UNIVERSITAT DE LLEIDA
Escola Politècnica Superior
Grau en Enginyeria Informàtica
Models de Computació i Complexitat

Restless Bandit

Joaquim Picó Mora, Sergi Simón Balcells

PraLab2

Professorat : M.Valls

Data : Divendres 27 de Març

Contents

1 Introducció

2 Context

El teu estomac gruny. Vas al restaurant Italià que t'encanta, o proves el nou Thailandes que acaba d'obrir? Hi vas amb el teu millor amic o amb una persona que no coneixes tant i que vols coneixer millor? Masa dificil, millor quedar-se a casa. Cuines aquella recepta que t'encanta, o optes per buscar-ne alguna de nova per Internet? Saps que, potser millor demanar una pizza a domicili. Demanes la teva preferida o preguntes per les especials? Tant dubtar entre una o l'altra te n'hauràs cansat abans de poguer fer la primera mossegada.

Cada dia ens veiem forçats a fer decisions entre dos opcions, que difereixen en dos dimensions: Ens quedem amb les nostres coses preferides, o n'explorem de noves? Intuitivament podem pensar que la vida és un balanç entre les dues, pero la pregunta és: Quin és el balanç?

Molts matemàtics i informàtics han estat treballant en aquest balanç des de fa més de 50 anys donant-li el nom de explore/exploit tradeoff.

En computació, la tensió entre explorar o explotar pren la seva forma més concreta an l'escenari anomenat multi-armed bandit, o k-armed bandit. Aquest nom li és donat degut a que es la forma coloquial de referir-se a les màquines escura-butxaques. Imagina entrar a un cassino ple de màquines escura-butxaques, cada una amb les seves possibilitats de fer una tirada guanyadora. Naturalment, s'està interessat en maximitzar els guanys. Està clar que hi haura una fase d'exploració en la qual testegarem les màquines, i una altra d'explotació tirant d'aquelles que creiem que són més beneficioses.
La primera passa cap a la solució va ser l'algorisme Win-Stay, Lose-Shift, proposat per Herbert Robins. Aquest consisteix en triar a una màquina aleatoria, mentres s'obtingui profit jugant en aquella màquina, es continua jugant en la

mateixa i, si després d'una certa tirada la màquina deixa de ser profitosa, es canvia a una altra. Aquesta tot i estar lluny d'una solució optima, es va demostrar que els resultats eren millors que els de la pura sort.
No va ser fins al 1970 que John Gittins va trobar una solució optima que resolia el problema. Gittins va enfocar el problema en termes de maximitzar els guanys per un futur que és interminable pero amb 'descontes'. Fent així la asumsió de que el valor assignat als guanys decreixia geométricament. Per exemple, es creu que hi ha un 1% de probabilitats de ser atropellat per un autobus un día, aleshores s'ha de valorar el sopar del seguent día un 99% del valor del d'aquesta nit, només perque l'endemà potser mai s'arriba a sopar. D'aquesta forma, va arribar a la conclusió de que cada màquina de la qual en sabem una mica o res, té un nombre que ens indica la probabilitat de guany que ens farà decidir si tornar a jugar en ella o no. Aquest nombre és conegut com l'index de Gittins.

Una variació d'aquest problema(multi-armed bandit), és que cada una de les màquines escura-butxaques es comporta com una màquina Markov. Es a dir, cada cop que una màquina en particular és jugada, l'estat d'aquesta canvia a un nou escollit dacord a l'evolució de probabilitats dels estats d'aquesta màquina de Markov. I si a la variació anterior se li aplica, que l'estat de les màquines no jugades pot evolucionar al llarg del temps, apareix el problema del restless bandit.

