Esame Intelligenza Artificiale

Analisi del paper
Batched Multi-armed Bandits Problem

di Zijun Gao, Yanjun Han, Zhimei Ren, Zhengqing Zhou

Studente Gianluca Viganò
Matricola 1046188
CdL Ingegneria Informatica Magistrale

Anno Accademico 2019-2020
Sessione Estiva, 24 Luglio 2020

Indice

1 Contesto

Prima di presentare il lavoro del paper, è opportuno introdurre gli aspetti principali e ciò che è realmente necessario per comprendere appieno il problema che ci si propone di risolvere.

1.1 Batch learning e online learning

Oltre alla prima grande suddivisione che è possibile fare nel classificare gli algoritmi di machine learning (*supervised*, *unsupervised* e *reinforcement learning*), è possibile separare le modalità di apprendimento in due grandi categorie: *batch* e *online* learning. Queste si focalizzano sulla capacità degli agenti di imparare dal flusso di dati che arriva loro. Quando si parla di batch learning, il sistema che si sta considerando apprende solo dopo aver processato un numero precisato di dati (batch vuol dire appunto *lotto* in inglese, indicando una suddivisione in gruppi dei dati). Prima che il modello cambi i propri parametri per predire il risultato o per trovare la giusta azione da compiere è dunque necessario aver considerato diverse informazioni. Questo può comportare una complessità computazionale e un tempo di training non trascurabili. Un esempio in ambito supervised learning potrebbe essere l'algoritmo *Least Squares*, che ha bisogno della matrice di tutte le feature, moltiplicare quest'ultima per la sua trasposta e calcolarne l'inversa. Spesso quindi si decide di diminuire notevolmente la grandezza dei batch, degli insiemi di dati, e si parla in questo caso di *mini-batch learning*. Come caso limite si potrebbero dunque usare batch di dimensione unitaria, ed è in questo caso che si parla di online learning. In questo scenario il sistema è in grado di apprendere in modo incrementale (al contrario del precedente che viene detto anche *offline learning* per questo motivo), cambiando quindi i propri parametri ogni volta che arriva un nuovo dato. É chiaro quindi come le risorse computazionali richieste siano inferiori rispetto a prima e il tempo di training è ridotto. Per fare un esempio si può prendere in considerazione la tecnica del *Stochastic Gradient Descent*, dove il gradiente si approssima ad ogni iterazione con quello calcolato su un singolo addendo della funzione costo (che corrisponde ad un elemento del dataset) e i parametri vengono quindi aggiornati dopo ogni valutazione. Per gli algoritmi di online learning è problematico il caso in cui avvenga una continua aggiunta di dati in grande contrasto con quelli già appresi (per esempio dovuti a un sensore guasto) che portano quindi il modello ad apprendere in modo errato, pregiudicando quanto di buono imparato da una serie di dati in precedenza corretti. Inoltre esistono alcuni scenari in cui non è possibile ottenere dati uno dopo l'altro, ma questi arrivano già di per sè suddivisi in gruppi. É il caso dei test clinici, dove diversi campioni giungono in lotti.

1.2 Descrizione del problema base

In teoria della probabilità, il *Multi-armed Bandits Problem* (o anche *K-armed bandits problem*) è un problema nel quale un numero finito e fissato di risorse deve essere impiegato per decidere in un insieme di scelte concorrenti e alternative quale possa rendere massimo il guadagno atteso. La probabilità di generare una ricompensa è all'inizio ignota, e può essere parzialmente appresa solo col passare del tempo, "provando" tali scelte. Richiesta fondamentale è che le diverse possibili alternative devono essere indipendenti tra loro, e che la possibilità di ricevere un reward non dipenda dall'esito della decisione presa nel turno precedente. Questo dunque è un classico problema di reinforcement learning, dove l'obiettivo è massimizzare il reward medio ottenibile. Il nome deriva dalle slot machine, che negli Stati Uniti d'America vengono a volte ironicamente chiamate *one armed bandit*. Si immagina quindi uno scenario in cui esista una slot machine con più leve (*multi-armed*) che possono essere tirate, ognuna con una propria probabilità di vincere denaro. Il compito del giocatore, che all'inizio non conosce nessuna caratteristica delle sue possibili scelte, è trovare la leva che lo possa portare a guadagnare maggiormente.

Exploration vs. exploitation

Aspetto fondamentale nella strategia del giocatore è capire quando conviene provare nuove scelte e quando invece continuare a scegliere la leva migliore in base a quanto è riuscito ad apprendere fino al turno attuale, fidandosi della conoscenza acquisita. Assume quindi una notevole importanza il tradeoff fra *exploration* ed *exploitation*. Decidere di non esplorare mai nuove possibilità potrebbe portare il sistema a ottenere spesso una ricompensa piccola, mentre scegliere di sfruttare poco ciò che si è imparato continuando a provare nuove alternative potrebbe mantenere molto bassa la conoscenza del sistema in quanto per ogni braccio il numero di tentativi non sarebbe significativo, provocando quindi possibili scelte sbagliate.

Regret

Un importante parametro per capire quanto la strategia del giocatore sia buona è il *regret*, traducibile come *rammarico*. Ogni qualvolta che si decide di abbassare una leva con una bassa probabilità di ricevere la ricompensa, o comunque non si tira la leva ottima, cioè quella con probabilità maggiore di ottenere il reward, il giocatore a posteriori (avendo ottenuto tutte le informazioni necessarie per scegliere il braccio migliore) prova del rammarico, cioè è dispiaciuto perchè ha compiuto una scelta che di per sè non era ottima, e tanto più è lontano dal massimo profitto ottenibile, tanto più questo parametro è elevato. Indicando quindi con K l'insieme delle leve, con $t \in T$ lo specifico istante in cui si effettua una scelta, con $\mu^{(i)}$ la ricompensa media ottenibile dall' i-esimo braccio, si pone $\mu^* = \max_{i \in K} \mu^{(i)}$ il reward atteso del braccio ottimo. Una politica di campionamento

$\pi = (\pi_t)_{t=1}^{T}$ è definita come una sequenza di variabili casuali $\pi_t \in [K]$ che indica per ogni istante $t \in [T]$ quale leva debba essere tirata, e in ogni istante una determinata politica porta il sistema ad ottenere una ricompensa media $\mu^{(\pi_t)}$. É quindi possibile definire il regret $R_T(\pi)$ relativo ad una specifica π come

$$R_T(\pi) = \sum_{t=1}^{T}(\mu^* - \mu^{(\pi_t)}) = T\mu^* - \sum_{t=1}^{T}\mu^{(\pi_t)} \tag{1.1}$$

calcolando in ogni istante la differenza fra la migliore scelta e la scelta effettuata realmente. Poichè la ricompensa attesa del braccio migliore può essere considerata costante perchè non dipende da t, può essere portata fuori dalla sommatoria e moltiplicata per il numero di iterazioni. L'obiettivo del sistema può quindi essere visto, in modo duale alla massimizzazione del reward medio, come la minimizzazione del regret atteso.

1.3 Possibili strategie risolutive del problema base

Si prendono ora in considerazione tre differenti strategie per affrontare il problema dei banditi multi-braccia nel caso più semplice: l'agente casuale, l'agente epsilon-greedy e quello di Thompson. Quanto segue è stato scritto dopo aver consultato [Pat17]. Per facilitarne la comprensione, dopo ogni descrizione è presente un breve esempio con codice sviluppato in Python, disponibile al link `https://github.com/VigHub/Exam_IA`.

Si è innanzitutto progettata la classe `MultiArmedBandit`, che ha il compito di descrivere il comportamento del sistema. Essa presenta un unico campo, `reward_prob_list`, che non è altro che una lista contenente variabili di tipo `float`. Ognuna rappresenta la probabilità effettiva di ogni singolo braccio appartenente al bandit, dove la probabilità di ricevere un reward è modellata con una variabile casuale di Bernoulli. La ricompensa assume quindi il valore 1 se ottenuta, e 0 altrimenti. Il numero K di possibili scelte è quindi equivalente alla lunghezza della lista inizializzata nel costruttore. La classe possiede un singolo metodo, denominato `step` che accetta come parametro un intero, indicante quale leva si è deciso di tirare. Dopo aver controllato che la scelta sia possibile (rimanendo all'interno della lunghezza della lista), si simula tramite il metodo `random.choice` del package numpy una *prova di Bernoulli* con probabilità p relativa al particolare braccio scelto di ricevere una ricompensa.

```
1 import numpy as np
2
3
4 class MultiArmedBandit:
5     """Framework multi armed bandit """
6     def __init__(self, reward_prob_list):
7         """ Crea un nuovo sistema, con i parametri indicati
8
```

```
9         :param reward_prob_list: probabilita effettiva di ottenere la
  ricompensa da un determinato braccio/leva
10        """
11        self.reward_prob_list = reward_prob_list
12
13    def step(self, choice: int):
14        """ | Abbassa il braccio selezionato
15        | (reward e' 1 o 0) => Bernoulli
16
17        :param choice: il braccio da abbassare
18        :return: risultato dovuto all'aver abbassato quella leva
19        """
20        if choice > len(self.reward_prob_list) or choice < 0:
21            raise Exception(f"Errore, la scelta {choice} non e'
  possibile. Possibili scelte: {self.reward_prob_list}")
22        success = self.reward_prob_list[choice]
23        failure = 1.0 - success
24        return np.random.choice(2, p=[failure, success])
```

Codice 1.1: classe MultiArmedBandit.

Per ogni agente si è scritto un piccolo script nel quale viene testata la propria politica in un numero di round ben definito, ognuno con un certo numero di turni. Si è calcolato poi per ognuno il reward medio ottenuto.
Per questo esempio si è scelto di simulare 100 round da 1000 passi ciascuno, usando un multi-armed bandit con 6 possibili scelte con le seguenti probabilità di ottenere una ricompensa: [0.2, 0.6, 0.7, 0.4, 0.6, 0.1]. Ne segue quindi che un giocatore onniscente, che conosce quindi a priori le probabilità di ogni leva, tirerebbe sempre la terza poichè è quella che ha la possibilità più alta di portare al reward. Dopo mille passi, ci si aspetta quindi che tale giocatore abbia guadagnato una ricompensa media di 700 ($1000 \cdot 0.7 = 700$). Questo è il massimo reward medio ottenibile teoricamente e quindi quanto più una strategia si avvicina a questo valore, quanto più essa sarà valutata positivamente.

1.3.1 Agente casuale

Il primo agente considerato è anche il più semplice da implementare. In questa strategia, ad ogni turno il giocatore sceglie in modo casuale quale leva abbassare, in modo indipendente ad ogni iterazione. In questo scenario quindi non avviene mai una fase di exploitation, non si sfrutta mai la conoscenza acquisita, anche perchè le decisioni non sono dovute a specifici esiti di turni precedenti. Nel codice d'esempio l'azione compiuta è codificata nel metodo `get_random_choice` che necessita in input del numero di braccia K. La funzione non fa altro che ritornare un intero casuale compreso fra 0 e K-1 inclusi, a rappresentare la leva da tirare.

```python
from agents_simple_case.multi_armed_bandit import MultiArmedBandit
import settings
import numpy as np


def get_random_choice(possible_choices: int):
    """Scelta dell'agente casuale.

    :param possible_choices: insieme di leve da cui effettuare la scelta
    :return: leva da abbassare
    """
    return np.random.randint(0, possible_choices)


def play(reward_prob_list: list, rounds: int, steps: int):
    """Aziona l'agente casuale

    :param reward_prob_list: insieme di braccia con probabilita per crare MultiArmedBandit
    :param rounds: numero di volte in cui viene ripetuto l'esperimento
    :param steps: numero di turni (scelte) da fare in ogni round
    :return: reward medio ottenuto per round
    """
    bandit = MultiArmedBandit(reward_prob_list)
    arms = len(bandit.reward_prob_list)
    reward_list = []
    reward_counter_arm = np.zeros(arms)
    counter_arm = np.zeros(arms)
    print("Inizio agente casuale")
    for i in settings.progress_bar(range(0, rounds)):
        tot_reward = 0
        for step in range(steps):
            choice = get_random_choice(arms)
            reward = bandit.step(choice)
            reward_counter_arm[choice] += reward
            counter_arm[choice] += 1
            tot_reward += reward
        reward_list.append(tot_reward)
    mean = np.mean(reward_list)
    print(f"Ricompensa totale media: {mean}")
    print(f"probabilita ottenuta per braccio: {np.true_divide(reward_counter_arm, counter_arm)}")
    print(f"Probabilita \"reale\" per braccio {bandit.reward_prob_list}")
    return mean
```

Codice 1.2: Implementazione dell'agente casuale.

Il valore del reward medio ottenuto è di circa 434.15 e le probabilità che l'agente ha ottenuto per ogni braccio sono [0.203 0.592 0.700 0.401 0.608 0.101], che rappresentano con buona precisione tutti i valori delle leve del multi-armed bandit. Questo accade perchè nessun braccio viene privilegiato maggiormente e vengono tutti tirati mediamente lo stesso numero di volte all'interno dei 100 round da 1000 turni.

1.3.2 Agente epsilon-greedy

Il secondo agente preso in considerazione è quello che attua la tecnica *epsilon-greedy*. Ad ogni passo, con una probabilità $p = 1 - \epsilon$ viene selezionato il braccio che *con la conoscenza attuale* presenta la probabilità maggiore di ottenere un reward. Con probabilità ϵ l'agente sceglie casualmente una leva. All'aumentare di ϵ cresce quindi l'importanza data all'esplorazione rispetto allo sfruttamento della conoscenza acquisita. Un valore tipico di ϵ è 0.1. Nel codice d'esempio l'azione dell'agente è implementata nel metodo `get_epsilon_greedy_choice`, che prende in input il valore della costante ϵ e il numero di leve che possono essere tirate. In particolare, grazie al metodo `uniform` del modulo `random` si simula un'estrazione casuale da una variabile uniforme e se tale estrazione è di valore inferiore o uguale al valore di ϵ si entrerà nella prima condizione e si sceglierà casualmente dall'insieme di leve. Se invece il valore è maggiore di ϵ si entra nel ramo else dove si seleziona la leva con il valore più alto. Si noti come sia necessario aggiungere l'istruzione `choice = np.random.choice(indices)` nel caso in cui siano presenti più leve con la stessa probabilità attuale di ottenere il reward. Nel metodo `play` si simula tutta la durata dell'esperimento, e si noti come nel ciclo for più interno si memorizzi quale scelta viene fatta ad ogni turno e si faccia poi un tiro di leva con il codice `bandit.step(choice)`.

```
from agents_simple_case.multi_armed_bandit import MultiArmedBandit
import settings
import numpy as np
import random as rnd


def get_epsilon_greedy_choice(epsilon, reward_counter_array):
    """Scelta dell'agente epsilon-greedy

    :param epsilon: probabilita con cui l'agente effettua exploration anziche exploitation
    :param reward_counter_array: insieme di leve da cui effettuare la scelta
    :return: leva da abbassare
    """
    possible_choice = len(reward_counter_array)
    if rnd.uniform(0, 1) <= epsilon:  # exploration
        choice = np.random.randint(0, possible_choice)
    else:  # exploitation
```

```
        max_arm = np.amax(reward_counter_array)
        indices = np.where(reward_counter_array == max_arm)[0]
        choice = np.random.choice(indices)
    return choice


def play(reward_prob_list: list, rounds: int, steps: int):
    """Aziona l'agente epsilon-greedy

    :param reward_prob_list: insieme di braccia con probabilita per crare MultiArmedBandit
    :param rounds: numero di volte in cui viene ripetuto l'esperimento
    :param steps: numero di turni (scelte) da fare in ogni round
    :return: reward medio ottenuto per round
    """
    bandit = MultiArmedBandit(reward_prob_list)
    espilon = 0.1
    arms = len(bandit.reward_prob_list)

    tot_reward_list = []
    mean_prob_arm_list = np.zeros(arms)
    print("Inizio agente epsilon-greedy")
    for i in settings.progress_bar(range(0, rounds)):
        tot_reward = 0
        reward_counter_arm = np.zeros(arms)
        counter_arm = np.full(arms, 1.0e-6)
        for step in range(0, steps):
            choice = get_epsilon_greedy_choice(espilon, reward_counter_arm)
            reward = bandit.step(choice)
            reward_counter_arm[choice] += reward
            counter_arm[choice] += 1
            tot_reward += reward
        tot_reward_list.append(tot_reward)
        prob_arm_list = np.true_divide(reward_counter_arm, counter_arm)
        mean_prob_arm_list += prob_arm_list
    mean = np.mean(tot_reward_list)
    print(f"Ricompensa totale media: {mean}")
    print(f"Probabilita ottenuta per braccio: {mean_prob_arm_list / rounds}")
    print(f"Probabilita \"reale\" per braccio {bandit.reward_prob_list}")
    return mean
```

Codice 1.3: Implementazione dell'agente epsilon-greedy.

Il reward medio ottenuto è di 542.84, ben maggiore dell'agente casuale. Questo accade perchè questa strategia non effettua solo esplorazione e con probabilità $1 - \epsilon$ sfrutta la

conoscenza acquisita nel corso dell'esperimento. La stima di questo agente della probabilità di ottenere il reward per ogni braccio è [0.193 0.583 0.707 0.400 0.588 0.089], ancora abbastanza vicino ai valori reali, anche se la differenza è leggermente maggiore in questo caso, soprattutto per il braccio con minore probabilità, dato che è stato scelto un piccolo numero di volte.

