POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Matematica
Scuola di Ingegneria Industriale e dell'Informazione

Solving Imperfect-Recall Games: New Representations and Algorithms.

Relatore: Prof. Nicola Gatti
Correlatore: Andrea Celli

Tesi di Laurea di:
Andrea Agostini
Matricola 862715

Anno Accademico 2017-2018

Sommario

In questa tesi studiamo

Abstract

This thesis focuses

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Research Area and Main Problem

The present work addresses problems.

1.2 Original Contributions

We start by focusing on s $\mathsf{P} = \mathsf{NP}$, even on a completely inflated tree. Then, we introduce the *hybrid column generation* algorithm which computes optimal team-maxmin equilibria with coordination devices (Celli and Gatti, 2018). We can use this

1.3 Thesis Structure

The Thesis is structured in the following way:

- In Chapter 5 we draw conclusions. We summarize the main results of our work and we propose some future developments.

Chapter 2

Problem Formulation

In this chapter we formally introduce the Multi-Armed Bandit problem with persistent rewards. After defining all the elements and details necessary to depict the interaction between the learning agent and the environment, we specify the performance measures adopted. Finally, we formalize two real-world scenarios which will be deeply analyzed during the thesis.

2.1 The MAB Persistent problem

The Mab Persistent problem that we are going to analyze in this thesis is formalized in this section. Over a finite time horizon, composed of N time instants, at each time instant t, an agent must pull (choose) an arm (action) a_t from an arm set $A = \{1, \dots, K\}$. When we pull an arm a_j at time t the environment returns a realization of a random variable $R_{j,t}$ and one of a random vector $\boldsymbol{Z_{j,t}} = (Z_{j,t,1}, \dots, Z_{j,t,Tmax})$. The vector $\boldsymbol{Z_{j,t}}$ represtens the persistency of the feedback $R_{j,t}$, meaning that each component $Z_{j,t,m}$ describes which fraction of $R_{j,t}$ the agent will collect at the m-step. At each time instant from the pull of an arm, the learner will collect a reward called *instant reward* defined as follows:

Definition 1 (Instant Reward)**.** *We define the instant reward achived at time s, consequently to the pull of the arm j at time t, as:*

$$r_{j,t,s} = R_{j,t} Z_{j,t,m}$$

Where $s \in \{t, \dots, t + T_{max} - 1\}$ *and* $m = s - t + 1$

By setting the size of the vector $\boldsymbol{Z_{j,t}}$ to T_{max}, which is a fixed constant, we impose that the lasting of the feedback can be at most T_{max} steps. According to the scenario we are dealing with, T_{max} will be known or not known by the

agent. We do not have any preliminary knowledge regarding the distributions of $R_{j,t}$ and $\boldsymbol{Z_{j,t}}$, we only know that $R_{j,t}$ has support in $[R_{min}, R_{max}]$ and $Z_{j,t,m}$ has support in $[0, 1]$ where $1 \leq j \leq K$, $1 \leq t \leq N$, $1 \leq m \leq T_{max}$. Each realization of $\boldsymbol{Z_{j,t}}$ is characterized by the number of steps that we have to wait before having a positive component $Z_{j,t,m}$. We call this quantity *delay* and we formalize it in the following way:

Definition 2 (Delay)**.** *We define the delay of a realization* $\boldsymbol{Z_{j,t}}$ *as:*

$$d_{j,t} = \sum_{m=0}^{T_{max}} \mathbb{1}_{\{Z_{j,t,m}=0 \ \wedge \ \forall k<m \ Z_{j,t,k}=0\}}$$

We are also interested in the position of the last positive component of a realization $\boldsymbol{Z_{j,t}}$. This quantity represents the true number of steps that we need to wait in order to collect all the instant rewards achievable after the pull of an arm. For this reason, we call it *true length* and we define it as follows:

Definition 3 (True Length)**.** *We define the true length of a realization* $\boldsymbol{Z_{j,t}}$ *as:*

$$l_{j,t} = \sum_{m=0}^{T_{max}} \mathbb{1}_{\{\exists k \ k \geq m \ | \ Z_{j,t,k}>0\}}$$

