, 2014. URL `http://jmlr.org/papers/v15/rooij14a.html`.

[8] R. Degenne, P. Menard, X. Shang, and M. Valko. Gamification of pure exploration for linear bandits. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2432–2442. PMLR, 13–18 Jul 2020. URL `https://proceedings.mlr.press/v119/degenne20a.html`.

[9] G. Farina, A. Marchesi, C. Kroer, N. Gatti, and T. Sandholm. Trembling-hand perfection in extensive-form games with commitment. In *International Joint Conferences on Artificial Intelligence Organization*, 2018.

[10] G. Farina, A. Celli, A. Marchesi, and N. Gatti. Simple uncoupled no-regret learning dynamics for extensive-form correlated equilibrium. *arXiv preprint arXiv:2104.01520*, 2021.

[11] T. Fiez, L. Jain, K. G. Jamieson, and L. Ratliff. Sequential experimental design for transductive linear bandits. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/8ba6c657b03fc7c8dd4dff8e45defcd2-Paper.pdf`.

[12] N. Gatti and M. Restelli. Equilibrium approximation in simulation-based extensive-form games. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '11, page 199–206, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 0982657153.

[13] K.-S. Jun, R. Willett, S. Wright, and R. Nowak. Bilinear bandits with low-rank structure. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3163–3172. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/jun19a.html`.

[14] H. Komiya. Elementary proof for Sion's minimax theorem. *Kodai Mathematical Journal*, 11(1):5 – 7, 1988. doi: 10.2996/kmj/1138038812. URL `https://doi.org/10.2996/kmj/1138038812`.

[15] T. Lai and H. 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. URL `https://www.sciencedirect.com/science/article/pii/0196885885900028`.

[16] A. Marchesi and N. Gatti. Trembling-hand perfection and correlation in sequential games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 5566–5574, 2021.

[17] A. Marchesi, M. Castiglioni, and N. Gatti. Leadership in congestion games: Multiple user classes and non-singleton actions. In *IJCAI*, pages 485–491, 2019.

[18] A. Marchesi, G. Farina, C. Kroer, N. Gatti, and T. Sandholm. Quasi-perfect stackelberg equilibrium. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2117–2124, 2019.

[19] A. Marchesi, F. Trovò, and N. Gatti. Learning probably approximately correct maximin strategies in simulation-based games with infinite strategy spaces. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '20, page 834–842, Richland, SC, 2020. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450375184.

[20] T. M. Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.

[21] M. Soare, A. Lazaric, and R. Munos. Best-arm identification in linear bandits. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL `https://proceedings.neurips.cc/paper/2014/file/f387624df552cea2f369918c5e1e12bc-Paper.pdf`.

[22] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, second edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

[23] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining.* Addison Wesley, us ed edition, May 2005. ISBN 0321321367. URL `http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0321321367`.

[24] C. Tao, S. Blanco, and Y. Zhou. Best arm identification in linear bandits with linear dimension dependency. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4877–4886. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/tao18a.html`.

[25] E. A. Viqueira, C. Cousins, E. Upfal, and A. Greenwald. Learning equilibria of simulation-based games, 2019. URL `https://arxiv.org/abs/1905.13379`.

[26] E. A. Viqueira, C. Cousins, Y. Mohammad, and A. Greenwald. Empirical mechanism design: Designing mechanisms from data. In R. P. Adams and V. Gogate, editors, *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pages 1094–1104. PMLR, 22–25 Jul 2020. URL `https://proceedings.mlr.press/v115/viqueira20a.html`.

[27] J. von Neumann. Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100: 295–320.

[28] Y. Vorobeychik and M. P. Wellman. Stochastic search methods for nash equilibrium approximation in simulation-based games. In *AAMAS*, 2008.

[29] J. yves Audibert, S. Bubeck, and R. Munos. Best arm identification in multi-armed bandits. In *In Proceedings of the Twenty-Third Annual Conference on Learning Theory*, pages 41–53, 2010.

List of Figures

Acknowledgements

Thanks to my parents for being with me all the time and for all the time and emotional effort made to help me.