1.3.3 Agente di Thompson

L'ultimo agente considerato per affrontare il problema base fa uso della strategia chiamata *Thompson Sampling* [Tho08]. Questa tecnica nasce da uno studio approfondito del *Teorema di Bayes* e sullo studio della *massima verosimiglianza*. In particolare, quando si vuole stimare una distribuzione di Bernoulli (che supponiamo essere la distribuzione dei nostri reward) tramite lo stimatore a Massima Verosimiglianza si può dire che

$$P(q) = \frac{s}{s+f}$$

dove $P(q)$ è la probabilità che un evento q si verichi, s il numero di successi e f il numero di fallimenti (failures). Questa sembra una formula del tutto ragionevole, ma può presentare problemi quando il numero totale di tentativi è basso. Per esempio, nel caso in cui la probabilità effettiva dell'evento con distribuzione di Bernoulli sia 30%, può accadere con una probabilità non così bassa (9%) che due tentativi su due siano entrambi successi. Questo porta il nostro stimatore a dire che la probabilità dell'evento è del 100%, sbagliando decisamente la previsione. Per ottenere buone previsioni anche con pochi dati bisogna far uso del teorema di Bayes, il cui enunciato è che la probabilità condizionata di un evento rispetto ad un altro è data dal prodotto della condizionata fra il secondo e il primo evento e della probabilità del primo, fratto la probabilità del secondo, cioè

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}.$$

In particolare bisogna trovare la distribuzione *a posteriori* $P(q|s,f)$, perchè conoscendola il giocatore può agire come un agente onniscente sapendo quale braccio porta a un reward medio più alto. Questa ditribuzione non è però di Bernoulli, perchè necessita di sapere anche il numero di successi e fallimenti attuali. Ecco che allora si applica il teorema di Bayes, trovando che

$$P(q|s,f) = \frac{P(s,f|q) \cdot P(q)}{P(s,f)} = \frac{P(s,f|q) \cdot P(q)}{\int P(s,f|q)P(q)dx}$$

dove la distribuzione $P(s,f|q)$ è la verosimiglianza e $P(q)$ è la distribuzione *a priori*. La verosimiglianza, poichè rappresenta il risultato di molti esperimenti indipendenti, può

essere calcolata tramite la distribuzione binomiale. Quindi:

$$P(s, f|q) = \binom{s+f}{s} q^s (1-q)^f.$$

Per calcolare la distribuzione a priori si può notare che la distribuzione a priori coniugata della Bernoulli è la *distribuzione Beta*, quindi

$$P(q) = \frac{q^{\alpha-1}(1-q)^{\beta-1}}{B(\alpha, \beta)},$$

dove α e β sono parametri maggiori di zero che rappresentano il tasso di successo e fallimento e $B(\alpha, \beta)$ è una costante di normalizzazione chiamata funzione Beta, che assicura che la somma di tutte le probabilità dia 1. Sostituendo tutti i termini e operando qualche semplificazione si ottiene

$$P(q|s, f) = \frac{q^{s+\alpha-1}(1-q)^{f+\beta-1}}{B(s+\alpha, f+\beta)}.$$

Guardando quanto ottenuto si nota che il risultato è ancora una distribuzione Beta, quindi per ottenere stime migliori non bisogna fare altro che inserire i dati ottenuti all'interno di una funzione Beta. La stima è un processo che avviene attraverso vari step, e bisogna inizializzare i parametri α e β a 1 perchè al tempo zero l'ipotesi migliore che si può fare è che la distribuzione a priori sia uniforme.

```
1  from agents_simple_case.multi_armed_bandit import MultiArmedBandit
2  import settings
3  import numpy as np
4
5
6  def get_thompson_choice(success_count_arm: np.ndarray, failure_count_arm
       : np.ndarray):
7      """Scelta dell'agente di Thompson
8
9      :param success_count_arm: numero di reward ottenuti per ogni braccio
10     :param failure_count_arm: numero di volte in cui ogni braccio non ha
       dato nulla
11     :return: leva da abbassare
12     """
13     beta_arr = np.random.beta(success_count_arm, failure_count_arm)
14     return np.argmax(beta_arr)
15
16
17 def play(reward_prob_list: list, rounds: int, steps: int):
18     """Aziona l'agente di Thompson
19
```

```
    :param reward_prob_list: insieme di braccia con probabilita per crare MultiArmedBandit
    :param rounds: numero di volte in cui viene ripetuto l'esperimento
    :param steps: numero di turni (scelte) da fare in ogni round
    :return: reward medio ottenuto per round
    """
    bandit = MultiArmedBandit(reward_prob_list)
    arms = len(bandit.reward_prob_list)
    tot_reward_list = []
    mean_prob_arm_list = np.zeros(arms)
    print("Inizio agente di Thompson")
    for i in settings.progress_bar(range(0, rounds)):
        tot_reward = 0
        success_count_arm = np.ones(arms)
        failure_count_arm = np.ones(arms)
        counter_arm = np.full(arms, 1.0e-6)
        for step in range(0, steps):
            choice = get_thompson_choice(success_count_arm, failure_count_arm)
            reward = bandit.step(choice)
            if reward == 1:
                success_count_arm[choice] += 1
            else:  # reward == 0
                failure_count_arm[choice] += 1
            counter_arm[choice] += 1
            tot_reward += reward
        tot_reward_list.append(tot_reward)
        prob_arm_list = np.true_divide(success_count_arm, counter_arm)
        mean_prob_arm_list += prob_arm_list
    mean = np.mean(tot_reward_list)
    print(f"Ricompensa totale media: {mean}")
    print(f"Probabilita ottenuta per braccio: {mean_prob_arm_list / rounds}")
    print(f"Probabilita \"reale\" per braccio {bandit.reward_prob_list}")
    return mean
```

Codice 1.4: Implementazione dell'agente di Thompson.

L'implementazione di questo agente è molto semplice, in quanto bisogna solo utilizzare l'istruzione `np.random.beta(success_count_arm, failure_count_arm)` ottenendo dato il numero di successi e di fallimenti un'estrazione da una distribuzione Beta per ogni braccio. Si sceglie poi la leva con il valore massimo e si aggiornano il numero di successi e di fallimenti in base all'esito dello step del bandit.

Il reward medio ottenuto è di 666.04, molto vicino al massimo valore teorico del giocatore onniscente (700). Questo dimostra la forza e la bontà di questa strategia, anche

se bisogna osservare che le stime per le altre leve sono le più diverse dalle probabilità effettive tra gli agenti considerati. Infatti i valori trovati sono [0.295 0.570 0.699 0.414 0.576 0.238] e alcune previsioni sono decisamente errate, come l'ultima dove il valore reale è 0.1 e il valore previsto 0.238. Però questo non deve soprendere: questo agente già dopo pochi passi riesce a individuare tramite la tecnica spiegata precedentemente quale è il braccio che verosimilmente porterà a un reward maggiore, scegliendo quasi mai altre leve. Questa tecnica sembra essere la strategia che meglio si adatta a risolvere il problema dei banditi, in realtà presenta buone prestazioni solo nei casi in cui si riesca a calcolare la distribuzioni a priori. Non sempre infatti il giocatore conosce la distribuzione dei reward, e di conseguenza non è sempre possibile trovare o addirittura calcolare la distribuzione coniugata a priori. In questi casi può essere utile impiegare la tecnica epsilon-greedy vista in precedenza o altre ancora.

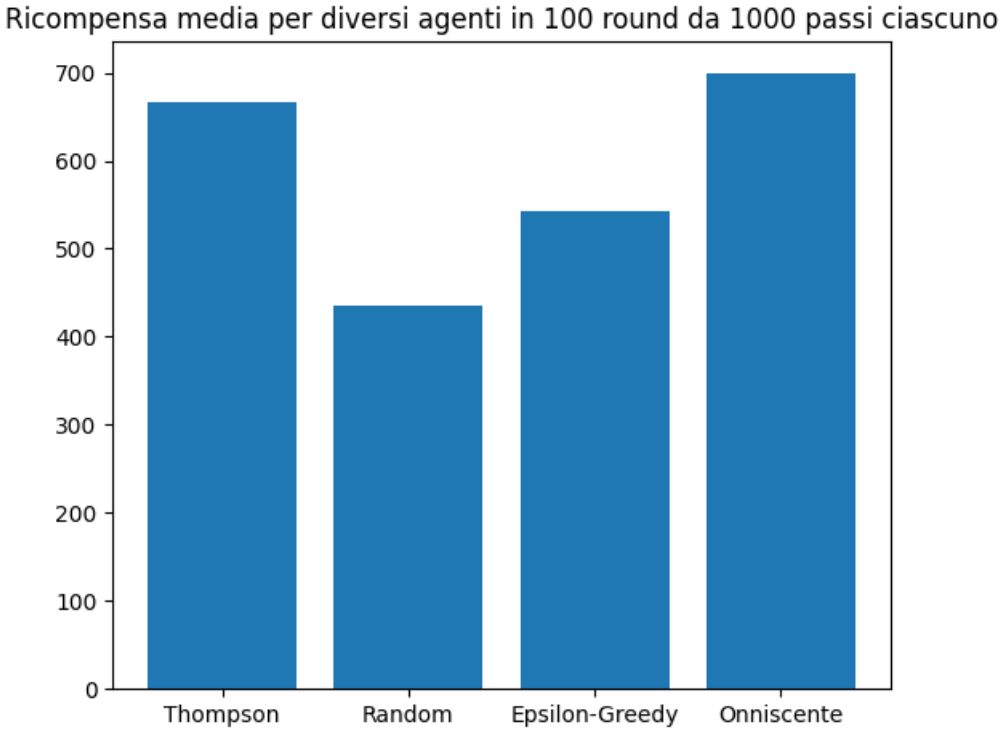


Figura 1.1: Grafico che mostra tutte e tre le strategie spiegate. Si noti l'ottima prestazione dell'agente di Thompson, che è di poco inferiore alla ricompensa media del giocatore onniscente.

1.4 Scenari applicativi

Sono numerosi gli scenari applicativi nei quali si applicano strategie risolutive a problemi reali modellati come il problema dei banditi. Di seguito ne vengono descritti un paio.

Netflix e le immagini di copertina

Netflix è una della società di distribuzione film, serie TV e altri contenuti più famose al mondo, con un enorme bacino d'utenza. Uno dei suoi obiettivi principali è quello di *personalizzare* al massimo l'esperienza dell'utente, consigliandogli contenuti che con molta probabilità apprezzerà, in base a quanto già visto in precedenza. La personalizzazione può essere però ancora maggiore: non solo il sistema vuole personalizzare i contenuti consigliati, ma addirittura all'interno degli stessi contenuti vuole catturare l'utente mostrandogli l'immagine di copertina (chiamata *thumbnail*) che più provoca interesse nell'utente. Per esempio, se il sistema prevede che è opportuno conisigliare una determinata commedia romantica, per l'utente maggiormente attratto dal lato comico si mostrerà la raccomandazione con un attore che ride, mentre se si preferisce il lato romantico verrà mostrata la thumbnail con l'immagine di un bacio. Si noti che entrambe le figure sono dello stesso film, e non è possibile effetturare *A/B testing* in uno scenario dove ci sono forti limiti di tempo (non si ha una settimana o più per capire quale sia la copertina migliore per ogni utente). Ecco che allora entra in gioco il *Contextual Multi-Armed Bandit* [Gar20], dove la parola contextual sta a indicare che la copertina può cambiare per ogni singola persona. Si cerca quindi di massimizzare il reward medio, simboleggiato dal voto che ogni utente indica per ogni contenuto.

Figura 1.2: Esempio di processo di decisione tramite Contextual Multi-Armed Bandit: se per due differenti utenti il sistema decide che deve consigliare il film *Genio Ribelle*, mostrerà all'utente che ha visto più film romantici una thumbnail con un bacio, per l'utente che ha visto più commedie verrà mostrata l'immagine con un sorriso.

Trial clinici

Con il termine *Trial Clinico* si intende uno studio medico farmacologico, biomedico o correlato alla salute sull'uomo. Questa tipologia di studio presenta dei protocolli ben definiti. L'obiettivo è quello di verificare che una nuova terapia o cura sia più efficace, migliore e soprattutto sicura di quella normalmente impiegata. Assume notevole importanza l'insieme di campioni sui quali poter testare nuove cure. Questi spesso vengono forniti in lotti,

in gruppi di numero preciso che devono essere etichettati e confezionati terapeuticamente. Si capisce dunque che ci sono dinamiche diverse rispetto allo scenario in cui per esempio si ha un sensore e periodicamente con tempo stabilito si ricevono nuovi dati, e ottenerli presenta un costo esiguo. Bisogna quindi cercare di ottimizzare la dimensione dei lotti e operare strategie che portano a piccoli livelli di incertezza anche con un numero piccolo di lotti. Ecco che allora nei trial clinici si considera il *Batched Multi-Armed Bandit*, dove la parola batched sta a indicare i gruppi di campioni. In particolare questo aspetto è l'oggetto principale di studio del paper in esame.

2 Studi correlati

In questo capitolo vengono elencati in ordine alfabetico di autore tutti i lavori citati dal paper. Per ognuno, si è letto l'abstract e laddove fossero necessari ulteriori chiarimenti per capire l'oggetto di studio si è letto l'intero lavoro. Nei casi in cui i lavori avessero più di trenta o quaranta anni e quindi senza abstract, si è letta l'introduzione e alcuni passi principali degli studi.

[AA88] Noga Alon and Yossi Azar.
"Sorting, approximate sorting, and searching in rounds".
***SIAM Journal on Discrete Mathematics*, 1(3):269–280, 1988**

Considera nel caso peggiore il numero di confronti necessario per ordinare o selezionare n oggetti in k turni, differenziando algoritmi deterministici e randomizzati. Migliora i bound precedentemente noti per l'ordinamento e per la ricerca della mediana, separando quest'ultimo dal problema di ricerca del minimo.

[AAAK17] Arpit Agarwal, Shivani Agarwal, Sepehr Assadi, and Sanjeev Khanna. "Learning with limited rounds of adaptivity: Coin tossing, multi-armed bandits, and ranking from pairwise comparisons".
***Conference on Learning Theory*, pages 39–75, 2017**

Studia la relazione fra la complessità e l'adattività nel richiedere attivamente nuovi dati per identificare le k monete che hanno una probabilità maggiore di ottenere il risultato desiderato (testa), in un insieme di n monete. Da questo primo aspetto passa poi a considerare il problema delle k miglior braccia nel Multi-Armed Bandit Problem e al problema dell'ordinamento con confronti a coppie dei primi k oggetti in un insieme finito.

[AB09] Jean-Yves Audibert and Sébastien Bubeck.
"Minimax policies for adversarial and stochastic bandits".
***COLT*, pages 217–226, 2009**

Studia a fondo la politica minimax nel problema dei banditi con k braccia nel caso stocastico (le ricompense sono indipendenti e identicamente distribuite) e in quello in cui i reward sono fissati prima dell'inizio dell'esperimento, ma non sono necessariamente tutti iid (adversarial case). Rimuove un fattore logaritmico nel limite superiore e propone una nuova famiglia di algoritmi che fanno uso di tecniche di randomizzazione e normalizzazione.

[AB10] Jean-Yves Audibert and Sébastien Bubeck.
"Regret bounds and minimax policies under partial monitoring".
***Journal of Machine Learning Research*, 11(Oct):2785–2836, 2010**

Il paper affronta diversi problemi di previsione e 4 tipologie di regret: pseudo-regret, regret atteso, regret di alta probabilità e tracciamento del miglior regret atteso. Introduce un nuovo stimatore chiamato INF basato su normalizzazione implicita e abbassa alcuni upper bound sul regret del Multi-Armed Bandit Problem.

[ACF02] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer.
"Finite-time analysis of the multi-armed bandit problem".
***Machine learning*, 47(2-3):235–256, 2002**

Studia il problema dei banditi focalizzandosi sulla variabile tempo, sul numero di turni. Prova quindi che il regret ottimo cresce almeno logaritmicamente non solo in relazione al numero di turni ma anche uniformemente in base al tempo con semplici ed efficienti politiche e per tutte le distribuzioni del reward con supporto limitato (cioè l'insieme dei punti in cui la distribuzione non è una funzione liscia, che vuol dire derivabile infinite volte in quel punto).

[AMS09] Jean-Yves Audibert, Rémi Munos, and Csaba Szepesvári.
"Exploration-exploitation tradeoff using variance estimates in multi-armed bandits".
***Theoretical Computer Science*, 410(19):1876–1902, 2009**

Considera una variante dell'algoritmo base per il multi-armed bandit problem usando la varianza empirica delle braccia. Si sofferma quindi sul bilanciamento fra esplorazione e sfruttamento. Discute infine di come l'upper bound per il regret sia logaritmico e che possa quindi non essere adatto a problemi reali con decisori avversi al rischio.

[AO10] Peter Auer and Ronald Ortner.
"UCB revisited: Improved regret bounds for the stochastic multi-armed bandit problem".
***Periodica Mathematica Hungarica*, 61(1-2):55–65, 2010**

Modifica l'algoritmo UCB di [ACF02] e migliora l'upper bound per il regret.

[AWBR09] Alekh Agarwal, Martin J. Wainwright, Peter L. Bartlett, and Pradeep K. Ravikumar.
"Information-theoretic lower bounds on the oracle complexity of convex optimization".
***Advances in Neural Information Processing Systems*, pages 1–9, 2009**

Si sofferma maggiormente sull'uso dell'ottimizzazione convessa, studiando la complessità di quest'ultima in un modello di computazione a oracolo. Ottiene infine bound più stringenti per l'approccio minimax per varie classi di funzioni.

[BC12] Sébastien Bubeck and Nicolò Cesa-Bianchi.
"Regret analysis of stochastic and non- stochastic multi-armed bandit problems".
***Foundations and Trends® in Machine Learning*, 5(1):1–122, 2012**

Esegue un'analisi approfondita del regret nei problemi multi-armed bandit evidenziando le differenze fra esplorazione e sfuttamento. In particolare, confronta due casi estremi: ricompense indipendenti e identicamente distribuite e il caso in cui non siano indipendenti. Analizza infine alcune varianti ed estensioni, come il contextual bandit model, dove ad ogni turno il giocatore può scegliere solo un sottoinsieme delle possibili scelte.

[BK97] Apostolos N. Burnetas and Michael N. Katehakis.
"Optimal adaptive policies for markov decision processes".
***Mathematics of Operations Research*, 22(1):222–255, 1997**

Considera il problema del controllo adattativo per processi decisionali di Markov (MDP), cioè identificare quale sia la migliore azione da eseguire in un dato stato, in modo da ottenere il massimo valore possibile di una funzione cumulativa della ricompensa. Trova la forma esplicita per una classe di politiche adattative che fanno aumentare in modo ottimale la ricompensa totale attesa in un orizzonte finito, sotto opportune ipotesi di stati e azioni finite e di irriducibilità nelle leggi di transizioni fra stati.

[BM07] Dimitris Bertsimas and Adam J. Mersereau.
"A learning approach for interactive marketing to a customer segment".
***Operations Research*, 55(6):1120–1135, 2007**

Analizzando il problema del marketing interattivo, dove un consulente deve comunicare ai propri clienti se investire in determinate azioni (sfruttamento) o se pagare per ottenere nuovi informazioni su nuove azioni (esplorazione), mostra un approccio di apprendimento di programmazione dinamica approssimativa basato sulla decomposizione lagrangiana e un'euristica basata su approssimazione asintotiche per la soluzione al multi-armed bandit problem.

[BMW16] Mark Braverman, Jieming Mao, and S. Matthew Weinberg.
"Parallel algorithms for select and partition with noisy comparisons".
***Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 851–862. ACM, 2016**

Considera il problema di trovare il k-esimo più alto elemento in un insieme ordinato di n elementi, partizionandolo in due sottoinsiemi formati dai primi k e dai restanti $n-k$ elementi, usando confronti a coppie. Confronta le prestazioni degli algoritmi proposti in tre scenari: senza rumore, con cancellazione (dove il risultato di un confronto può essere senza risultato), e con rumore (dove il risultato del confronto può essere errato con una certa probabilità).

[BPR13] Sébastien Bubeck, Vianney Perchet, and Philippe Rigollet.
"Bounded regret in stochastic multi-armed bandits".
***Proceedings of the 26th Annual Conference on Learning Theory*,**
pages 122–134, 2013

Studia il multi-armed bandit problem nel caso in cui si conosca quale sia il valore della scelta ottima e il lower bound sulla più piccola differenza fra valore di una scelta e scelta ottima. Propone quindi una politica di randomizzazione che porta a un regret uniformemente limitato nel tempo, e mostra diversi lower bound, dimostrando che conoscere solo una delle due ipotesi sopra dichiarate renda impossibile ottenere bound più bassi.

[BT83] Béla Bollobás and Andrew Thomason.
"Parallel sorting".
***Discrete Applied Mathematics*, 6(1):1–11, 1983**

Mostra che se il numero di elementi n è sufficientemente grande, esiste un grafo G con particolari proprietà relative alla chiusura transitiva di ogni orientazione aciclica, cioè l'assegnamento di una direzione od orientamento per ogni arco che non forma un ciclo diretto (portando quindi il grafo a essere privo di cicli diretti). La chiusura transitiva di un grafo è un grafo con gli stessi nodi che presenta un arco da un nodo ad un altro solo se è presente un cammino tra quei due nodi nel grafo di partenza. Partendo dalla considerazione di prima viene mostrato che un grande numero ($n^{32} \log n$) di processori in parallelo rende possibile ordinare in due intervalli di tempo n oggetti.

[CDS13] Nicolò Cesa-Bianchi, Ofer Dekel, and Ohad Shamir.
"Online learning with switching costs and other adaptive adversaries".
***Advances in Neural Information Processing Systems*, pages 1160–1168, 2013**

Studia differenti tipi di avversari adattativi in predizione. Misura le performance del giocatore introducendo un nuovo tipo di regret: il policy regret, che sembra spiegare

maggiormente l'adattatività dell'avversario in funzione del comportamento del giocatore. In particolare mostra dei bound nel caso in cui cambiare scelta (scegliere un altro braccio) presenti un costo (switching costs).