To better suit a variety of scenarios that require a persistence reward framework, we devise two distinct configurations:

- **General Persistency** We do not assume anything regarding $\boldsymbol{Z_{j,t}}$. This configuration turns to be suitable for scenarios in which the istant reward could be missing at a certain point and then reappear at a later time.
- **Tight Persistency** We impose that, giving a realization of a persistency vector, every non-zero component must be adjacent. More formally, we say that we are in *Tight Persistency* configuration if for each realization $\boldsymbol{Z_{j,t}} = (Z_{j,t,1}, \ldots, Z_{j,t,Tmax})$ the following condition holds:

$$\nexists i, m, k \quad i < m < k \quad \text{s.t.} \quad Z_{j,t,i} > 0 \wedge Z_{j,t,m} = 0 \wedge \ Z_{j,t,k} > 0 \quad \forall i, m, k \in (1, \ldots, T_{max})$$

We now present a couple of examples derived from practical cases with the aim to highlight the needs that motivate the two configurations mentioned above.

Example 1 (Pricing of a magazine subscription)**.** *We are the seller of an online magazine that works via subscription. In order to have access to our service, a new user can stipulate a contract with fixed duration and monthly fees. We let the possibility to suspend and restart the service in every moment during the contract, simply stopping or making the monthly payments. Intuitively, we think that high prices discourage a continuous usage of the service while low prices could lead to stable subscriptions but with the risk of generating unsatisfactory profit. We are facing the problem of finding the best monthly price to assign at the service in order to maximize the revenue. This scenario can be directly modeled as a MAB persistent problem in* ***General Persistency****. Each arm can be assigned to a specific fee designed as a valid option. When the agent pulls an arm the extracted feedback* $R_{j,t}$ *will be the price related to that arm. The persistency vector* $\boldsymbol{Z_{j,t}}$*, on the other side, will capture the adherence of the user to the service and will have a size of Tmax equal to the number of months of the contract. Every component of the vector will be a Bernoulli variable that takes the value 1 if the user has made the payment for a certain month or the value 0 in the opposite case.*

Example 2 (Medical Trial)**.** *We want to conduct an ethical clinical trial in order to define which is the best medical treatment for a specific chronic illness. Let's say that we have two options available, a red pill and a blue one, hence we model them as two arms. Every day the agent must choose which one of the two therapies administer to a new patient on the basis of previous observations. Different from prior MAB application for this task, here we want to consider also the life quality of a patience in addition to his lifespan. For this reason after the treatment administration, a patient is tested every day and an index of his health status is computed. We can assume that this index is ranging from 0 to 1, where 1 represents a perfect state of health and 0 means that the patient is dead. This scenario could be easily addressed as a MAB persistent problem in* ***Tight Persistency*** *configuration with delay steps equal to zero for each realization of the persistency vector. As a matter of fact, we can set* T_{max} *at the maximum lifespan possible after the diagnosis of the considered ilness, and we can model every component of the vector* $\boldsymbol{Z_{j,t}}$ *as the health status index. For this scenario,* $R_{j,t}$ *could be fixed to a constant equal for each arm, letting the role of capturing the reward only to the persistency vector* $\boldsymbol{Z_{j,t}}$*. The Tight Peristency condition holds, as a matter of fact, it does not make sense to have a positive index health status after a death.*

The nature of the presented problem leads us to introduce two definitions of reward achievable pulling an arm. In a straightforward manner, we call

Pull Reward the the sum of the instant rewards gained thanks to the pull. In both Example 1 and Example 2, the goal of the learning agent was to find the arm able to maximize this quantity. However, in some scenarios could be reasonable to take in consideration also the time needed to collect all the instant rewards of a pull. In particular, we call *Normalized Pull Reward* the sum of instant rewards divided by the true length of the persistency vector. This measure is particularly relevant when we consider case studies in which we want to allocate resources and we must take into consideration possible vacant periods, as outlined in Example 3. Formal definitions of rewards are provided below.