[CG09] Stephen E. Chick and Noah Gans.
"Economic analysis of simulation selection problems".
***Management Science*, 55(3):421–437, 2009**

Presenta un nuovo approccio per il problema di selezionare una simulazione per una progettazione di sistema. Nello specifico, mostra un framework che comunica quando ha senso spendere tempo e soldi per simulare un design di sistema per comprendere le sue performance e per quanto tempo si debba continuare la simulazione per approvare o rifiutare il design proposto.

[CT06] Thomas M. Cover and Joy A. Thomas.
"Elements of Information Theory". Wiley, New York, second edition, 2006

E' uno dei libri più autorevoli e maggiormente consultati riguardo alla teoria dell'informazione e della trasmissione.

[DKMR14] Susan Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy.
"Top-k and clustering with noisy comparisons".
***ACM Transactions on Database Systems (TODS)*, 39(4):35, 2014**

Studia i problemi di top k (trovare i k migliori oggetti relativi a una specifica query) e clustering (raggruppare elementi secondo specifiche proprietà) quando le operazioni di confronto possono essere svolte da oracoli le cui risposte possono però essere errate, proponendo alcuni efficienti algoritmi che garantiscono risultati corretti. Mostra infine come tale risultato possa essere usato nel generare oracoli tramite crowdsourcing.

[DRY18] John Duchi, Feng Ruan, and Chulhee Yun.
"Minimax bounds on stochastic batched convex optimization".
***Conference On Learning Theory*, pages 3065–3162, 2018**

Studia il multi-armed bandit problem nel caso batched proponendo una soluzione che fa uso di molte osservazioni parallele per ottimizzare una funzione convessa dato un numero limitato di turni del giocatore. In ogni round l'algoritmo può richiedere informazione su un numero limitato di punti e riceve una risposta alterata da una qualche funzione di rumore. Dopo una serie di turni l'algoritmo produce uno stimatore e presenta quindi i bound per l'approccio minimax.

[EMM06] Eyal Even-Dar, Shie Mannor, and Yishay Mansour.
"Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems". ***Journal of machine learning research*****, 7(Jun):1079–1105, 2006**

Mostra gli intervalli di confidenza nel problema dei banditi dimostrando quale sia il numero minimo di scelte per trovare il braccio ottimale con una probabilità definita. Propone un framework che cerca di eliminare le azioni che non sono ottime con alta probabilità. Infine mostra una variante basata su modello e una senza modello per il metodo di eliminazione, derivando anche le condizioni di stop che garantiscono che la politica imparata sia approssimativamente ottima con alta probabilità.

[EKMM19] Hossein Esfandiari, Amin Karbasi, Abbas Mehrabian, and Vahab Mirrokni. "Batched multi-armed bandits with optimal regret".
arXiv preprint arXiv:1910.04959**, 2019**

Propone tre algoritmi online per massimizzazione di funzioni submodulari, cioè che soddisfano l'equazione $f(x) + f(y) >= f(x \,\&\, y) + f(x \mid y) \;\; \forall x, y \in B^n$, che sono una classe importante di funzioni pseudo-booelane, dato che possono essere ottimizzate in tempo polinomiale. Mostra quindi per ogni algoritmo i propri bound.

[FRPU94] Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal.
"Computing with noisy information".
SIAM Journal on Computing**, 23(5):1001–1018, 1994**

Studia la profondità negli alberi decisionali con rumore, dove ogni nodo può dare una risposta sbagliata con una certa probabilità. Il numero di interrogazioni è correlato al calcolo delle funzioni soglia. Studia infine selezione e ordinamento parallelo con confronti che presentano rumore, mostrando i bound di diversi problemi.

[GC11] Aurélien Garivier and Olivier Cappé.
"The kl-ucb algorithm for bounded stochastic bandits and beyond".
Proceedings of the 24th annual conference on learning theory**, pages 359–376, 2011**

Presenta un'analisi dell'algoritmo KL-UCB (Kullback-Leibler Upper Confidence Bounds for optimal Sequential Allocation) con una politica di apprendimento online, senza orizzonte temporale per il problema dei banditi con scelte identicamente distribuite e indipendenti. Mostra quindi come tale algoritmo porti a lower bound ottimi, e lo confronta con altri algoritmi come UCB e UCB2, dimostrando che possiede bound migliori riguardo al regret.

[JJNZ16] Kwang-Sung Jun, Kevin G. Jamieson, Robert D. Nowak, and Xiaojin Zhu. "Top arm identification in multi-armed bandits with batch arm pulls". *AISTATS*, pages 139–148, 2016

Introduce una nuova tipologia di problema per il multi-armed bandit problem dove le braccia sono campionate in lotti (batch), anzichè una ogni istante di tempo. Ciò accade soprattutto in applicazioni di social media o in esperimenti biologici. Mostrano quindi i risultati teorici e alcune simulazioni per identificare geni importanti nella replicazione dei virus e per trovare gli utenti più attivi su Twitter rispetto a uno specifico argomento.

[KCS08] Aniket Kittur, Ed H. Chi, and Bongwon Suh. "Crowdsourcing user studies with mechanical turk". *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 453–456. ACM, 2008

Studia l'importanza dello sviluppo collettivo (crowdsourcing) in relazione a compiti di piccole dimensioni (micro-task), prendendo in considerazione la piattaforma Amazon Mechanical Turk. Mostra come sia di fondamentale importanza la giusta formulazione dei task per sfruttare appieno le capacità di questo apporoccio.

[LR85] Tze Leung Lai and Herbert Robbins. "Asymptotically efficient adaptive allocation rules". *Advances in applied mathematics*, 6(1):4–22, 1985

Studia catene di Markov dove ad ogni transizione di stato è associato un reward, e mostra una legge di controllo che massimizza la ricompensa media su un percorso infinito. L'idea base è quella di aggiungere aggiustamenti per migliorare l'incertezza grazie a test teorici.

[NY83] Arkadii Semenovich Nemirovsky and David Borisovich Yudin. "Problem complexity and method efficiency in optimization". Wiley, 1983

In questo libro gli autori studiano algoritmi per minimizzare una funzione convessa soggetta a vincoli convessi. La quantità di lavoro svolto dall'algoritmo viene identificata con il numero di punti in cui le funzioni devono essere valutate per garantire un determinato grado di precisione. L'analisi è del tipo worst-case. Vengono approfonditi l'algoritmo del metodo dei centri di gravità (MCG) e del metodo della discesa speculare (MMD). Vengono stabiliti risultati con limite inferiore per mostrare che il primo è subottimale per problemi su domini arbitrari, mentre il secondo è subottimale per problemi con molte variabili e un dominio uniforme.

[PR13] Vianney Perchet and Philippe Rigollet.
"The multi-armed bandit problem with covariates".
***The Annals of Statistics*, pages 693–721, 2013**

Considera il problema dei banditi dove ogni braccio presenta una ricompensa con rumore che dipende da variabile casuale. Al contrario del problema classico, questa variante permette cambiamenti dinamici dei reward che descrivono meglio scenari in cui l'informazione non è certa. Utilizza un modello non parametrico e introudce una politica chiamata ABSE (Eliminazione Successiva Adattativa e Categorizzata, Binned) che decompone adattativamente il problema in più piccoli problemi dei banditi di tipo statico.

[PRCS16] Vianney Perchet, Philippe Rigollet, Sylvain Chassang, and Erik Snowberg. "Batched bandit problems".
***The Annals of Statistics*, 44(2):660–681, 2016**

Studia il regret ottenibile nel problema dei banditi sotto l'ipotesi che la politica impiegata debba dividere i campioni in gruppi (batch). Questo è motivato da numerose applicazioni pratiche, come per esempio i test clinici. Mostra quindi come anche solo un piccolo numero di gruppi porti i limiti del regret a quelli ottimi dell'approccio minimax. Deriva infine una politica ottima per il caso in cui cambiare scelta non abbia costo nullo (switching cost).

[Rob52] Herbert Robbins.
"Some aspects of the sequential design of experiments".
***Bulletin of the American Mathematical Society*, 58(5):527–535, 1952**

E' un articolo che non ha ricevuto molte attenzioni al tempo della stesura ma è stato rivalutato negli ultimi anni e viene spesso citato. Studia la teoria dell'analisi sequenziale, soffermandosi anche sulla numerosità campionaria che deve essere raggiunta. Mostra quindi alcuni semplici problemi modellizzandoli con la tecnica del design sequenziale.

[Sha13] Ohad Shamir.
"On the complexity of bandit and derivative-free stochastic convex optimization". *Conference on Learning Theory*, pages 3–24, 2013

Approfondisce le questioni di ottimizzazione convessa nel problema dei banditi e senza conoscere i gradienti soffermandosi maggiormente sulla complessità e sui lower bound. Mette in relazione il regret con la dimensione d e il numero di interrogazioni T. Dimostra che il numero di interrogazioni scala almeno quadraticamente con la dimensione. Mostra infine che è possibile ottenere un error rate veloce in termini di T anche senza essere a conoscenza dei gradienti, contrariamente a quanto si pensava prima della stesura del paper.

[Tho08] William R. Thompson.
"On the likelihood that one unknown probability exceeds another in view of the evidence of two samples".
***Biometrika*, 25(3/4):285–294, 1933**

Articolo nel quale William R. Thompson introduce l'euristica denominata Thompson Sampling, di notevole importanza applicativa. Il procedimento è stato spiegato precedentemente nella sezione Contesto.

[Tsy08] A. Tsybakov.
"Introduction to Nonparametric Estimation"". Springer-Verlag, 2008

Libro che si propone di introdurre la teoria della stima non parametrica. Mostra inoltre i risultati di un approccio minimax per stimatori di densità e funzioni di regressione da osservazioni indipendenti.

[Val75] Leslie G Valiant.
"Parallelism in comparison problems".
***SIAM Journal on Computing*, 4(3):348–355, 1975**

Esamina la complessità temporale nel caso peggiore per calcolatori multiprocessore con confronti binari. Mostra la relazione fra il numero di elementi e il numero di processori per i problemi di ricerca del massimo, ordinamento e merging di liste ordinate. Spiega nel dettaglio l'algoritmo di ricerca del massimo dimostrando che è ottimo per tutti i valori di n, numero di elementi, e k, numero di processori.

[Vog60] Walter Vogel.
"A sequential design for the two armed bandit".
***The Annals of Mathematical Statistics*, 31(2):430–443, 1960**

Introduce il problema dei banditi con sole due braccia, mostrando un design sequenziale, dove l'obiettivo è capire quale fra due esperimenti con diverse probabilità generi un risultato migliore. Lo sperimentatore (cioè il giocatore) ha un numero stabilito di step nei quali di volta in volta può scegliere quale sperimento provare. Descrive quindi alcune semplici strategie provandone la non ottimalità teorica, mostrando però che in scenari reali queste possono essere più economiche di strategie più complesse.

3 Descrizione del paper

Si presenta ora il lavoro del paper studiato.

3.1 Griglie temporali

Dopo aver introdotto il concetto di *regret* (1.1) e dopo aver supposto che le ricompense per ogni braccio si comportino come delle variabili casuali gaussiane con media μ e varianza unitaria (quindi $\nu^{(i)} = \mathcal{N}(\mu^{(i)}, 1)$), viene spiegato come l'orizzonte temporale T venga suddiviso in M batch. L'insieme dei batch è rappresentato dall'insieme $\mathcal{T} = \{t_1, t_2, ..., t_M\}$ con $1 < t_1 < t_2 < ... < t_M = T$, chiamato griglia. Tale insieme può essere di due tipi: *statico* o *adattativo*. Nel primo caso la griglia è definita prima del tempo, prima di selezionare qualunque braccio. Nel secondo invece $\forall j \in [M]$, il valore di t_j viene calcolato dopo aver osservato i reward fino al tempo t_{j-1} e usando alcuni elementi di casualità. La griglia di tipo adattativo è più potente e anche pratica rispetto al caso statico, e si può fare batch learning settando $M = 1$ e online learning con $M = T$.

3.2 Risultati principali

Nel paper viene poi mostrato il concetto di politica (*policy*) π, già discussa nel capitolo 1, e viene poi indicato lo scopo principale del lavoro: caratterizzare il *minimax regret* e il *problem dependent regret* nel problema dei banditi con insieme delle politiche $\Pi_{M,T}$ con M batch, orizzonte temporale T e K braccia. In particolare si definisce:

$$R^*_{min-max}(K, M, T) \triangleq \inf_{\pi \in \Pi_{M,T}} \sup_{\{\mu^{(i)}\}_{i=1}^K : \Delta_i < \sqrt{K}} \mathbb{E}[R_T(\pi)], \tag{3.1}$$

$$R^*_{pro-dep}(K, M, T) \triangleq \inf_{\pi \in \Pi_{M,T}} \sup_{\Delta > 0} \Delta \cdot \sup_{\{\mu^{(i)}\}_{i=1}^K : \Delta_i \in \{0\} \cup [\Delta, \sqrt{K}]} \mathbb{E}[R_T(\pi)]. \tag{3.2}$$

Si noti come le differenze fra le possibili scelte possono essere arbitarie nella definizione minimax, mentre viene richiesto che il gap sia maggiore di 0 ($\Delta > 0$) nel caso di regret dipendente dal problema. Il limite $\sqrt{K}$ è relativo a una condizione tecnica, dovuta al fatto che per il caso $M = T$, gli upper bound del regret hanno la forma

$$\mathbb{E}[R_T(\pi^1)] \leq C\sqrt{KT},$$

$$\mathbb{E}[R_T(\pi^2)] \leq C \sum_{i \in [K] : \Delta_i > 0} \frac{max\{1, \log(T\Delta^2)\}}{\Delta_i},$$

con π^1 e π^2 politiche specifiche, e $C > 0$ costante assoluta.

Come risultato, nel caso adattativo limite dove $M = T$ si hanno i regret ottimi $R^*_{min-max}(K, M, T) = \Theta(\sqrt{KT})$ e $R^*_{pro-dep}(K, M, T) = \Theta(K \log T)$, il paper si pone come

obiettivo di trovare la dipendenza di queste quantità dal numero M di batch.

Sequono i principali teoremi dimostrati nel paper.

3.2.1 Teorema 1: Upper Bound

Per ogni $K > 2, T \geq 1, 1 \leq M \leq T$, esistono due politiche π^1 e π^2 con *griglia statica* tali che se $\max_{i \in [K]} \Delta_i \leq \sqrt{K}$ allora:

$$\begin{aligned} \mathbb{E}[R_T(\pi^1)] &\leq \texttt{polylog}(K,T) \cdot \sqrt{K} T^{\frac{1}{2-2^{1-M}}}, \\ \mathbb{E}[R_T(\pi^2)] &\leq \texttt{polylog}(K,T) \cdot \frac{K T^{\frac{1}{M}}}{min_{i \neq *} \Delta_i}, \end{aligned} \tag{3.3}$$

dove `polylog`(K,T) indica una funzione con fattori polinomiali e logaritmici in K e T. Si noti come per la seconda policy sia necessaria la condizione $i \neq *$ per il minimo, perchè altrimenti si sceglierebbe ovviamente il braccio migliore cioè con $\Delta = 0$ annullando il denominatore.

Per i limiti inferiori vengono trattati in modo diverso i casi in cui la griglia sia statica o adattativa.

3.2.2 Teorema 2: Lower Bound con griglia statica

Per il problema dei banditi con K braccia e M batched con orizzonte temporale T e griglia statica, i lower bound del regret minimax e problem-dependent sono:

$$\begin{aligned} R^*_{min-max}(K,M,T) &\geq c \cdot \sqrt{K} T^{\frac{1}{2-2^{1-M}}}, \\ R^*_{pro-dep}(K,M,T) &\geq c \cdot K T^{\frac{1}{M}}, \end{aligned} \tag{3.4}$$

con $c > 0$ costante numerica indipendente da K, M, T.
Si noti come i limiti siano molto simili al teorema precedente, con la differenza fra la costante c e i fattori poli-logaritmici.

3.2.3 Teorema 3: Lower Bound con griglia adattativa

Per il problema dei banditi con K braccia e M batched con orizzonte temporale T e griglia adattativa, i lower bound del regret minimax e problem-dependent sono:

$$\begin{aligned} R^*_{min-max}(K,M,T) &\geq c M^{-2} \cdot \sqrt{K} T^{\frac{1}{2-2^{1-M}}}, \\ R^*_{pro-dep}(K,M,T) &\geq c M^{-2} \cdot K T^{\frac{1}{M}}, \end{aligned} \tag{3.5}$$

con $c > 0$ costante numerica indipendente da K, M, T.
Confrontati col Teorema 2, questi limiti sono di valore minore e presentano come unica differenza il termine M^{-2}, ed è attualmente oggetto di studio se e come questo parametro possa essere rimosso.

3.3 Politica BaSE

Nello studio viene quindi presentata la stategia ideata dagli autori: la BaSE policy. Questa si basa principalmente sull'algoritmo SE (Successive Elimination) [PR13], che è stato dimostrato portare a bound ottimi nel caso $M = T$, e viene adattata nel caso in esame chiamandosi Batched Successive Elimination diventando quindi più generale con $M \leq T$. Data una griglia $\mathcal{T} = \{t_1, t_2, ..., t_M\}$, l'idea è quella di esplorare i primi $M-1$ gruppi batch per poi scegliere il braccio migliore nell'ultimo batch. Alla fine di ogni esplorazione in un lotto, si rimuovono le braccia che si è previsto avere una scarsa probabilità di ottenere una ricompensa basandosi sulle osservazioni passate.
Nello specifico si indica con $\mathcal{A} \subseteq \mathcal{I}$ l'insieme di braccia *attive* cioè che si possono ancora scegliere e non eliminate dall'insieme globale di tutte le braccia. Ovviamente si inizializza $\mathcal{A} = \mathcal{I}$, e ad ogni fine esplorazione si riduce sempre più l'insieme $\mathcal{A}$. Nell'ultimo batch si seleziona da $\mathcal{A}$ il braccio migliore.
Si indica con

$$\bar{Y}^i(t) = \frac{1}{|\{s \leq t : \text{braccio } i \text{ è tirato al tempo } s\}|} \sum_{s=1}^{t} Y_s \mathbb{1}\{\text{braccio } i \text{ è tirato al tempo } s\}$$

la ricompensa media del braccio i fino al tempo t, con $\mathbb{1}$ funzione indicatrice che quindi porta a considerare nella sommatoria i reward solo negli istanti temporali nei quali è stato tirato quel braccio specifico.
Risulta necessario specificare anche il tipo di griglia, in particolare si calcolano le due tipologie in questo modo:

$$u_1 = a, \quad u_m = a\sqrt{u_{m-1}}, \quad m = 2, ..., M \quad t_m = \lfloor u_m \rfloor, \quad m \in [M],$$

$$u'_1 = b, \quad u'_m = bu'_{m-1}, \quad m = 2, ..., M \quad t'_m = \lfloor u'_m \rfloor, \quad m \in [M],$$

dove a, b sono parametri scelti in modo che $t_M = t'_M = T$. Si dimostra quindi questa proprietà per entrambi i parametri. Nel primo caso:

$$u_1 = a, \quad u_2 = a\sqrt{u_1} = a^{1+\frac{1}{2}}, \quad u_3 = a\sqrt{u_2} = a^{1+\frac{1}{2}+\frac{1}{4}} \quad ...$$

quindi $u_m = a^{\sum_{i=0}^{M-1}(\frac{1}{2})^i}$, che significa $u_m = a^{\frac{1-2^{-M}}{1-\frac{1}{2}}} = a^{2-2^{-M+1}}$ e dato che $u_m = t_m = T$ risulta che $T = a^{2-2^{-M+1}}$ da cui

$$a = T^{\frac{1}{2-2^{1-M}}}.$$

Si può notare come l'esponente di T sia lo stesso che compare in (3.3) relativamente alla politica per regret minimax.
Per il secondo caso invece:

$$u'_1 = b, \quad u_2 = bu'_1 = b^2, \quad u'_3 = bu'_2 = b^3 \quad ...$$

quindi $u'_m = b^M$ e dato che come prima $u'_m = t'_m = T$ risulta che $T = b^M$ da cui

$$b = T^{\frac{1}{M}}.$$

Si noti come anche in questo caso l'esponente di T sia lo stesso visto in (3.3), questa volta relativamente al caso di regret problem-dependent.
Quindi per minimizzare il minimax regret si userà la griglia "minimax" definita da $\mathcal{T}_{minimax} = \{t_1, t_2, ..., t_M\}$, e per minimizzare il problem-dependent regret la griglia "geometrica" $\mathcal{T}_{geometric} = \{t'_1, t'_2, ..., t'_M\}$.
Si indicano con π^1_{BaSE} e π^2_{BaSE} le rispettive politiche per le griglie.
Ecco lo pseudo-codice dell'algoritmo:

Algorithm 1: Batched Successive Elimination (BaSE)

Input: Braccia $\mathcal{I} = [K]$; orizzonte temporale T; numero di batch M; griglia $\mathcal{T} = \{t_1, ..., t_M\}$; parametro di tuning γ.
Inizializzazione: $\mathcal{A} \leftarrow \mathcal{I}$
for $m \leftarrow 1, M-1$ **do**
  (a) Durante il periodo $[t_{m-1}+1, t_m]$, tira un braccio da $\mathcal{A}$ per uno stesso numero di volte.
  (b) Al tempo t_m :
  Sia $\bar{Y}^{max}(t_m) = \max_{j\in\mathcal{A}} \bar{Y}^j(t_m)$, e sia τ_m il numero totale di tiri per ogni braccio.
  for $i \in \mathcal{A}$ **do**
    if $\bar{Y}^{max}(t_m) - \bar{Y}^i(t_m) \geq \sqrt{\gamma \log(TK)/\tau_m}$ **then**
      $\mathcal{A} \leftarrow \mathcal{A} - \{i\}$
    end
  end
end
for $t \leftarrow t_{M-1}+1, T$ **do**
  tira il braccio i_0 tale che $i_0 \in \arg\max_{j\in\mathcal{A}} \bar{Y}^j(t_{M-1})$
  (in caso di parità scegli arbitrariamente).
end
Output: Politica risultante π.

3.4 Analisi Upper Bound

Le performance dell'algoritmo BaSE sono riassunte nel seguente Teorema.

3.4.1 Teorema 4: Precisazioni su Upper Bound

Si consideri un problema dei banditi a K braccia, con M batch dove l'orizzonte temporale è T. Sia π^1_{BaSE} la strategia BaSE per la griglia $\mathcal{T}_{minimax}$ e π^2_{BaSE} quella per la griglia $\mathcal{T}_{geometric}$. Per $\gamma \geq 12$ e $\max_{i\in[K]} \Delta_i = O(\sqrt{K})$ risulta che:

$$\begin{aligned} \mathbb{E}[R_T(\pi^1)] &\leq C \log K \sqrt{\log(KT)} \cdot \sqrt{K} T^{\frac{1}{2-2^{1-M}}}, \\ \mathbb{E}[R_T(\pi^2)] &\leq C \log K \log(KT) \cdot \frac{K T^{\frac{1}{M}}}{min_{i\neq *}\Delta_i}, \end{aligned} \tag{3.6}$$

con $C > 0$ costante indipendente da K, M e T.
Questo teorema è una formulazione più precisa del Teorema 1, perchè sono stati identificati chiaramente i termini delle funzioni poli-logaritmiche.

3.4.2 Dimostrazione Upper Bound

Per semplificare la notazione si assume che ci siano $K+1$ braccia, con il braccio 0 la miglior scelta possibile con il più alto reward atteso (indicato con $*$). Δ_i assume lo stesso significato visto in precedenza. Si definiscono poi tre eventi particolari.

1. Sia A_i l'evento in cui il braccio i viene eliminato prima dell'istante t_{m_i}, dove

$$m_i = \min \left\{ j \in [M] : i \text{ è tirato almeno } \tau_i^* \triangleq \frac{4\gamma \log(KT)}{\Delta_i^2} \text{ volte prima di } t_j \in \mathcal{T} \right\}$$

   e aggiungendo che se il minimo non esiste, si setta $m_i = M$ e accade l'evento A_i.

2. Sia B l'evento in cui il braccio $*$ non è eliminato in tutto l'orizzonte temporale T.

3. L'evento finale desiderato E è definito come $E = (\cap_{i=1}^K A_i) \cap B$. Questa definizione diventa chiara nel momento in cui ci si ricorda che l'obiettivo è eliminare tutte le braccia non ottime $(\cap_{i=1}^K A_i)$ e al tempo stesso scegliere il braccio migliore (evento B).

Nel paper viene poi dimostrato che se $\gamma \geq 12$ allora l'evento E accade con un'alta probabilità, nello specifico vale il seguente lemma:

Lemma 1 L'evento E accade con probabilità maggiore o uguale di $1 - \frac{2}{KT}$.

Da questo si può dedurre che al massimo il regret atteso per una qualunque delle due politiche quando non accade l'evento E è

$$\mathbb{E}[R_T(\pi)\mathbb{1}(E^c)] \geq T \max_{i\in[K]} \Delta_i \cdot \mathbb{P}(E^c) = O(1),$$

indicando con E^c l'evento complementare all'evento E e con $\mathbb{P}$ la probabilità che accada un certo evento.
Il paper si focalizza poi nel dimostrare l'upper bound solo nel caso minimax, dicendo che per il caso a griglia geometrica sono necessari gli stessi passaggi. Si introducono quindi ulteriori variabili, chiamando $\mathcal{I}_0 \subseteq \mathcal{I}$ l'insieme casuale di braccia che sono state eliminate alla fine dell'esplorazione del primo batch, $\mathcal{I}_1 \subseteq \mathcal{I}$ l'insieme casuale delle braccia rimanenti che vengono eliminate prima dell'ultimo batch, e $\mathcal{I}_2 = \mathcal{I} - \mathcal{I}_0 - \mathcal{I}_1$ l'insieme casuale di braccia che rimangono nell'ultimo batch. Il rammarico totale massimo che si può sperimentare in $\mathcal{I}_0$ è $t_1 \cdot \max_{i\in[K]} \Delta_i = O(\sqrt{K}a)$, che equivale al caso in cui in ogni istante del primo batch si sceglie sempre il braccio peggiore.
Rimane quindi solo da considerare il regret per gli insiemi $\mathcal{I}_1$ e $\mathcal{I}_2$.
Per quanto riguarda ogni braccio $i \in \mathcal{I}_1$, sia σ_i il numero casuale di braccia che sono state eliminate *prima* del braccio i. In relazione al numero totale di possibili scelte effettuate (numero di volte che si è abbassata una leva) il valor massimo per il braccio i del numero di volte che è stato scelto prima esso che sia stato eliminato è $\dfrac{1}{K-\sigma_i}$. Quindi, ricordando la definizione di t_{m_i} si può dire che

$$\tau_i^* > (\text{numero di scelte del braccio } i \text{ prima di } t_{m_i}) \geq \frac{t_{m_i-1}}{K} \to \Delta_i\sqrt{t_{m_i-1}} \leq \sqrt{4\gamma K \log(KT)},$$

perchè

$$\frac{4\gamma\log(KT)}{\Delta_i^2} \geq \frac{t_{m_i-1}}{K} \quad \to \quad K \cdot 4\gamma\log(KT) \geq \Delta_i^2 \cdot t_{m_i-1}.$$

Da questa considerazione si può dedurre che il regret massimo ottenibile in $\mathcal{I}_1$ è (notando che $t_j \leq 2a\sqrt{t_{j-1}}$ per $2 \leq j \leq M$ per come è stata costruita la griglia minimax):

$$\Delta_i \cdot \frac{t_{m_i}}{K-\sigma_i} \leq \Delta_i \cdot \frac{2a\sqrt{t_{m_i-1}}}{K-\sigma_i} \leq \frac{2a\sqrt{4\gamma K\log(KT)}}{K-\sigma_i}.$$

Il valore di σ_i può essere al massimo t e almeno $K-t$ per $t = 2, ..., K$. Segue quindi che il massimo regret sperimentabile in $\mathcal{I}_1$ è:

$$\sum_{i\in\mathcal{I}_1} \frac{2a\sqrt{4\gamma K\log(KT)}}{K-\sigma_i} \leq 2a\sqrt{4\gamma K\log(KT)} \cdot \sum_{t=2}^{K} \frac{1}{t} \leq 2a\log K\sqrt{4\gamma K\log(KT)}, \quad (3.7)$$

perchè $\sum_{t=2}^{K} \dfrac{1}{t} \leq \log K$.

Si considera quindi l'ultimo insieme rimanente, definendo T_i il numero di volte in cui si è tirato il braccio $i \in \mathcal{I}_2$ e con il massimo regret ottenibile per braccio che risulta essere

$$\Delta_i T_i \leq T_i \sqrt{\frac{4\gamma K \log(KT)}{t_{M-1}}} \leq \frac{T_i}{T} \cdot 2a\sqrt{4\gamma K \log(KT)},$$

dove si è usata la relazione $T = t_M$. Dato che $\sum_{i \in \mathcal{I}_2} T_i \leq T$, il regret totale ottenibile in $\mathcal{I}_2$ è:

$$\sum_{i \in \mathcal{I}_2} \frac{T_i}{T} \cdot 2a\sqrt{4\gamma K \log(KT)} \leq 2a\sqrt{4\gamma K \log(KT)}. \tag{3.8}$$

Considerando i bound di $\mathcal{I}_1$ (3.7) e $\mathcal{I}_2$ (3.8) e ragruppando i termini risulta che

$$R_T(\pi^1_{BaSE})\mathbb{1}(E) \leq 2a\sqrt{4\gamma K \log(KT)} \cdot (1 + \log K) + O(\sqrt{K}a),$$

e considerando quest'ultima disequazione con i bound per l'insieme $\mathcal{I}_0$, si ritrova l'upper bound indicato nel Teorema 4 (3.6).

3.5 Analisi Lower Bound

Ci si sofferma ora sui lower bound indicati precedentemente, differenziando lo scenario in cui si ha griglia statica da quello con griglia adattativa.

3.5.1 Dimostrazione Lower Bound con griglia statica

Come prima passo il paper propone il seguente lemma.

Lemma 2 Per una qualunque griglia statica $0 = t_o < t_1 < ... < t_M = T$ e la più piccola differenza $\Delta \in (0, \sqrt{K}]$, vale il seguente *minimax lower bound* per qualunque politica π:

$$\sup_{\{\mu^{(i)}\}_{i=1}^K : \Delta \in \{0\} \cup [\Delta, \sqrt{K}]} \mathbb{E}[R_T(\pi)] \geq \Delta \cdot \sum_{j=1}^{M} \frac{t_j - t_{j-1}}{4} \exp\left(-\frac{2t_{j-1}\Delta^2}{K-1}\right).$$

Viene poi mostrato come da questo lemma discenda poi il lower bound indicato nel Teorema 2 per un opportuno valore di Δ. Infatti, ricordandosi le definzioni del minimax regret $R^*_{min-max}$ (3.1) e del problem-dependent regret $R^*_{pro-dep}$ (3.2) e scegliendo

$\Delta = \Delta_j = \sqrt{\frac{K-1}{t_{j-1}+1}}$ con $\Delta_j \in [0, \sqrt{K}]$ risulta che:

$$R^*_{min-max}(K, M, T) \geq c_0\sqrt{K} \cdot \max_{j\in[M]} \frac{t_j}{\sqrt{t_{j-1}+1}},$$

$$R^*_{pro-dep}(K, M, T) \geq c_0 K \cdot \max_{j\in[M]} \frac{t_j}{t_{j-1}+1},$$

per una qualche costante numerica $c_0 > 0$. Si dimostra quanto detto sostituendo nella disequazione del lemma il valore di Δ indicato, così che l'esponente diventi $\frac{2t_{j-1}\cancel{(K-1)}}{(t_{j-1}+1)\cancel{(K-1)}} \approx 2$, cioè una costante numerica, e la sommatoria risulta essere $\sum_{j=1}^{M}(t_j - t_{j-1}) = t_M - t_0$. Dato che $t_M = T$ e $t_0 = 0$, e per come è stata costruita la griglia (cioè $t_j = a\sqrt{t_{j-1}}$), si può semplificare la frazione scrivendo $\frac{a\sqrt{t_{j-1}}}{\sqrt{t_{j-1}+1}} \approx a$. Ricordandosi che $a = \Theta(T^{\frac{1}{2-2^{1-M}}})$ si ritrova il limite del Teorema 2 (3.4). Possono essere fatti gli stessi passaggi per il regret problem dependent giungendo alle stesse conslusioni.

Il paper cerca poi di dimostrare la veridicità del lemma. Considera innanzitutto K diverse distribuzioni dei reward così definite:

$$\begin{aligned}
P_1 &= \mathcal{N}(\Delta, 1) \otimes \mathcal{N}(0, 1) \otimes \mathcal{N}(0, 1) \otimes ... \otimes \mathcal{N}(0, 1),\\
P_2 &= \mathcal{N}(\Delta, 1) \otimes \mathcal{N}(2\Delta, 1) \otimes \mathcal{N}(0, 1) \otimes ... \otimes \mathcal{N}(0, 1),\\
P_3 &= \mathcal{N}(\Delta, 1) \otimes \mathcal{N}(0, 1) \otimes \mathcal{N}(2\Delta, 1) \otimes ... \otimes \mathcal{N}(0, 1),\\
&\vdots\\
P_K &= \mathcal{N}(\Delta, 1) \otimes \mathcal{N}(0, 1) \otimes \mathcal{N}(0, 1) \otimes ... \otimes \mathcal{N}(2\Delta, 1).
\end{aligned}$$

Con $\otimes$ si intende una *diade*, tensore di secondo ordine che indica un prodotto tra vettori. Si noti come questa costruzione non sia simmetrica in quanto la distribuzione del primo braccio è sempre $\mathcal{N}(\Delta, 1)$. Da questa segue che:

1. Per ogni braccio $i \in [K]$, il braccio i è quello ottimo sotto la distribuzione P_i;
2. Per ogni braccio $i \in [K]$, tirare il braccio sbagliato porta a un regret di almeno Δ sotto la distribuzione P_i.

Di conseguenza, indicando con P_i^t la distribuzione delle osservazioni disponibili al tempo t in P_i, e con $R^T(\pi)$ il regret istantaneo provato dalla politica π_t al tempo t, risulta vero che l'estremo superiore del regret medio deve essere almeno maggiore della seguente quantità:

$$\sup_{\{\mu^{(i)}\}_{i=1}^{K}:\Delta\in\{0\}\cup[\Delta,\sqrt{K}]} \mathbb{E}[R_T(\pi)] \geq \frac{1}{K}\sum_{i=1}^{K}\sum_{t=1}^{T}\mathbb{E}_{P_i^t}[R^T(\pi)] \geq \Delta\sum_{t=1}^{T}\frac{1}{K}\sum_{i=1}^{K}P_i^t(\pi_t \neq i). \quad (3.9)$$

Rimane quindi solo da trovare il limite inferiore di $\frac{1}{K}\sum_{i=1}^{K}P_i^t(\pi_t \neq i)\ \forall t \in [T]$, argomento affrontato nel seguente lemma.

Lemma 3 Sia $Q_1, \cdots, Q_n$ la misura di probabilità di un qualche spazio delle probabilità $(\Omega, \mathcal{F})$ e sia $\Phi : \Omega \to [n]$ una sua qualche funzione di misura, cioè che assegna un numero reale ai sottoinsiemi di Ω per rendere quantitativa la nozione della sua estensione. Allora per ogni albero $T = ([n], E)$ con insieme dei vertici n e insieme degli archi E si ha:

$$\frac{1}{n}\sum_{i=1}^{n}Q_i(\Phi \neq i) \geq \sum_{(i,j)\in E}\frac{1}{2n}\exp(-D_{KL}(Q_i||Q_j)),$$

dove la divergenza di Kullback–Leibler (detta anche *entropia relativa*) è definita come $D_{KL}(P||Q) = \sum_i P(i)\log_2\left(\frac{P(i)}{Q(i)}\right)$, con P e Q distribuzioni discrete.

Si applica quindi questo lemma alla disequazione del lower bound prima trovata con l'albero $T = ([n], \{(1, i) : 2 \leq i \leq n\})$. Si definisce quindi $T_i(t)$ il numero di volte in cui si è tirato il braccio i prima del batch corrente t. Quindi $\sum_{i=1}^{K}T_i(t) = t_{j-1}$ se $t \in (t_{j-1}, t_j]$. Inoltre dato che $D_{KL}(P_1^t||P_i^t) = 2\Delta^2\mathbb{E}_{P_1^t}[T_i(t)]$ si può scrivere:

$$\begin{aligned}\frac{1}{K}\sum_{i=1}^{K}P_i^t(\pi_t \neq i) &\geq \frac{1}{2K}\sum_{i=2}^{K}\exp(-D_{KL}(P_1^t||P_i^t)) = \frac{1}{2K}\sum_{i=2}^{K}\exp(-2\Delta^2\mathbb{E}_{P_1^t}T_i(t)) \\ &\geq \frac{K-1}{2K}\exp\left(-\frac{2\Delta^2}{K-1}\mathbb{E}_{P_i^t}\sum_{i=2}^{K}T_i(t)\right) \geq \frac{1}{4}\exp\left(-\frac{2\Delta^2 t_{j-1}}{K-1}\right). \quad (3.10)\end{aligned}$$

Combinando 3.10 con il 3.9 si ottiene la prova del Lemma 2.

3.5.2 Dimostrazione Lower Bound con griglia adattativa

Infine il paper prende in considerazione il caso con griglia adattativa, cioè generata sequenzialmente in modo adattativo e con randomizzazioni. In modo analogo ai casi precedenti, si dimostra solo per il minimax regret dato che i passaggi per il problem-dependent regret sono analoghi.

Si consideri il tempo $T_1, \cdots, T_M \in [1, T]$ e le differenze $\Delta_i, \cdots, \Delta_M \in (0, \sqrt{K}]$ con

$$T_j = \lfloor T^{\frac{1-2^{-j}}{1-2^{-M}}} \rfloor, \qquad \Delta_j = \frac{\sqrt{K}}{36M} \cdot T^{\frac{1-2^{-j}}{2(1-2^{-M})}}, \qquad j \in [M].$$

Sia poi $\mathcal{T} = \{t_1 \cdots, t_M\}$ una qualunque griglia adattativa, e sia π una qualche politica sulla griglia $\mathcal{T}$. Per ogni $j \in [M]$ si definisce l'evento $A = \{t_{j-1} < T_{j-1}, t_j > T_j\}$ nella politica π con la convenzione che $t_0 = 0, t_M = T$. Si noti che gli eventi $A_1, \cdots, A_M$ formano una partizione dell'intero spazio delle probabilità. Si definisce poi la seguente famiglia di distribuzioni

$$P_{j,k} = \mathcal{N}(0,1) \otimes \cdots \otimes \mathcal{N}(0,1) \otimes \mathcal{N}(\Delta_j + \Delta_M, 1) \otimes \mathcal{N}(0,1) \otimes \cdots \otimes \mathcal{N}(0,1) \otimes \mathcal{N}(\Delta_M, 1),$$

per $j \in [M-1], k \in [K-1]$, e con la k-esima componente di $P_{j,k}$ che ha una media non nulla. Per $j = M$ si definisce

$$P_M = \mathcal{N}(0,1) \otimes \cdots \otimes \mathcal{N}(0,1) \otimes \mathcal{N}(\Delta_M, 1).$$

Aver costruito le distribuzioni in questo modo assicura che $P_{j,k}$ e P_M differiscano solo per la k-esima componente. La quantità p_j è definita come

$$p_j = \frac{1}{K-1} \sum_{k=1}^{K-1} P_{j,k}(A_j), \qquad j \in [M-1], \qquad p_M = P_M(A_M),$$

dove $P_{j,k(A)}$ indica la probabilità dell'evento A data la vera distribuzione $P_{j,k}$ e la politica π. Seguono poi due lemmi.