Definition 4 (Pull Reward)**.** *We define the pull reward achived pulling the arm j at time t as:*

$$X_{j,t} = \sum_{s=t}^{t+T_{max}-1} r_{j,t,s}$$

Definition 5 (Normalized Pull Reward)**.** *We define the normalized pull reward achived pulling the arm j at time t as:*

$$Y_{j,t} = \frac{\sum_{s=t}^{t+T_{max}-1} r_{j,t,s}}{l_{j,t}}$$

Example 3 (Pricing of a Cloud Computing Service)**.** *A cloud computing company has a new set of servers at its disposal and is facing the problem of deciding the daily price to rent a server. Once a specific price has been chosen, the company will disclose its offer online and later will enter into a contract of a fixed duration with the purchaser. Each day of the contract, the user will pay a fixed cost concering the rent, in addition to a variable cost related to the resources usage. The company assumes that by publishing an offer with an high price it will take a long time to find a buyer, on the contrary, with a very low price it will immediately be able to rent it but with little profit. In this scenario we see how the unused server time affects the income, therefore, not only the accumulated reward must be taken into account but also the time necessary to find a buyer. The problem is well modeled in **Tight persistency**. Each arm a_j is associated to a deterministic daily price $R_{j,t}$ and the delay $d_{j,t}$ of each peristency vector $\boldsymbol{Z_{j,t}}$ will represent the days between the publication of the offer and the stipulation of the contract. Hence, $R_{j,t}$ can be seen as the price of one day of full use of the service, and finally each positive component $Z_{j,t,m}$ will indicate the fraction of $R_{j,t}$ to be daily paid by the user. We are interested in finding the arm that maximizes the **Normalized Pull Reward**, taking into account also the penalty imposed by the vacant periods.*

Table 2.1: Configurations-Rewards scenarios.
The combination General-Persistency/Normalized Pull Reward leads to impractical cases that are not of interest to us.

	Pull Reward	Normalized Pull Reward
General Persistency	example 1 Spotify Scenario	
Tight Persistency	example 2	example 3 Rent Scenario

Successive plays of arm a_j yield pull rewards $X_{j,t_1}, X_{j,t_2}, \dots$ which are random variables independent and identically distributed according to an unknown distribution with unknown expectation μ_j. In the same way, we can consider the normalized pull rewards, $Y_{j,t_1}, Y_{j,t_2}, \dots$ which are random variables i.i.d. with unknown expectation η_j. For the sake of simplicity, we will refer to a generic reward of arm a_j at time t as $X_{j,t}$, in order to adopt the same notation of the standard MAB literature. However, the definitions stated below will apply evenly to the *Pull reward* and the *Normalized pull Reward*, unless otherwise specified.

2.1.1 Performance measures

The goal of a learning agent is to maximize its cumulated reward, the pulling strategy adopted to accomplish this task is referred as *policy*. In order to measure the performance of a policy, we compare its beahviour with the one of a fictitious algorithm, called *Oracle*, which for any horizon of n time steps constantly plays the optimal arm. For this purpose, we introduce the concept of *Regret*.

Definition 6 (Regret)**.** *The Regret of a policy cumulated after n plays is definied as:*

$$R_n = \max_{j=\{1,\dots,k\}} \sum_{t=1}^{n} X_{j,t} - \sum_{t=1}^{n} X_{a_t,t}$$

Where a_t is the arm played by the learner at time t and the first term $\max_{j=\{1,\dots,k\}} \sum_{t=1}^{n} X_{j,t}$ represents the reward cumulated by the Oracle up to time n.

Since both the rewards and the player's actions are stochastic, we introduce a form of average regret called *pseudo-regret*.

Definition 7 (Pseudo-Regret)**.** *The Pseudo-Regret of a policy cumulated after n plays is definied as:*

$$\bar{R}_n = n\mu^* - \sum_{t=1}^{n} \mu_{a_t}$$

Where $\mu^* = \max_{j=\{1,\dots,k\}} \mu_j$ *is the expected reward of the optimal arm and* μ_{a_t} *is the expected reward of the arm played at time t.*

For clearity, we give the definition of *Normalized-Pseudo Regret*

This Pseudo-Regret form is more suitable for the purpose of our analysis, therefore in the rest of the thesis we will evaluate the algorithms in terms of pseudo-regret and, in order to simplify, we will refer to it as regret.