Lemma 4 Se $p_j \geq \frac{1}{2M}$ per qualche $j \in [M]$, allora

$$\sup_{\{\mu^{(i)}\}_{i=1}^K : \Delta \leq \sqrt{K}} \mathbb{E}[R_T(\pi)] \geq cM^{-2} \cdot \sqrt{K} T^{\frac{1}{2-2^{1-M}}}$$

dove $c > 0$ è una qualche costante indipendente da K,M,T e $\pi, \mathcal{T}$.

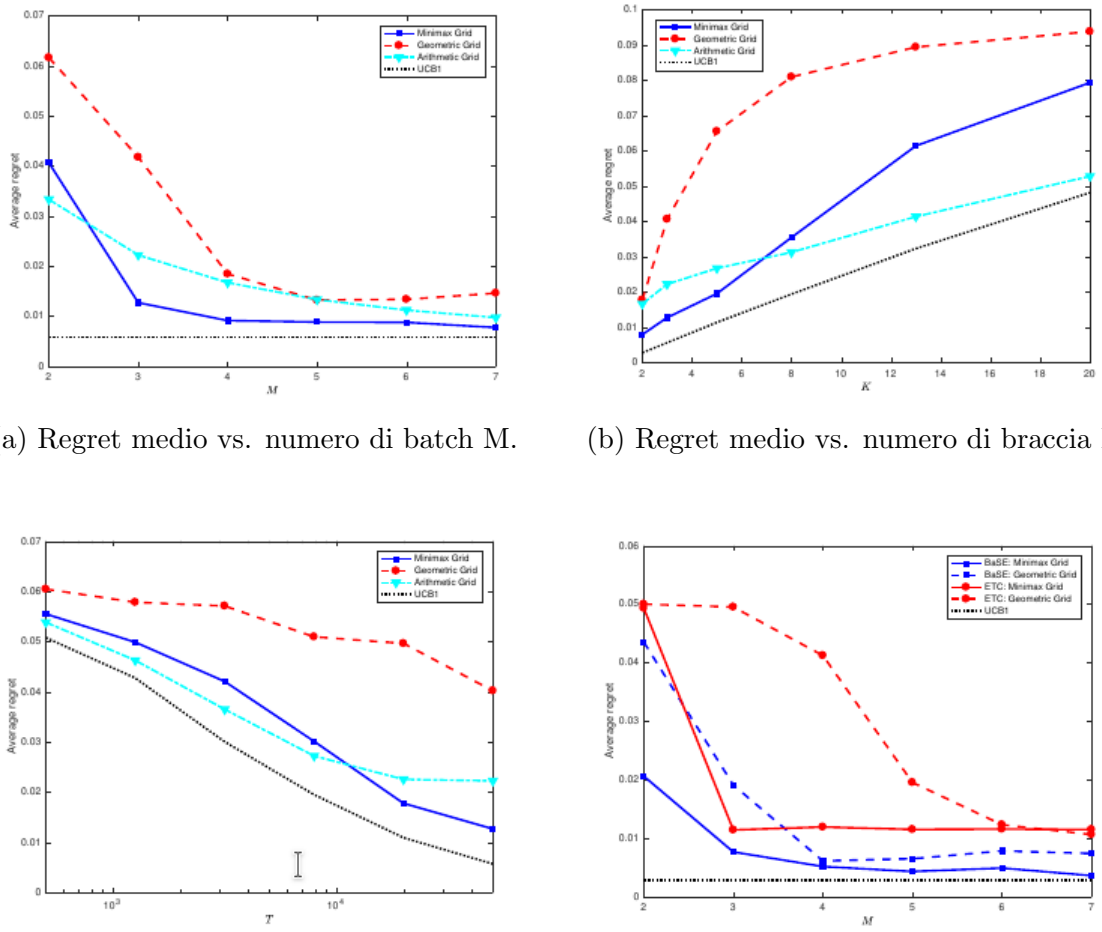
Lemma 5 Vale la seguente disuguaglianza: $\sum_{j=1}^M p_j \geq \frac{1}{2}$.

Il Lemma 4 porta a concludere che, se qualche evento A_j accade con una probabilità non piccola nel rispettivo j-esimo *contesto* (cioè considerando $P_{j,k}$ e P_M), allora la politica π ha un grande regret nel caso peggiore. L'intuizione che sta alla base di queste osservazioni risiede nel fatto che non è sempre possibile distinguere una distribuzione di un reward dalla sua versione *perturbata* dalla differenza Δ. Il Lemma 5 completa quanto detto finora nel Lemma 4 mostrando che almeno una delle quantità p_j deve essere grande.

Dunque deve esistere un qualche $p_j \geq \frac{1}{2M}$, e aggiungendo a ciò il risultato del Lemma 4 e l'arbitrarietà della politica π, si arriva al lower bound indicato nel Teorema 3 (3.5).

4 Esperimenti

In questa sezione vengono come prima cosa mostrati i risultati ottenuti dal paper. Tramite il link `https://github.com/Mathegineer/batched-bandit` è possibile accedere al codice che gli autori hanno realizzato per testare il proprio lavoro. I parametri di default sono $T = 5 \times 10^4$, $K = 3$, $M = 3$ e $\gamma = 0.5$, mentre per le medie dei reward il braccio ottimo presenta $\mu^* = 0.6$ e per tutte le altre braccia $\mu = 0.5$. Oltre alle griglie spiegate in precedenza



(a) Regret medio vs. numero di batch M.

(b) Regret medio vs. numero di braccia K.

(c) Regret medio vs. orizzonte temporale T.

(d) Confronto fra BaSE e ETC.

Figura 4.1: Grifici mostrati nel paper.

(minimax e geometrica), è stata aggiunta anche la griglia aritmetica con $t_j = \dfrac{jT}{M}$ per $j \in [M]$. Nella figura (4.1a) vengono confrontate le prestazioni dell'algoritmo BaSE con le diverse griglie e l'algoritmo UCB1 (indipendente da M) facendo variare il numero di batch M. Si nota che per $M = 4$ l'algoritmo con griglia minimax raggiunge buone prestazioni, avvicinandosi molto a quelle di UCB1. Nella figura (4.1b) vengono invece misurate le prestazioni in relazione al numero di braccia K. É chiaro come aumentare il numero di braccia porti a maggiore indecisione e a un regret maggiore, con la griglia

aritmetica che sembra comportarsi meglio al crescere di K. Nella figura (4.1c) viene invece fatto variare l'orizzonte temporale. All'aumentare del tempo, e quindi al crescere del numero dei possibili tentativi che ha il giocatore per trovare il braccio migliore, il regret medio diminuisce. Per $T < 10^4$ sembra comportarsi meglio la griglia aritmetica, mentre oltre quella soglia la griglia minimax offre prestazioni migliori. In figura (4.1d) vengono invece confrontate le prestazioni dell'algoritmo BaSE con griglia minimax e geometrica con l'algoritmo ETC di [PRCS16] (implementato per $K = 2$) con le stesse griglie. Dal grafico sembra presentare regret medio minore quasi sempre BaSE, per $2 \leq M \leq 7$.

4.1 Analisi del codice degli autori

Il materiale consiste in quattro file matlab:

(i) `UCB1.m` e `PRCS_twoarm.m`, che gli autori hanno ricavato da precedenti lavori e nei quali sono implementati gli algoritmi UCB1 e ETC,

(ii) `BASEFunc.m` e `main.m`, dove nel primo è implementato l'algoritmo ad eliminazione successiva di tipo batched e nel secondo vengono creati i grafici.

Si è deciso quindi di analizzare il file `BASEFunc.m`, vero oggetto di studio del paper. Di seguito se ne riporta il codice.

```
% function of BASE
% parameters
%     K: number of batches
%     TSeq: horizon
%     M: number of batches
%     b = T^(1/M); TGridAdaptive = floor(b.^(1:M));...,
%         TGridAdaptive = floor(TGridAdaptive/K) * K; TGridAdaptive(M) = T; ...,
%         TGridAdaptive = [0,TGridAdaptive]; % adaptive batch grids
%     a = T^(1/(2 - 2^(1-M))); TGridMinimax = floor(a.^(2.-1./2.^(0:M-1)));...,
%     TGridMinimax(M) = T; ...,
%     TGridMinimax = [0,TGridMinimax]; % minimax batch grids
%     mu: batch mean
%     gamma: tunning parameter

function [regret, activeSet] = BASEFunc(mu, K, T, M, gridType, gamma)
    % record
    regret = 0;
    if strcmp(gridType,'minimax')
        a = T^(1/(2 - 2^(1-M))); TGrid = floor(a.^(2.-1./2.^(0:M-1)));...,
        TGrid(M) = T; TGrid = [0,TGrid]; % minimax batch grids
    elseif strcmp(gridType,'geometric')
```

```
22          b = T^(1/M); TGrid = floor(b.^(1:M)); TGrid(M) = T; ...,
23          TGrid = [0,TGrid]; % geometric batch grids
24      elseif strcmp(gridType,'arithmetic')
25          TGrid = floor(linspace(0, T, M+1));
26      end
27
28      % initialization
29      activeSet = ones(1,K); numberPull = zeros(1,K); averageReward =
    zeros(1,K);
30
31      for i = 2:M+1
32          availableK = sum(activeSet);
33          pullNumber = max(floor((TGrid(i) - TGrid(i-1))/availableK), 1);
34          TGrid(i) = availableK * pullNumber + TGrid(i-1);
35          for j = find(activeSet == 1)
36              averageReward(j) = averageReward(j) * (numberPull(j)/(
    numberPull(j) ...,
37                  + pullNumber)) + (mean(randn(1,pullNumber)) + mu(j)) *
    ...,
38                  (pullNumber/(numberPull(j) + pullNumber));
39              regret = regret + (pullNumber * (mu(1) - mu(j)));
40              numberPull(j) = numberPull(j) + pullNumber;
41          end
42          maxArm = max(averageReward(find(activeSet == 1)));
43          for j = find(activeSet == 1)
44              if ((maxArm - averageReward(j)) >= sqrt(gamma * log(T*K) /
    numberPull(j)))
45                  activeSet(j) = 0;
46              end
47          end
48      end
49  end
```

Codice 4.1: Contenuto del file BASEFunc.m.

L'algoritmo necessita in input di sei parametri. Il primo è il vettore delle medie dei reward μ, a cui seguono il numero di braccia K, il valore dell'istante finale T, il numero di batch M, la tipologia della griglia (implementata come una stringa) e il valore di γ. Viene inizialmente settata a zero la variabile `regret`, che verrà ritornata a fine metodo. Si genera poi la griglia in base al valore di `gridType` in base a quanto detto nel capitolo precedente. Alla riga 29 si inizializzano le variabili `activeSet` che rappresenta l'insieme delle braccia che si possono ancora scegliere (all'inizio tutte), `numberPull` che indica il numero di volte che un braccio è stato scelto (inizialmente 0) e il reward medio `averageReward`.
Si entra poi nel primo ciclo `for` che viene ripetuto M volte. Ad ogni inizio di esplorazione batch, si calcola come prima cosa il numero di braccia ancora attive, e si calcola di conseguenza quale è il numero uguale di volte che ogni braccio attivo deve essere tirato.

Per ogni braccio attivo (ciclo for alla riga 35), viene aggiornato il proprio regret medio sommando due quantità che vengono pesate in modo diverso. In particolare, il regret medio del batch precedente viene moltiplicato per una frazione dove a numeratore si ha il numero precedente di volte in cui è stato scelto quel braccio, e al denominatore la somma fra questo numero e il numero di *pull* del batch attuale. La somma fra la media reale del proprio reward e un termine di randomizzazione (`mean(randn(1, pullNumber))`) viene moltiplicata per la frazione complementare della precedente. Queste due quantità, sommate, formano il nuovo regret medio per quel braccio. Viene poi calcolato il regret sommando a quello precedente il prodotto fra il numero di volte in cui non si è scelto il braccio migliore (che per ipotesi è il primo) e la differenza fra quest'ultimo e la media degli altri. Infine si aggiorna il numero di volte in cui un braccio è stato scelto. Si esce così dal ciclo dell'insieme delle braccia attive, passando alla fase in cui bisogna scegliere quali leve eliminare. Dapprima si trova il braccio con reward medio più alto e successivamente si cicla su tutte le braccia attive: se la differenza fra il proprio reward e quello migliore è maggiore di `sqrt(gamma * log(T*K) /numberPull(j))` (cioè $\sqrt{\gamma \log(KT)/\tau_m}$) non sarà più possibile scegliere quel particolare braccio.
Nel file `main.m` viene invece implementato il codice per confrotare le prestazioni dei diversi algoritmi e con griglie diverse. Nel seguito si mostrano solo le sezioni più rilevanti.
Nella prima sezione si confrontano le prestazioni dell'algoritmo BaSE con diverse griglie e con l'algoritmo UCB1.

```
1  regretMinimax_M = zeros(m,length(M_set));
2  regretGeometric_M = zeros(m,length(M_set));
3  regretArithmetic_M = zeros(m,length(M_set));
4  regretUCB_M = zeros(m,1);
5  mu = [mu_max, mu_min * ones(1,K-1)];
6  for iter = 1 : m
7      regretUCB_M(iter) = UCB1(mu,K,T);
8      for iter_M = 1 : length(M_set)
9          temp_M = M_set(iter_M);
10         regretMinimax_M(iter,iter_M) = BASEFunc(mu,K,T,temp_M,'minimax
   ',gamma);
11         regretGeometric_M(iter,iter_M) = BASEFunc(mu,K,T,temp_M,'
   geometric',gamma);
12         regretArithmetic_M(iter,iter_M) = BASEFunc(mu,K,T,temp_M,'
   arithmetic',gamma);
13     end
14 end
15 regretMinimax_M_mean = mean(regretMinimax_M) / T;
16 regretGeometric_M_mean = mean(regretGeometric_M) / T;
17 regretArithmetic_M_mean = mean(regretArithmetic_M) / T;
18 regretUCB_M_mean = mean(regretUCB_M) / T;
```

Codice 4.2: Analisi sul numero di batch.

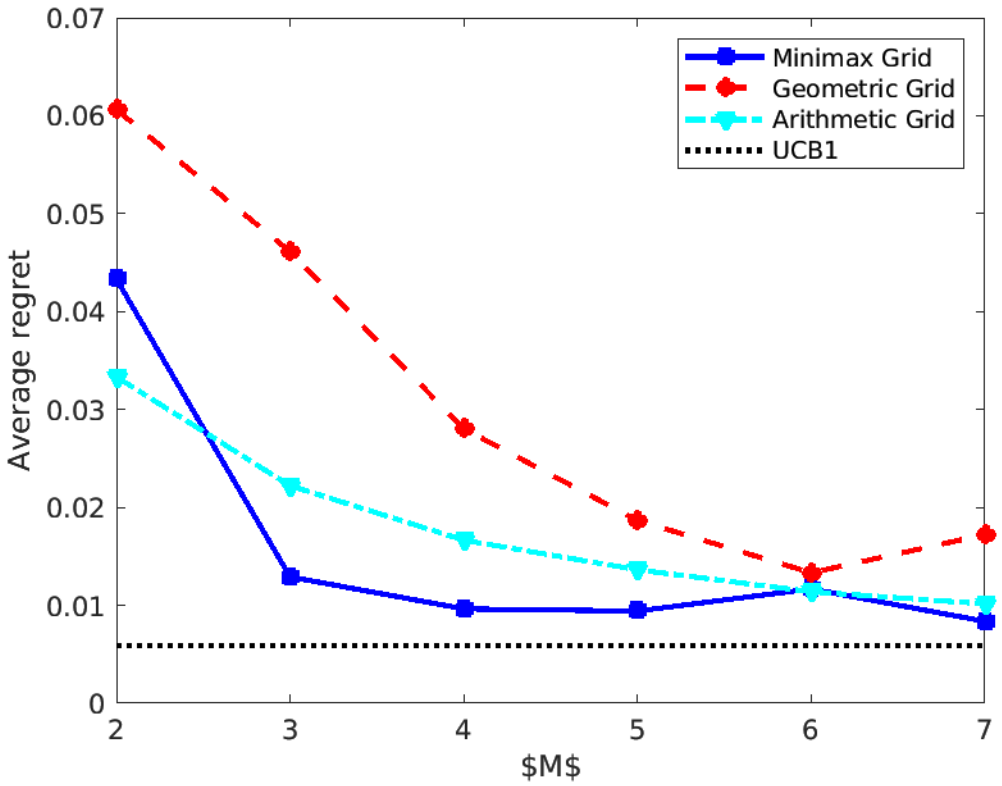


Figura 4.2: Confronto algoritimo BaSE con diverse griglie e UCB1 in relazione al numero di batch M.

Le variabili `M_set` e successivamente `K_set` e `T_set` sono vettori che presentano i particolari valori di ogni quantità che vengono fatti variare, per esempio `M_set = [2,3,4,5,6,7]`. Per ogni sezione di codice verrà mostrata la figura creata eseguendo il codice su un calcolatore locale. La figura è molto simile a quella mostrata nel paper, e l'unica cosa diversa è la poca differenza dei valori del regret medio per $M = 6$ dove nel grafico del paper questi sono leggermente più separati.

Nella seconda si confrontano gli stessi algortimi e le stesse griglie in relazione al numero K di braccia.

```
regretMinimax_K = zeros(m,length(K_set));
regretGeometric_K = zeros(m,length(K_set));
regretArithmetic_K = zeros(m,length(K_set));
regretUCB_K = zeros(m,length(K_set));
for iter = 1 : m
    for iter_K = 1 : length(K_set)
        temp_K = K_set(iter_K);
        mu = [mu_max, mu_min * ones(1, temp_K-1)];
        regretUCB_K(iter,iter_K) = UCB1(mu,temp_K,T);
        regretMinimax_K(iter,iter_K) = BASEFunc(mu,temp_K,T,M,'minimax
 ',gamma);
        regretGeometric_K(iter,iter_K) = BASEFunc(mu,temp_K,T,M,'
```

```
    geometric',gamma);
12          regretArithmetic_K(iter,iter_K) = BASEFunc(mu,temp_K,T,M,'
    arithmetic',gamma);
13      end
14  end
15  regretMinimax_K_mean = mean(regretMinimax_K) / T;
16  regretGeometric_K_mean = mean(regretGeometric_K) / T;
17  regretArithmetic_K_mean = mean(regretArithmetic_K) / T;
18  regretUCB_K_mean = mean(regretUCB_K) / T;
```

Codice 4.3: Analisi sul numero di braccia.

Il grafico risultante è il seguente, ancora molto simile a quello dello studio.

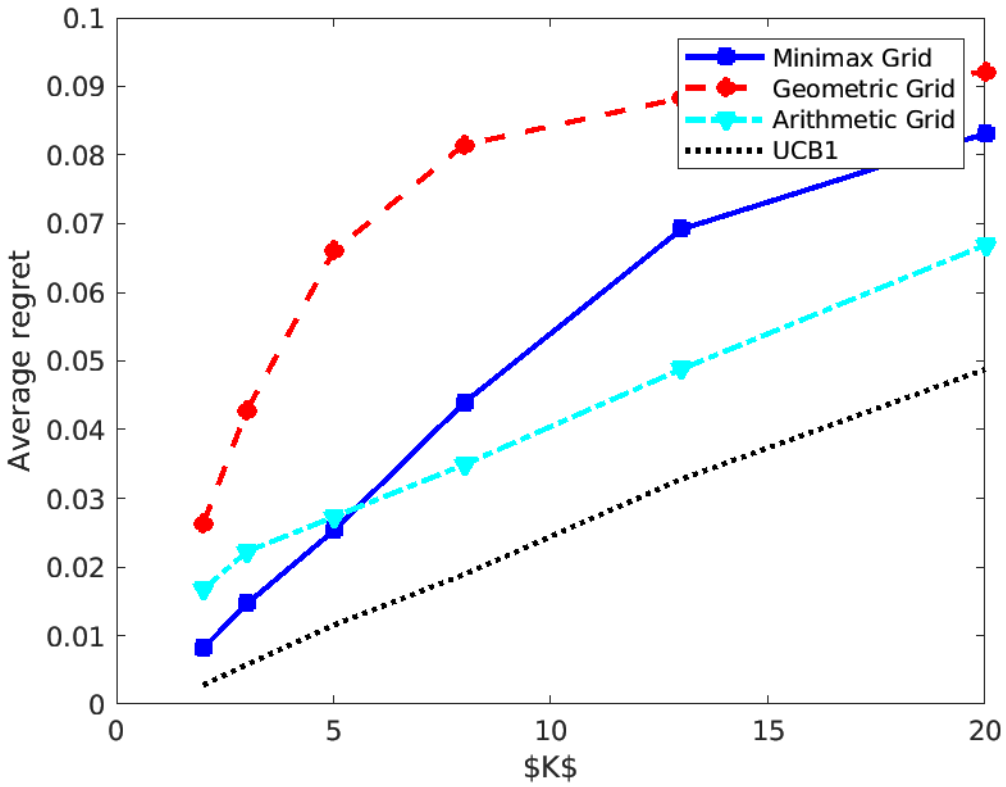


Figura 4.3: Confronto algoritimo BaSE con diverse griglie e UCB1 in relazione al numero di braccia K.

Per quanto riguarda la terza sezione, si confrontano le stesse quantità in relazione all'orizzonte temporale.

```
1  regretMinimax_T = zeros(m,length(T_set));
2  regretGeometric_T = zeros(m,length(T_set));
3  regretArithmetic_T = zeros(m,length(T_set));
4  regretUCB_T = zeros(m,length(T_set));
5  mu = [mu_max, mu_min * ones(1, K-1)];
6  for iter = 1 : m
7      for iter_T = 1 : length(T_set)
```

```
8           temp_T = T_set(iter_T);
9           regretUCB_T(iter,iter_T) = UCB1(mu,K,temp_T)/temp_T;
10          regretMinimax_T(iter,iter_T) = BASEFunc(mu,K,temp_T,M,'minimax
    ',gamma)/temp_T;
11          regretGeometric_T(iter,iter_T) = BASEFunc(mu,K,temp_T,M,'
    geometric',gamma)/temp_T;
12          regretArithmetic_T(iter,iter_T) = BASEFunc(mu,K,temp_T,M,'
    arithmetic',gamma)/temp_T;
13      end
14  end
15  regretMinimax_T_mean = mean(regretMinimax_T);
16  regretGeometric_T_mean = mean(regretGeometric_T);
17  regretArithmetic_T_mean = mean(regretArithmetic_T);
18  regretUCB_T_mean = mean(regretUCB_T);
```

Codice 4.4: Analisi sull'orizzonte temporale.

Il grafico segue in maniera precisa quanto detto e mostrato in precedenza.

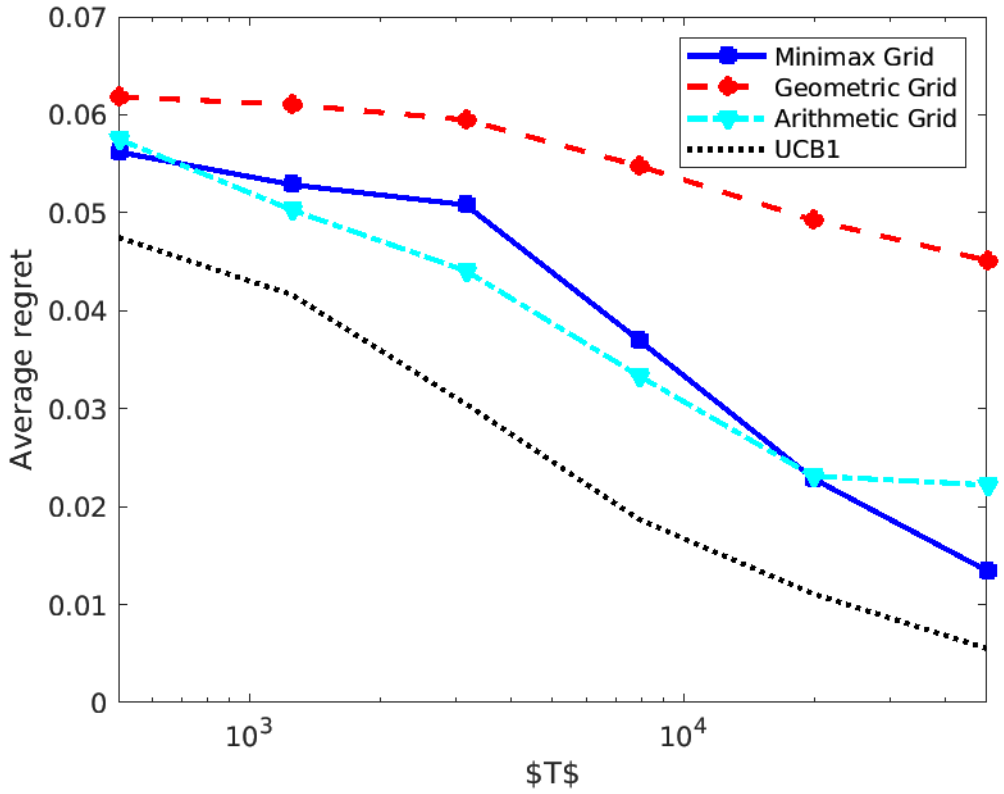


Figura 4.4: Confronto algoritimo BaSE con diverse griglie e UCB1 in relazione al tempo T.

Nell'ultima sezione invece si confrontano le prestazioni di BaSE e ETC con le griglie minimax e geometrica.

```
1  regretMinimax = zeros(m,length(M_set));
2  regretGeometric = zeros(m,length(M_set));
3  regretPRCSminimax = zeros(m,length(M_set));
```

```
regretPRCSgeometric = zeros(m,length(M_set));
regretUCB = zeros(m,1);
mu = [mu_max, mu_min];
for iter = 1 : m
    regretUCB(iter) = UCB1(mu,2,T);
    for iter_M = 1 : length(M_set)
        temp_M = M_set(iter_M);
        regretMinimax(iter,iter_M) = BASEFunc(mu,2,T,temp_M,'minimax',
 gamma);
        regretGeometric(iter,iter_M) = BASEFunc(mu,2,T,temp_M,'
 geometric',gamma);

        regretPRCSminimax(iter,iter_M) = PRCS_twoarm(mu,temp_M,T,'
 minimax');
        regretPRCSgeometric(iter,iter_M) = PRCS_twoarm(mu,temp_M,T,'
 geometric');
    end
end
regretMinimax_mean = mean(regretMinimax) / T;
regretGeometric_mean = mean(regretGeometric) / T;
regretPRCSminimax_mean = mean(regretPRCSminimax) / T;
regretPRCSGeometric_mean = mean(regretPRCSgeometric) / T;
regretUCB_mean = mean(regretUCB) / T;
```

Questa volta il grafico generato in locale è diverso da quanto mostrato nel paper. Le

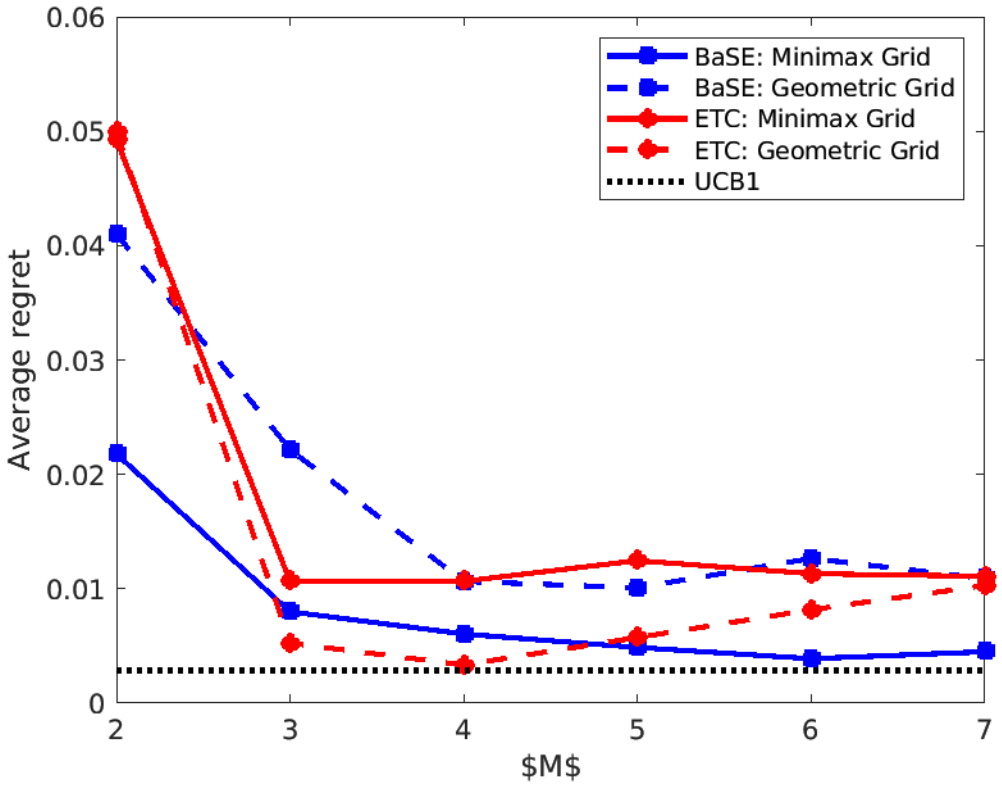
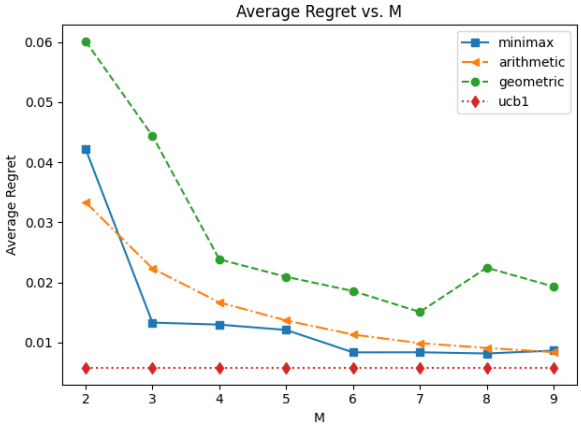


Figura 4.5: Confronto algoritimo BaSE con ETC

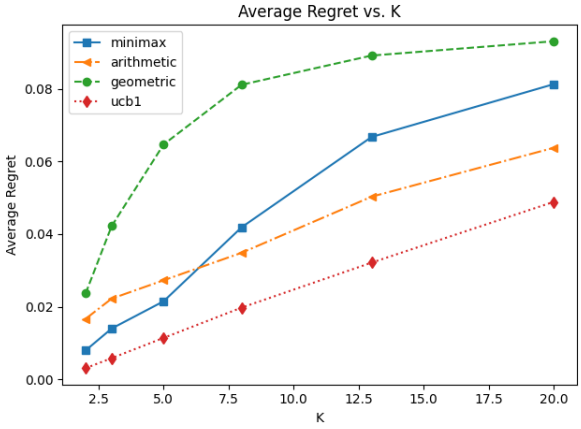
prestazioni dell'algoritmo BaSE sono molto simili a quelle viste nello studio, ma le curve relative a ETC sono molto diverse. In particolare, entrambi i regret medi di ETC sembrano migliori (perchè minori) di quelli di BaSE, contrariamente a quanto dichiarato nel paper. Per cercare di ottenere lo stesso grafico si è provato a variare singolarmente ogni possibile input, senza mai arrivare ad un risultato anche solo lontanamente simile.

4.2 Grafici con codice scritto in Python

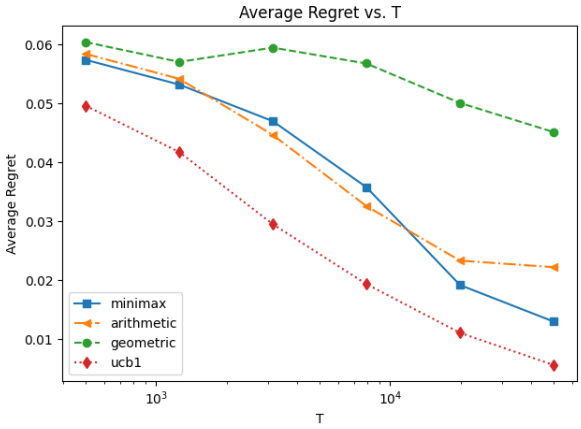
Dato che si desiderava effettuare esperimenti su dati reali, si è deciso come prima cosa di tradurre il codice Matlab degli autori in Python (di più facile comprensione), e passare poi in un secondo momento ad uno scenario con dati non casuali. Non si ripropone qui il codice, disponibile comunque al link `https://github.com/VigHub/Exam_IA`. Vengono invece presentatati i grafici ottenuti.



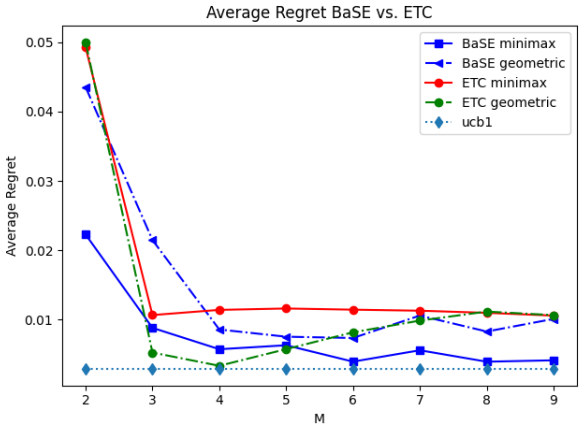
(a) Regret medio vs. numero di batch M.



(b) Regret medio vs. numero di braccia K.



(c) Regret medio vs. orizzonte temporale T.



(d) Confronto fra BaSE e ETC.

Figura 4.6: Grafici risultanti dopo aver scritto il codice degli autori in Python.

Come si può vedere, i grafici ottenuti sono molto simili a quelli ottenuti con Matlab. Si noti in particolare come nel quarto esperimento le prestazioni dell'algoritmo ETC siano

ancora diverse da quelle mostrate all'interno del paper (4.1d).

4.3 Esperimenti su dati reali

La prima questione da affrontare nel momento in cui si sceglie di replicare degli studi con dati casuali usando dati tratti da scenari reali è la scelta della tipologia di dato e dell'ambiente che si vuole considerare. Dopo aver fatto qualche ricerca si è giunti alla conclusione che lo scenario delle recensioni di prodotti su internet è facilmente adattabile al problema dei banditi.
Ogni oggetto recensito è una possibile scelta, una delle K braccia. La ricompensa che si può ottenere da questa leva è il voto che l'utente ha dato nella propria recensione, la quale simboleggia un *pull* di quello specifico braccio. L'obiettivo è quindi quello di capire quale sia, fra le K alternative, quella migliore, cioè che potrebbe portare a una recensione positiva. I dati selezionati sono quindi stati forniti da [NLM19], che presenta un insieme di database in formato json dove ognuno rappresenta delle recensioni di prodotti Amazon suddivisi per categoria.

```
{
  "reviewerID": "A2SUAM1J3GNN3B",
  "asin": "0000013714",
  "reviewerName": "J. McDonald",
  "helpful": [2, 3],
  "reviewerText": "I bought this for my husband who plays the piano.  He
   is having a wonderful time playing these old hymns.  The music  is
  at times hard to read because we think the book was published for
  singing from more than playing from.  Great purchase though!",
  "overall": 5.0,
  "summary": "Heavenly Highway Hymns",
  "UnixReviewerTime": 1252800000,
  "reviewerTime": "09 13, 2009"
}
```

Codice 4.5: Esempio di recensione.

4.3.1 Preparazione dei dati

All'interno di una singola recensione, ciò che realmente ha importanza per lo studio che si vuole effettuare è il codice identificativo dell'oggetto recensito (chiamato *asin*) e il voto dato dall'utente (il *reward*, nell'esempio sopra indicato come *overall*). Una recensione quindi contiene di per sè molti dati inutili ai fini dell'esperimento, per questo si è deciso di utilizzare un unico file di 9 GB circa in formato csv (`http://deepyeti.ucsd.edu/jianmo/amazon/categoryFilesSmall/all_csv_files.csv`) che contiene soltanto l'asin, il reward e l'id univoco della recensione. Questo file contiene quindi tutte le recensioni di tutti i database, portando ad avere una gran mole di dati. Inoltre, dato che si è deciso

di simulare lo scenario con la possibilità di poter effettuare solo una tirata di un singolo braccio in ogni istante, si è deciso di filtrare i dati originali tenendo soltanto recensioni di oggetti cosiddetti *popolari*, cioè recensiti più di 20000 volte. Per questa operazione è risultato necessario utilizzare il seguente codice:

```
import pandas as pd

chunksize = 10**7
names = ['asin', 'reviewerID', 'reward', 'reviewTime']
famous = pd.DataFrame(columns=names)
for chunk in pd.read_csv('all_rewiew.csv', dtype={'asin': str}, names=
    names, chunksize=chunksize):
     c = chunk[chunk.groupby('asin')['asin'].transform('count') > 20000]
     famous = famous.append(c)

famous.to_csv('famous.csv', index=False, header=True)
```

Codice 4.6: Estrazione delle recensioni di oggetti popolari.

Si noti che data la grandezza di diversi GB del dataset si è dovuto dividere il file in *chunk* di 10 milioni di righe per poterlo leggere. Il file creato presenta una grandezza di circa 30 MB, diventando quindi molto più semplice da usare e analizzare.