2.2 Modeling of real-world scenarios

2.2.1 The Spotify Playlist problem

- motivation: cold users, stato attuale -> possibile miglioramento
- fixed R constant - Z ascolti
- General Persistency - Pull Reward

2.2.2 The Rental Pricing problem

- soluzione ad hoc
- delay = sfitto
- Thigh Persistency - Normalized Pull Reward

Chapter 3

Novel Algorithms

3.1 Grounding Concepts

Multi-armed Bandit problems pertain to Online Learning, which means that in order to evaluate algorithms we need to design a simulation environment where we can mimic real-scenarios dynamics. Informally, we will call *Bucket* a realization $\boldsymbol{Z_{j,t}} = (Z_{j,t,1}, \ldots, Z_{j,t,Tmax})$ of the persistency vector. During the simulation, each bucket will be parsed according to the experiment time t, meaning that the learner can only visit the bucket-cells containing the information of the past. As stated in chapter 2, when at time t an arm a_j is played, the environment generates a bucket $\boldsymbol{Z_{j,t}}$ and a feedback $R_{j,t}$ that is collected by the learner. Changing the point of view, we can say that, during the experiment, each arm a_j collects a sequence of pulls, characterized by the extracted feedback-bucket pairs $(R_{j,t}, \boldsymbol{Z_{j,t}})$. We define as P_j the set of pulls of arm a_j, hence, when an arm is pulled a pair $(R, \boldsymbol{Z})$ id added to the his set. In order to simplify the notation we will omit the arm and time indices where not needed.
The algorithms described below have a structural difference from the standard ones designed for traditional Multi-armed Bandit, infact in the presented framework more than one arms can have a set of active buckets (not totally parsed yet) simultaneously. This parallelism implies that at each time instant we need to update the terms of every arms and not only of the last one played. In order to facilitate the understanding of the code, the described algorithms will be decoupled into two modules, the *Policy* and the *Update* function. The policy module will describe a generic learner's interaction with the environment and will require an update function passed as an argument. The update function will be responsible to update the data and compute the indices needed by the policy to take decisions, concretizing

the overall strategy.
BOZZA:

1. Baseline UCB
2. Baseline TS
3. Bound 1
4. TS persistent
5. TS persistent forced exploartion
6. Bayes UCB Persistent

3.2 Frequentist Algorithms

The pseudo-code of the Frequentist Policy is depicted in Algorithm 1. The first k istants form the initialization phase, during which each arm is chosen once. After the k-th time istant the loop phase begins. Here the agent plays the arm having the largest index $u_j(t)$, the upper confidence bound of the arm a_j at time t. After the play of an arm, the *Update Procedure* occurs.

Algorithm 1 `Frequentist Policy`

Require: arm set $A = \{a_1, a_2, \ldots, a_k\}$, time horizon N, update function U

```
1: function POLICY(A,N,U)
2:     for t ∈ {1,...,k} do                          ▷ init phase
3:         play arm a_t
4:         call U
5:     end for
6:     for t ∈ {k+1,...,N} do                        ▷ loop phase
7:         play arm a_i such that i = arg max_j u_j
8:         call U
9:     end for
10: end function
```

3.2.1 Baseline UCB

This algorithm 2 extends the idea of the well known UCB algorithm in the case of persistent rewards. The reward is considered as a unique variable available after the termination of the bucket.

Algorithm 2 `Baseline UCB`

Require: bucket size T_{max}, maximum feedback R_{max}
function UPDATE(T_{max},R_{max})
  for each arm $a_j \in A$ **do**
    for each pair $(R, \boldsymbol{Z}) \in P_j$ **do**
      if $\boldsymbol{Z}$ is ended **then**
        compute reward x
        compute average reward $\hat{\mu}_j$
        $P_j \leftarrow P_j \setminus \{(R, \boldsymbol{Z})\}$
      end if
    end for
    $c_j \leftarrow R_{max} T_{max} \sqrt{\frac{2log(t)}{T_j(t)}}$
    $u_j \leftarrow \hat{\mu}_j + c_j$
  end for
end function