4.3.2 Cambiamenti nel codice

Si presentano ora le funzioni utilizzate per calcolare i regret nei diversi scenari. Sono ovviamente molto simili a quelle degli autori dello studio, e presentano differenze solo dove si utilizzano i dati che non vanno più generati in modo casuale ma letti da un file. É stato creato un file chiamato `load.py` dove viene caricato inizialmente il dataset e un altro file di nome `settings.py` dove è presente una funzione per stampare a video una barra di caricamento nel corso dell'esecuzione.

```
import pandas as pd

df = pd.read_csv('famous.csv', dtype={'asin': str}).sample(frac=1)
```

Codice 4.7: Lettura del dataset iniziale, si noti l'uso dell'istruzione sample che permette di mescolare i dati

UCB

```
import numpy as np
import paper_real_data.load as load


def ucb1(arms, mu, K: int, T: int) -> np.ndarray:
    """Implementazione algoritmo UCB1 con dati reali
```

```
7
8       :param arms: lista degli 'asin' per ogni braccio
9       :param mu: vettore delle medie per ogni braccio
10      :param K: numero di braccia
11      :param T: orizzonte temporale
12      :return: regret
13      """
14      arms_pulls = np.zeros(K, dtype=int)
15      pull_number = np.ones(K, dtype=int)
16      average_reward = np.zeros(K, dtype=float)
17      datas = []
18      for i in range(0, K):
19          datas.append(load.df.loc[load.df['asin'] == arms[i], 'reward'])
20      for i in range(0, K):
21          average_reward[i] = (datas[i].iloc[[arms_pulls[i]]])
22          arms_pulls[i] += 1
23
24      for t in range(K, T):
25          UCB = average_reward + np.sqrt(2 * np.log(T) / pull_number)
26          pos = np.argmax(UCB, 0).astype(int)
27          weight = 1 / (pull_number[pos] + 1)
28          arms_pulls[pos] += 1
29          average_reward[pos] = (1 - weight) * average_reward[pos] + \
30                                weight * \
31                                 (datas[pos].iloc[[arms_pulls[pos]]])
32          pull_number[pos] += 1
33
34      regret = np.dot((mu[0]-mu), pull_number)
35      return regret
```

Codice 4.8: Funzione UCB1 con dati reali

Si noti l'uso della variabile `datas`: al suo interno vengono caricate tutte le righe relative ad uno specifico braccio. Per esempio, per accedere alle righe del secondo braccio sarà sufficiente scrivere `datas[1]`. Dove nella funzione degli autori si sommava alla media del braccio una singola estrazione di una variabile casuale normale di media nulla e varianza unitaria (`mu[pos] + np.random.randn()`), si legge ora il voto dato dall'utente in una recensione relativa a quel braccio, `datas[pos].iloc[[arms_pulls[pos]]]`. Per fare questo è stato necessario introdurre la variabile `arms_pulls`, vettore che tiene in memoria quante volte è stato scelto un braccio.

BaSE

```
1 import numpy as np
2 import math
```

```python
import paper_real_data.load as load


def BASEFunc(arms, mu, K: int, T: int, M: int, grid_type: str, gamma: float) -> [float, np.ndarray]:
    """Implementazione algoritmo BaSE con dati reali

    :param arms: lista degli 'asin' per ogni braccio
    :param mu: vettore delle medie per ogni braccio
    :param K: numero di braccia
    :param T: orizzonte temporale
    :param M: numero di batch
    :param grid_type: tipologia di griglia
    :param gamma: parametro di tuning
    :return: [regret medio, insieme delle braccia attive nell'ultimo batch]
    """

    regret = 0
    datas = []
    for i in range(0, K):
        datas.append(load.df.loc[load.df['asin'] == arms[i], 'reward'])
    if grid_type == 'minimax':
        a = np.power(T, np.divide(1, np.subtract(2,
                                    np.power(2, 1.0 - M))))
        t_grid = np.floor(a ** (2 - 1 / 2 ** np.arange(0, M)))
        t_grid[M - 1] = T
        t_grid = np.insert(t_grid, 0, 0)
    elif grid_type == 'geometric':
        b = T ** (1.0 / M)
        t_grid = np.floor(b ** np.arange(1, M + 1))
        t_grid[M - 1] = T
        t_grid = np.insert(t_grid, 0, 0)
    elif grid_type == 'arithmetic':
        t_grid = np.floor(np.linspace(0, T, M+1))

    active_set = np.ones((K, 1), dtype=int)
    number_pull = np.zeros(K, dtype=int)
    average_reward = np.zeros(K, dtype=float)
    for i in range(1, M+1):
        available_k = np.sum(active_set)
        pull_number = int(np.round(np.maximum(np.floor(
                        (t_grid[i]-t_grid[i-1])/available_k), 1)))
        t_grid[i] = available_k * pull_number + t_grid[i-1]
        row_active, col_active = np.where(active_set == 1)
        for j in row_active:
            average_reward[j] = average_reward[j] * (number_pull[j]/(
```

```
    number_pull[j]+pull_number)) + \
48                  (datas[j].iloc[number_pull[j]:(number_pull[j]+
    pull_number)]).mean() * \
49                  (pull_number / (number_pull[j] + pull_number))
50              regret += (pull_number * (mu[0] - mu[j]))
51              number_pull[j] += pull_number
52
53          row_active, col_active = np.where(active_set == 1)
54          max_arm = np.max(average_reward[row_active])
55          for j in row_active:
56              if (max_arm - average_reward[j]) >= np.sqrt(gamma * math.log
    (K*T) / number_pull[j]):
57                  active_set[j] = 0
58
59      return [regret, active_set]
```

Codice 4.9: Funzione BaSE con dati reali

Come nel caso precedente, si fa uso della variabile `datas` che viene inizializzata nel ciclo `for` a riga 9. Le estrazioni di variabili casuali di media μ del singolo braccio e varianza unitaria, prima simulate con `mu[j] + np.mean(np.random.randn(1, pull_number)` vengono ora effettuate leggendo per ogni estrazione una recensione relativa ad uno specifico braccio e facendo la media dei reward ottenuti:
`datas[j].iloc[number_pull[j]:(number_pull[j] + pull_number)]).mean()`.

PRCS_twoarm

```
1  import math
2  import numpy as np
3  import paper_real_data.load as load
4
5
6  def PRCS_twoarm(arms, mu, M: int, T: int, grid_type: str) -> float:
7      """Implementazione algoritmo ETC con dati reali
8
9      :param arms: lista degli 'asin' per ogni braccio
10     :param mu: vettore delle medie per ogni braccio
11     :param T: orizzonte temporale
12     :param M: numero di batch
13     :param grid_type: tipologia di griglia
14     :return: regret
15     """
16     datas = []
17     for i in range(0, 2):
18         datas.append(load.df.loc[load.df['asin'] == arms[i], 'reward'])
```

```
    if grid_type == 'minimax':
        # a = T ** (1 / (2 - 2 ** (1-M)))
        a = np.power(T, np.divide(1, np.subtract(2, np.power(2, 1.0 - M)
))))
        t_grid = np.floor(a ** (2 - 1 / 2 ** np.arange(0, M)))
        t_grid[M - 1] = T
        t_grid = np.insert(t_grid, 0, 0)
    elif grid_type == 'geometric':
        b = T ** (1.0 / M)
        t_grid = np.floor(b ** np.arange(1, M + 1))
        t_grid[M - 1] = T
        t_grid = np.insert(t_grid, 0, 0)

    pull_number = np.round(t_grid[1] / 2).astype(int)
    # il braccio migliore e' il primo
    regret = pull_number * (mu[0] - mu[1])
    reward = np.array([
        datas[0].iloc[0: pull_number].sum(),
        datas[1].iloc[0: pull_number].sum()
    ])
    opt = 0

    for m in range(1, M):
        t = t_grid[m]
        threshold = math.sqrt(4 * math.log(2 * T / t) / t)
        if opt == 0:
            if (reward[0] - reward[1]) / t > threshold:
                opt = 1
            elif (reward[1] - reward[0]) / t > threshold:
                opt = 2
            else:
                cur_number = np.round((t_grid[m + 1] - t_grid[m]) / 2).
astype(int)
                reward += np.array([
                        datas[0].iloc[pull_number:
                                      pull_number+cur_number].sum(),
                        datas[1].iloc[pull_number:
                                      pull_number+cur_number].sum()
                ])
                pull_number += cur_number
                regret += cur_number * (mu[0] - mu[1])
        if opt == 2:  # scelgo il secondo quindi ho regret
            regret += (t_grid[m + 1] - t_grid[m]) * (mu[0] - mu[1])
        if m == (M - 2):
            if reward[0] > reward[1]:
                opt = 1
            else:
```

```
                opt = 2
    return regret
```

Codice 4.10: Funzione PRCS_twoarm con dati reali

Le differenze rispetto alla funzione degli autori sono la fase di inizializzazione di `datas` e le estrazioni casuali che diventano come al solito letture nel dataset. In particolare, dove prima veniva dichiarato il reward come somma delle estrazioni di una normale con media la media del braccio e varianza unitaria:

```
reward = np.sum([np.random.randn(pull_number, 1) + mu[0],
                 np.random.randn(pull_number, 1) + mu[1]], 1),
```

vengono ora sommate le prime `pull_number` votazioni per quell'oggetto:

```
reward = np.array([
     datas[0].iloc[0: pull_number].sum(),
     datas[1].iloc[0: pull_number].sum()
]).
```

Allo stesso modo, il reward viene ora aggiornato cosi:

```
reward  += np.array([
     datas[0].iloc[pull_number: pull_number + cur_number].sum(),
     datas[1].iloc[pull_number: pull_number + cur_number].sum()
].
```

Si mostra ora il codice nel file `main_real.py`, dove si eseguono tutti i metodi sopra descritti. Per fare considerazioni e analisi più precise sugli algoritmi, si è deciso di ripetere il numero di volte dell'esecuzione di ogni confronto un numero preciso di volte (dichiarato nella variabile `iterations`). Ad ogni iterazione, viene applicata una funzione di `sample` al dataset, in modo tale da perdere possibili errori dovuti al particolare ordine delle recensioni.

```
import numpy as np
import matplotlib.pyplot as plt

from paper_real_data import load
from paper_real_data.ucb1 import ucb1
from paper_real_data.BASEFunc import BASEFunc
from paper_real_data.PRCS_twoarm import PRCS_twoarm
import math
import settings
```

```
10
11 df = load.df
12
13 K = 3
14 T = 25000
15 M = 3
16 iterations = 10
17 gamma = 1
18 K_set = np.round(np.logspace(math.log(2, 10), math.log(15, 10),
19                               endpoint=True, num=10)).astype(int)
20 T_set = np.round(np.logspace(math.log(500, 10), math.log(T, 10),
21                               endpoint=True, num=8)).astype(int)
22 M_set = np.arange(2, 10).astype(int)
23
24 # set True if you want to do that analysis
25 m1 = True  # M batches
26 k1 = True  # K armes
27 t1 = True  # T horizon
28 c1 = True  # comparison BaSE & PRCS
29
30 # dependence on M
31 if m1:
32     p = len(M_set)
33     reg_minimax_M = np.zeros((iterations, p))
34     reg_arithmetic_M = np.zeros((iterations, p))
35     reg_geometric_M = np.zeros((iterations, p))
36     reg_ucb1_M = np.zeros(iterations)
37
38     print('Start on M')
39     for i in settings.progress_bar(range(0, iterations)):
40         df = df.sample(frac=1)
41         mu = df.groupby('asin')['reward'].mean()
42                                .sort_values(ascending=False)[1:K + 1]
43         arms = mu.keys().to_numpy()
44         reg_ucb1_M[i] = ucb1(arms, mu, K, T)
45         for m in range(0, p):
46             m_now = M_set[m]
47             reg_minimax_M[i, m] = BASEFunc(arms, mu, K, T, m_now,
48                                            'minimax', gamma)[0]
49             reg_arithmetic_M[i, m] = BASEFunc(arms, mu, K, T, m_now,
50                                               'arithmetic', gamma)[0]
51             reg_geometric_M[i, m] = BASEFunc(arms, mu, K, T, m_now,
52                                              'geometric', gamma)[0]
53
54     mean_reg_minimax_M = np.mean(reg_minimax_M, 0) / T
55     mean_reg_arithmetic_M = np.mean(reg_arithmetic_M, 0) / T
56     mean_reg_geometric_M = np.mean(reg_geometric_M, 0) / T
```

```
    mean_reg_ucb1_M = np.mean(reg_ucb1_M, 0) / T

    plt.figure(0)
    plt.plot(M_set, mean_reg_minimax_M, label='minimax', marker='s')
    plt.plot(M_set, mean_reg_arithmetic_M, label='arithmetic',
                            marker='<', linestyle='dashdot')
    plt.plot(M_set, mean_reg_geometric_M, label='geometric',
                            marker='o', linestyle='dashed')
    plt.plot(M_set, mean_reg_ucb1_M * np.ones(len(M_set)), label='ucb1',
                            marker='d', linestyle='dotted')
    plt.legend()
    plt.xlabel('M')
    plt.ylabel('Average Regret')
    plt.title('Average Regret vs. M')
    plt.show()

# dependence on K
if k1:
    p = len(K_set)
    reg_minimax_K = np.zeros((iterations, p))
    reg_arithmetic_K = np.zeros((iterations, p))
    reg_geometric_K = np.zeros((iterations, p))
    reg_ucb1_K = np.zeros((iterations, p))
    print('Start on K')
    for i in settings.progress_bar(range(0, iterations)):
        df = df.sample(frac=1)
        for k in range(0, p):
            k_now = K_set[k]
            mu = df.groupby('asin')['reward'].mean()
                            .sort_values(ascending=False)[1:k_now + 1]
            arms = mu.keys().to_numpy()
            reg_ucb1_K[i, k] = ucb1(arms, mu, k_now, T)
            reg_minimax_K[i, k] = BASEFunc(arms, mu, k_now, T, M,
                                           'minimax', gamma)[0]
            reg_arithmetic_K[i, k] = BASEFunc(arms, mu, k_now, T, M,
                                           'arithmetic', gamma)[0]
            reg_geometric_K[i, k] = BASEFunc(arms, mu, k_now, T, M,
                                           'geometric', gamma)[0]

    mean_reg_minimax_K = np.mean(reg_minimax_K, 0) / T
    mean_reg_arithmetic_K = np.mean(reg_arithmetic_K, 0) / T
    mean_reg_geometric_K = np.mean(reg_geometric_K, 0) / T
    mean_reg_ucb1_K = np.mean(reg_ucb1_K, 0) / T

    plt.figure(1)
    plt.plot(K_set, mean_reg_minimax_K, label='minimax', marker='s')
    plt.plot(K_set, mean_reg_arithmetic_K, label='arithmetic',
```

```
                                marker='<', linestyle='dashdot')
    plt.plot(K_set, mean_reg_geometric_K, label='geometric',
                                marker='o', linestyle='dashed')
    plt.plot(K_set, mean_reg_ucb1_K, label='ucb1',
                                marker='d', linestyle='dotted')
    plt.legend()
    plt.xlabel('K')
    plt.ylabel('Average Regret')
    plt.title('Average Regret vs. K')
    plt.show()

# dependence on T
if t1:
    p = len(T_set)
    reg_minimax_T = np.zeros((iterations, p))
    reg_arithmetic_T = np.zeros((iterations, p))
    reg_geometric_T = np.zeros((iterations, p))
    reg_ucb1_T = np.zeros((iterations, p))
    print('Start on T')
    for i in settings.progress_bar(range(0, iterations)):
        df = df.sample(frac=1)
        mu = df.groupby('asin')['reward'].mean()
                        .sort_values(ascending=False)[1:K + 1]
        arms = mu.keys().to_numpy()
        for t in range(0, p):
            t_now = T_set[t]
            reg_minimax_T[i, t] = BASEFunc(arms, mu, K, t_now, M,
                                           'minimax', gamma)[0] / t_now
            reg_arithmetic_T[i, t] = BASEFunc(arms, mu, K, t_now, M,
                                           'arithmetic', gamma)[0] / t_now
            reg_geometric_T[i, t] = BASEFunc(arms, mu, K, t_now, M,
                                           'geometric', gamma)[0] / t_now
            reg_ucb1_T[i, t] = ucb1(arms, mu, K, t_now) / t_now

    mean_reg_minimax_T = np.mean(reg_minimax_T, 0)
    mean_reg_arithmetic_T = np.mean(reg_arithmetic_T, 0)
    mean_reg_geometric_T = np.mean(reg_geometric_T, 0)
    mean_reg_ucb1_T = np.mean(reg_ucb1_T, 0)

    plt.figure(2)
    plt.semilogx(T_set, mean_reg_minimax_T, label='minimax', marker='s')
    plt.semilogx(T_set, mean_reg_arithmetic_T, label='arithmetic',
                                   marker='<', linestyle='dashdot')
    plt.semilogx(T_set, mean_reg_geometric_T, label='geometric',
                                   marker='o', linestyle='dashed')
    plt.semilogx(T_set, mean_reg_ucb1_T, label='ucb1',
                                   marker='d', linestyle='dotted')
```

```
151     plt.legend()
152     plt.title('Average Regret vs. T')
153     plt.xlabel('T')
154     plt.ylabel('Average Regret')
155     plt.show()
156 
157 # comparison with [PRCS16]
158 if c1:
159     p = len(M_set)
160     reg_minimax = np.zeros((iterations, p))
161     reg_geometric = np.zeros((iterations, p))
162     reg_prcs_minimax = np.zeros((iterations, p))
163     reg_prcs_geometric = np.zeros((iterations, p))
164     reg_ucb1 = np.zeros(iterations)
165     print('Start on comparison with [PRCS16]')
166     for i in settings.progress_bar(range(0, iterations)):
167         df = df.sample(frac=1)
168         mu = df.groupby('asin')['reward'].mean()
169                         .sort_values(ascending=False)[1:3]
170         arms = mu.keys().to_numpy()
171         reg_ucb1[i] = ucb1(arms, mu, 2, T)
172         for m in range(0, p):
173             m_now = M_set[m]
174             reg_minimax[i, m] = BASEFunc(arms, mu, 2, T, m_now,
175                                          'minimax', gamma)[0]
176             reg_geometric[i, m] = BASEFunc(arms, mu, 2, T, m_now,
177                                          'geometric', gamma)[0]
178             reg_prcs_minimax[i, m] = PRCS_twoarm(arms, mu, m_now, T,
179                                          'minimax')
180             reg_prcs_geometric[i, m] = PRCS_twoarm(arms, mu, m_now, T,
181                                          'geometric')
182 
183     mean_reg_minimax = np.mean(reg_minimax, 0) / T
184     mean_reg_geometric = np.mean(reg_geometric, 0) / T
185     mean_reg_ucb1 = np.mean(reg_ucb1, 0) / T
186     mean_reg_prcs_minimax = np.mean(reg_prcs_minimax, 0) / T
187     mean_reg_prcs_geometric = np.mean(reg_prcs_geometric, 0) / T
188 
189 
190 
191     plt.figure(3)
192     plt.plot(M_set, mean_reg_minimax, label='BaSE minimax',
193                                 color='blue', marker='s')
194     plt.plot(M_set, mean_reg_geometric, label='BaSE geometric',
195                     color='blue', marker='<', linestyle='dashdot')
196     plt.plot(M_set, mean_reg_prcs_minimax, label='ETC minimax',
197                     color='red', marker='o', )
```

```
198     plt.plot(M_set, mean_reg_prcs_geometric, label='ETC geometric',
199                     color='green', marker='o', linestyle='dashdot')
200     plt.plot(M_set, mean_reg_ucb1 * np.ones(len(M_set)), label='ucb1',
201                     marker='d', linestyle='dotted')
202     plt.legend()
203     plt.xlabel('M')
204     plt.ylabel('Average Regret')
205     plt.title('Average Regret BaSE vs. ETC')
206
207     plt.show()
```

Codice 4.11: Codice nel file main_real.py

Gli insiemi di braccia, orizzonte temporale e batch (`K_set`, `T_set`, `M_set`) vengono realizzati in modo analogo a quello dello studio, anche se alcuni valori sono stati cambiati. Per esempio il numero massimo di braccia è stato abbassato perchè non si disponeva di un numero sufficiente di oggetti con un numero elevato di recensioni. Inoltre anche l'orizzonte temporale è stato settato a 25000 in modo tale da non superare il numero massimo di recensioni per ogni oggetto. Sono state aggiunte quattro variabili booleane (`m1`, `k1`, `t1`, `c1`), in modo tale da eseguire solo l'analisi desiderata. Si noti anche l'uso di `settings.progress_bar(range(0, iterations))`, che genera a terminale una barra di caricamento nel corso dell'esecuzione che indica per ogni analisi la percentuale di completamento. Si è pensata a questa soluzione dal momento che occorrono diversi minuti per eseguire l'intero file. Dove si è usata la funzione per l'algoritmo BaSE si noti che è necessario indicare che si vuole ritornare solo il primo dei due elementi in output, cioè il regret. Infatti tale funzione ritorna come secondo elemento anche l'insieme delle braccia attive nell'ultimo batch, come nello studio degli autori. Infine si mette in evidenza l'uso della variabili `mu` e `arms`. Nella prima vengono inseriti per ogni braccio la media di tutte le recensioni con un'operazione di `groupby` e ordinandone i valori in senso decrescente in modo tale da avere sempre il braccio migliore al primo posto. Nella seconda vengono invece inseriti gli identificativi degli oggetti in modo da poter poi inizializzare correttamente la variabile `datas` all'interno dei metodi.