Algorithm 3 `Bound1`

Require: bucket size T_{max}
function UPDATE(T_{max})
  for each arm $a_j \in A$ **do**
    for $m \in \{1, \ldots, T_{max}\}$ **do**
      compute average persistency $\overline{Z_{j,m}}$ on the available buckets
      $c_{j,m} \leftarrow \sqrt{2log(tT_{max}^{\frac{1}{4}})/v_{j,m}(t)}$
    end for
    $u_j \leftarrow R_j \sum_{m=1}^{T_{max}} min(1, \overline{Z_{j,m}} + c_{j,m})$
  end for
end function

3.2.2 Bound 1

3.3 Bayesian Algorithms

3.4 Farsighted versions

3.5 Adaptive versions

Chapter 4

Experimental Analysis

Appendix: growing tmax + farsighted vs myopic + framework codice esperimenti

4.1 s1

	Persistency		**Tmax**		**Reward**	
Experiment name	*General*	*Thight*	*Known*	*Unknown*	*P.R.*	*N.P.R.*
synthetic A		x	x		x	
synthetic B		x	x		x	
synthetic C		x		x	x	
Spotify Scenario	x		x		x	
Rent Scenario		x	x			x

Table 4.1: Experimental Analysis Summary

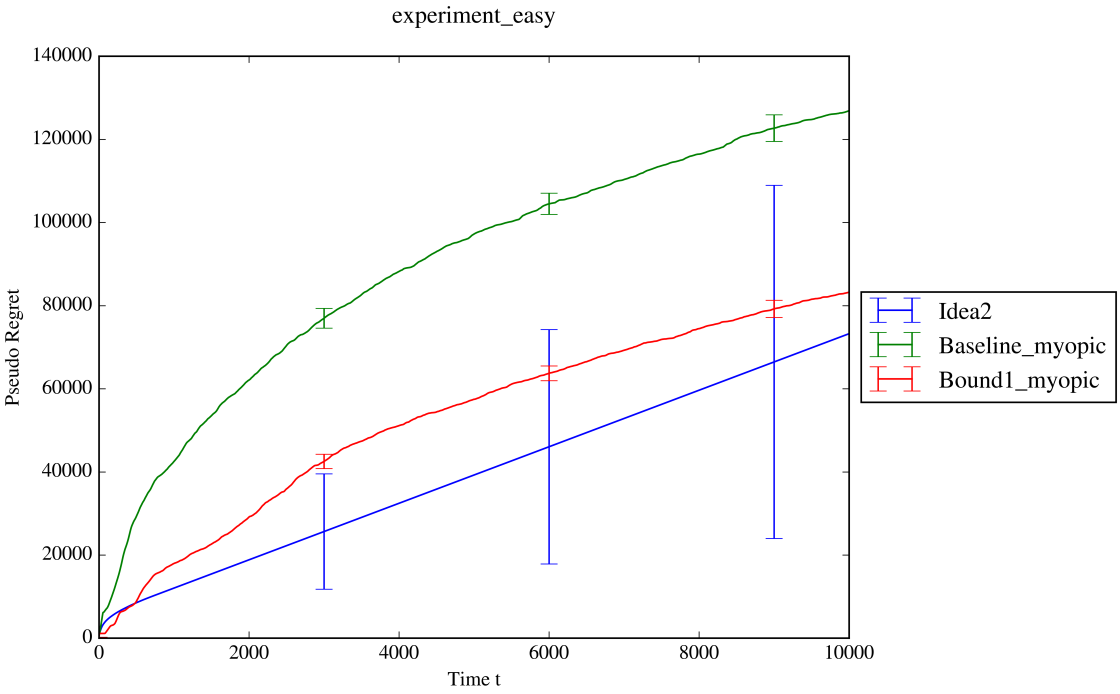

Figure 4.1: pseudo regret uniformi (R: 1...10) 30 runs Tmax=50

Chapter 5

Conclusions

In this work, we studied the relationship between games with imperfect recall and team games. First, we described how to apply the inflation operation in practice. Computing a completely inflated tree requires only polynomial time, but a completely inflated tree may still have imperfect recall. Then, we defined personalities as a way to decompose a player with imperfect recall in a set of team members with perfect recall. Specifically, we defined an auxiliary team game

Bibliography

A. Celli and N. Gatti. Computational results for extensive-form adversarial team games. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, 2018.