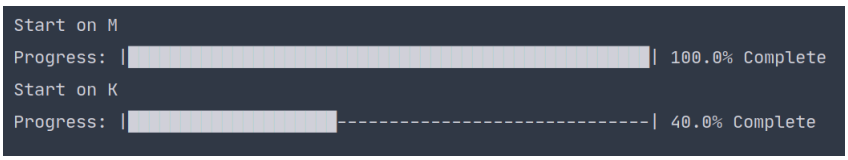
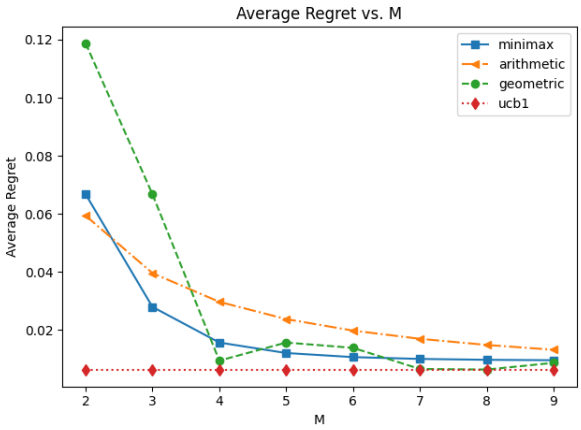


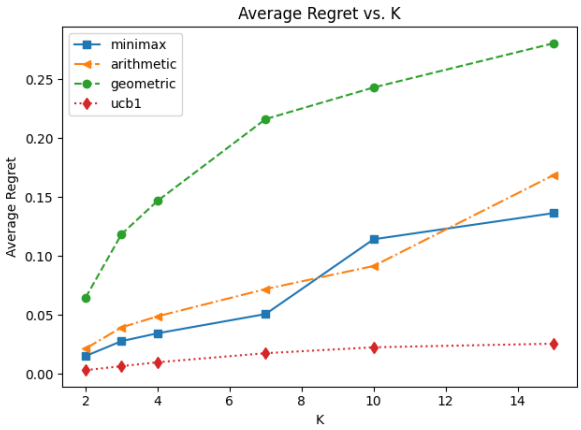
Figura 4.7: Esecuzione del file main_real.py dove si mostra la percentuale di completamento delle analisi (in PyCharm).

4.3.3 Grafici ottenuti

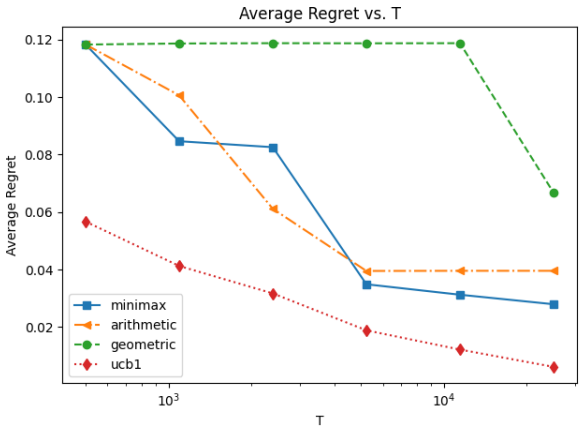
Vengono ora presentate le considerazioni e i grafici risultanti dall'esecuzione delle analisi.



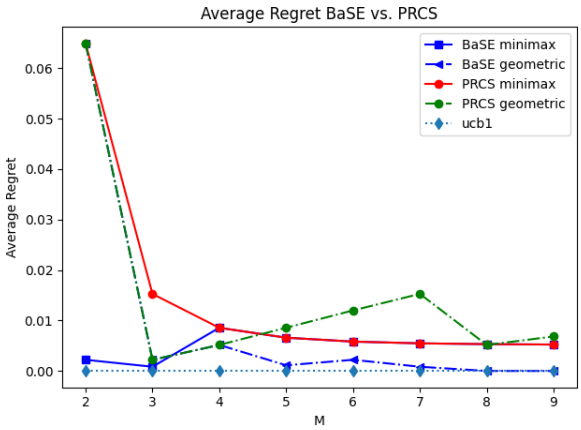
(a) Regret medio vs. numero di batch M.



(b) Regret medio vs. numero di braccia K.



(c) Regret medio vs. orizzonte temporale.



(d) Confronto fra BaSE e ETC.

Figura 4.8: Grafici con dati reali.

Analisi BaSE rispetto al numero di batch

Il grafico risultante (4.8a) è conforme con quelli dello studio, con le prestazioni di tutte le tipologie di griglie che tendono a migliorare il proprio regret all'aumentare di M.

Analisi BaSE rispetto al numero di braccia

Anche in questo caso il grafico (4.8b) rispetta quanto detto in precedenza, con il regret medio che aumenta all'aumentare del numero di possibili scelte. In particolare la griglia geometrica si comporta peggio delle griglie minimax e aritmetica, che presentano prestazioni simili.

Analisi BaSE rispetto all'orizzonte temporale

Il grafico (4.8c) mostra come il comportamento della griglia minimax e di quella aritmetica sia ancora molto simile. La griglia geometrica ha prestazioni ancora peggiori rispetto alle altre, e fino a che T non supera 10000 il suo regret medio rimane costante.

Confronto BaSE e ETC rispetto al numero di batch

Il grafico che risulta da questa analisi (4.8d) è diverso da quello presentato nel paper (4.1d), ma rimane simile a quanto ottenuto col codice degli autori (4.5), rinnovando i dubbi presentati in precedenza. In particolare si noti come per la griglia minimax molti valori di BaSE e ETC siano simili tant'è che le curve sono coincidenti per M maggiore di 4.

5 Conclusioni

L'algoritmo BaSE proposto dagli autori è una soluzione semplice ed elegante al problema dei banditi a K braccia in configurazione batch. Tutto ciò che il giocatore deve fare è scegliere uno stesso numero di volte ogni braccio dal primo al penultimo batch, e ad ogni fine ciclo esplorativo rimuovere quelli che superano un certo valore dinamico di soglia. Nell'ultimo batch si continua a scegliere il braccio che ha portato al rewad medio maggiore. Per alti valori del numero di batch le prestazioni dell'algoritmo sono simili a quelle teoriche ottime di UCB1. La scelta della griglia, cioè il modo in cui si decide di dividere gli intervalli temporali, influisce notevolmente sulle prestazioni, ognuna presentando pregi e difetti rispetto a una particolare analisi, come visto nello scorso capitolo.

I limiti teorici ricavati dallo studio sono di fondamentale importanza per il problema dei banditi, in quanto caratterizza e quantifica l'effetto del numero di braccia sul regret, aspetto che non era ancora stato considerato con precisione per $K > 2$. Inoltre viene provato come sia possibile raggiungere lower bound ottimi con qualunque tipo di griglia.

Gli autori dello studio affermano che il proprio algoritmo si comporta nettamente meglio di quello mostrato in [PRCS16] con griglia geometrica anche se, come ripetuto più volte, non si è riusciti a confermare questo risultato, anche con il codice fornito dagli autori. Quello che è certo è che BaSE funziona con un numero di braccia $K \geq 2$, laddove il secondo è solo per $K = 2$. Ciò porta quindi BaSE a essere uno degli algoritmi più importanti quando si parla di Batched Multi-armed Bandits Problem, e in generale la strategia risolutiva potrebbe essere applicata ai più svariati problemi di Reinforcement Learning.

In particolare Batched Successive Elimination potrebbe portare a un risparmio significativo di tempo, risorse e denaro per i trial clinici, portando ad avere risultati accettabili anche già con pochi insiemi di campioni. La sempre più crescente personalizzazione dei contenuti e delle preferenze online e dei servizi potrebbe aumentare a un ritmo ancora maggiore facendo uso di questa strategia, scartando subito e con buoni intervalli di confidenza ciò che non piace all'utente (cioè che porta a una ricompensa di valore inferiore rispetto ad altre). In ambito finanziario BaSE può far capire quale azione possa portare al profitto maggiore, considerando comunque una certa quantità di rischio dato che sicuramente si compie qualche investimento sbagliato in fase esplorativa.

6 Considerazioni personali

All'inizio dell'attività di analisi il paper preso in considerazione pareva complesso. Solo dopo aver letto buona parte del materiale citato dal lavoro e dopo aver deciso di eseguire ogni singolo passaggio matematico cercando di ricavare singolarmente quanto detto dallo studio si è riuscito a capirne realmente il contenuto. Si è rivelata un'attività molto stimolante e impegnativa che, come tale, ha portato notevole soddisfazione una volta conclusa.

Realizzare gli algoritmi proposti in modo tale da poterli eseguire su dati reali ha richiesto un notevole impiego di tempo nella ricerca del giusto tipo di dato, facendo capire che ogni particolare scenario presenta particolari modelli per essere descritto e specifiche strategie risolutive. Risulta quindi essenziale conoscere più tecniche possibile, perchè nessuna soluzione può adattarsi a ogni problema. Inoltre spesso può presentarsi uno scenario che necessita di più tecniche insieme per essere risolto.

Il reinforcement learning era un argomento già di per sè molto interessante, diventando ora dopo questo approfondimento l'oggetto di studio che più attrae chi sta scrivendo. Si è infatti continuata la lettura di documenti scientifici che riguardano questa tipologia di apprendimento automatico, analizzando diversi problemi. Il primo articolo che si è letto è relativo allo scenario della *Mounting Car* [Moo90], dove una macchina è situata fra due colline in una conca, e deve capire come muoversi per sfruttare l'inerzia e la forza di gravità per arrivare in cima alla seconda collina. Si è poi proseguito con il problema del controllo di un *Pendolo Inverso* [LB10], dove bisogna tenere un oggetto legato a una corda su un perno in posizione verticale. Infine si è letto [KBP13], dove si descrivono scenari di controllo di robot. In particolare ci si è soffermati sulla sezione riguardante il controllo di droni autonomi in fase di atterraggio. Tutte queste letture hanno confermato e addirittura aumentato l'interesse verso l'apprendimento per rinforzo, che diventerà con molta probabilità un aspetto principale del lavoro di tesi realtivo al prossimo anno.

Questo lavoro non è quindi stato solo un'occasione per approfondire un problema specifico, ma una vera e propria opportunità per capire appieno come comportarsi davanti a un lavoro scientifico che tratta di un ambito mai affrontato in precedenza.

Materiale consultato

[Gar20] Saket Garodia. *Contextual multi-armed bandit - (Intuition behind Netflix Artwork Recommendation)*. 7.05.2020. URL: https://medium.com/analytics-vidhya/contextual-multi-armed-bandit-intuition-behind-netflix-artwork-recommendation-b221a983c1cb. Consultato il 14.06.2020.

[KBP13] Jens Kober, J. Andrew Bagnell e Jan Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274. DOI: 10.1177/0278364913495721. eprint: https://doi.org/10.1177/0278364913495721. URL: https://doi.org/10.1177/0278364913495721.

[LB10] Kent H. Lundberg e Taylor W. Barton. "History of Inverted-Pendulum Systems". In: *IFAC Proceedings Volumes* 42 (24 2010), pp. 131–135. ISSN: 1474-6670. DOI: 10.3182/20091021-3-jp-2009.00025.

[Moo90] Andrew William Moore. *Efficient Memory-based Learning for Robot Control*. Rapp. tecn. 1990.

[NLM19] Jianmo Ni, Jiacheng Li e Julian McAuley. "Justifying recommendations using distantly-labeled reviews and fined-grained aspects". In: *Empirical Methods in Natural Language Processing* (2019).

[Pat17] Massimiliano Patacchiola. *Dissecting Reinforcement Learning-Part.6*. 14.08.2017. URL: https://mpatacchiola.github.io/blog/2017/08/14/dissecting-reinforcement-learning-6.html. Consultato il 12.06.2020.

Studi citati dal paper

[AA88] Noga Alon e Yossi Azar. "Sorting, approximate sorting, and sear-ching in rounds". In: *SIAM Journal on Discrete Mathematics* 1.3 (1988), pp. 269–280.

[AAAK17] Arpit Agarwal, Shivani Agarwal, Sepehr Assadi e Sanjeev Khanna. "Learning with limited rounds of adaptivity: Coin tossing, multi-armed bandits, and ranking from pairwise comparisons". In: *Conference onLearning Theory* (2017), pp. 39–75.

[AB09] Jean-Yves Audibert e Sébastien Bubeck. "Minimax policies for adversarial and stochastic bandits". In: *COLT* (2009), pp. 217–226.

[AB10] Jean-Yves Audibert e Sébastien Bubeck. "Regret bounds and mini-max policies under partial monitoring". In: *Journal of Machine Learning Research* 11.Oct (2010), pp. 2785–2836.

[ACF02] Peter Auer, Nicolò Cesa-Bianchi e Paul Fischer. “Finite-time analysis of the multi- armed bandit problem”. In: *Machine learning* 47.2-3 (2002), pp. 235–256.

[AMS09] Jean-Yves Audibert, Remi Munos e Csaba Szepesvar. “Exploration–exploitation tradeoff using variance estimates in multi-armed bandits”. In: *Theoretical Computer Science,* 410.19 (2009), pp. 1876–1902.

[AO10] Peter Auer e Ronald Ortne. “UCB revisited: Improved regretbounds for the stochastic multi-armed bandit problem”. In: *Periodica Mathematica Hungarica* 61.1-2 (2010), pp. 55–65.

[AWBR09] Alekh Agarwal, Martin J. Wainwright, Peter L. Bartlett e Pradeep K. Ravikumar. “Information-theoretic lower bounds on the oracle complexity of convex optimization”. In: *Advances in Neural Information Processing Systems* (2009), pp. 1–9.

[BC12] Sébastien Bubeck e Nicolò Cesa-Bianchi. “Regret analysis of stochastic and non-stochastic multi-armed bandit problems”. In: *Foundations and Trends® in Machine Learning* 5.1 (2012), pp. 1–122.

[BK97] Apostolos N. Burnetas e Michael N. Katehakis. “Optimal adaptive policies for markov decision processes”. In: *Mathematics of Operations Research* 22.1 (1997), pp. 222–255.

[BM07] Dimitris Bertsimas e Adam J Mersereau. “A learning approach for interactive marketing to a customer segment”. In: *Operations Research* 55.6 (2007), pp. 1120–1135.

[BMW16] Mark Braverman, Jieming Mao e S. Matthew Weinberg. “Parallel algorithms for select and partition with noisy comparisons”. In: *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing* (2016), pp. 851–862.

[BPR13] Sébastien Bubeck, Vianney Perchet e Philippe Rigollet. “Bounded regret in stochastic multi-armed bandits”. In: *Proceedings of the 26th Annual Conference on Learning Theory* (2013), pp. 122–134.

[BT83] Béla Bollobás e Andrew Thomason. “Parallel sorting”. In: *Discrete Applied Mathematics* 6.1 (1983), pp. 1–11.

[CDS13] Nicolò Cesa-Bianchi, Ofer Dekel e Ohad Shamir. “Online learning with switching costs and other adaptive adversaries”. In: *Advances in Neural Information Processing Systems* (2013), pp. 1160–1168.

[CG09] Stephen E. Chick e Noah Gans. “Economic analysis of simulation selection problems”. In: *Management Science* 55.3 (2009), pp. 421–437.

[CT06] Thomas M. Cover e Joy A. Thomas. *Elements of Information Theory*. seconda edizione. Wiley, New York, 2006.

[DKMR14] Susan Davidson, Sanjeev Khanna, Tova Milo e Sudeepa Roy. "Top-k and clustering with noisy comparisons". In: *ACM Transactions on Database Systems (TODS)* 39.4 (2014).

[DRY18] John Duchi, Feng Ruan e Chulhee Yun. "Minimax bounds on stochastic batched convex optimization". In: *Conference On Learning Theory* (2018), pp. 3065–3162.

[EKMM19] Hossein Esfandiari, Amin Karbasi, Abbas Mehrabian e Vahab Mirrokni. "Batched multi-armed bandits with optimal regret". In: *arXiv preprint arXiv* 1910.04959 (2019).

[EMM06] Eyal Even-Dar, Shie Mannor e Yishay Mansour. "Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems". In: *Journal of machine learning research* 7.Jun (2006), pp. 1079–1105.

[FRPU94] Uriel Feige, Prabhakar Raghavan, David Peleg e Eli Upfal. "Computing with noisy information". In: *SIAM Journal on Computing* 23.5 (1994), pp. 1001–1018.

[GC11] Aurélien Garivier e Olivier Cappé. "The kl-ucb algorithm for bounded stochastic bandits and beyond". In: *Proceedings of the 24th annual conference on learning theory* (2011), pp. 359–376.

[JJNZ16] Kwang-Sung Jun, Kevin G. Jamieson, Robert D. Nowak e Xiaojin Zhu. "Top arm identification in multi-armed bandits with batch arm pulls". In: *AISTATS* (2016), pp. 139–148.

[KCS08] Aniket Kittur, Ed H. Chi e Bongwon Suh. "Crowdsourcing user studies with mechanical turk". In: *Proceedings of the SIGCHI conference on human factors in computing systems* (2008), pp. 453–456.

[LR85] Tze Leung Lai e Herbert Robbins. "Asymptotically efficient adaptive allocation rules". In: *Advances in applied mathematics* 6.1 (1985), pp. 4–22.

[NY83] Arkadii Semenovich Nemirovsky e David Borisovich Yudin. *Problem complexity and method efficiency in optimization*. Wiley, 1983.

[PR13] Vianney Perchet e Philippe Rigollet. "The multi-armed bandit problem with covariates". In: *The Annals of Statistics* (2013), pp. 693–721.

[PRCS16] Vianney Perchet, Philippe Rigollet, Sylvain Chassang e Erik Snowberg. "Batched bandit problems". In: *The Annals of Statistics* 44.2 (2016), pp. 660–681.

[Rob52] Herbert Robbins. "Some aspects of the sequential design of experiments". In: *Bulletin of the American Mathematical Society* 58.5 (1952), pp. 527–535.

[Sha13] Ohad Shamir. "On the complexity of bandit and derivative-free stochastic convex optimization". In: *Conference on Learning Theory* (2013), pp. 3–24.

[Tho08] William R. Thompson. "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples". In: *Biometrika* 25.3/4 (2008), pp. 285–294.

[Tsy08] A. Tsybakov. *Introduction to Nonparametric Estimation*. Springer-Verlag, 2008.

[Val75] Leslie G. Valiant. "Parallelism in comparison problems". In: *SIAM Journal on Computing* 4.3 (1975), pp. 348–355.

[Vog60] Walter Vogel. "A sequential design for the two armed bandit". In: *The Annals of Mathematical Statistics* 31.2 (1960), pp. 430–443.