3 Desenvolupament de la temàtica

Dins de les complexitats que poden tenir els problemes, la complexitat de PSPACE compleix:

$$NP \subseteq PSPACE \subseteq EXP$$

De la mateixa manera que no es sap si $P = NP$ tampoc es sap si $NP = PSPACE$. La demostració que NP es contingut dins de PSPACE es realitzar per reducció a l'absurd:

Sigui M una màquina de Turing NP, és a dir, donada una instància d'un problema ens diu en un temps polinomial si aquest pertany al problema. Si l'espai per a desenvolupar l'algorisme fos més gran a polinòmic, llavors forçosament per a llegir o escriure aquesta informació es necessitaria aquest temps, pel que arriba a l'absurd amb la definició de la màquina M.

El problema de *Restless bandit* es troba dincs la complexitat PSPACE. Això va ser demostrat l'any 1999 en l'article [].En aquest s'explica un problema de xarxes que es demostrar ser exponencial. Al mateix temps, un problema relaxat d'aquest es demostra ser PSPACE-complet, és a dir, tots els problemes de PSPACE poden ser reduïts a aquest problema i aquest pot ser reduït a tots els problemes de PSPACE. Finalment es dóna una fórmula de cost del problema de *restless bandit* i es redueix el problema relaxat a aquest, demostrant que és PSPACE-hard (tots els problemes de PSPACE són reduïbles a aquest).

3.1 Upper confidence bound

Que el problema es mostri com a PSPACE-hard significa que el càlcul del mínim és, si no es demostra la igualtat entre les diferents complexitats, tan dificultós que deixa de ser pràctic intentar-ho. Per sort, el problema relaxat a un *k-armed bandits* dona una bona heurística al problema principal. Aquest ja hem dit que és decidible amb una complexitat P per l'índex de Gittins, però el problema recau en el cost de calcular aquest índex, ja que tot i que sigui polinomial en el temps el cost és bastant elevat, pel que s'utilitzen altres algoritmes que tot i no ser perfectes es queden molt propers a l'òptim del problema relaxat. Amb aquests algorismes trobem el d'*Upper confidence bound*.

L'algorisme UCB fa les seleccions de a quina màquina jugar en base a l'optimisme. Es a dir, centrant-nos en el millor que pot acabar sent una acció. Seguint aquesta estrategia, per exemple una heurística possible és fer la selecció de la màquina

seguint la seguent formula:

On Qn(a) és el valor actual d'una màquina escura-butxaques a. El valor sota l'arrel és el logartime del nombre de màquines a les que hem jugat dividit per kn, que és el nombre de tirades que hem fet en la màquina a. I finalment c és una constant.
Per tal de calcular Qn(a) podem fer una mitjana:

On Rn és la recompensa que obtenim de realitzar aquesta acció.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
class ucb_bandit:
    '''
    Upper Confidence Bound Bandit

    Inputs
    ==========================================
    k: number of arms (int)
    c:
    iters: number of steps (int)
    mu: set the average rewards for each of the k-arms
        .
        Set to "random" for the rewards to be selected
            from
        a normal distribution with mean = 0.
        Set to "sequence" for the means to be ordered
           from
```

```
    0 to k-1.
    Pass a list or array of length = k for user-
        defined
    values.
'''
def __init__(self, k, c, iters, mu='random'):
    # Number of arms
    self.k = k
    # Exploration parameter
    self.c = c
    # Number of iterations
    self.iters = iters
    # Step count
    self.n = 1
    # Step count for each arm
    self.k_n = np.ones(k)
    # Total mean reward
    self.mean_reward = 0
    self.reward = np.zeros(iters)
    # Mean reward for each arm
    self.k_reward = np.zeros(k)

    if type(mu) == list or type(mu).__module__ ==
        np.__name__:
        # User-defined averages
        self.mu = np.array(mu)
    elif mu == 'random':
        # Draw means from probability distribution
        self.mu = np.random.normal(0, 1, k)
    elif mu == 'sequence':
```

```
        # Increase the mean for each arm by one
        self.mu = np.linspace(0, k-1, k)

def pull(self):
    # Select action according to UCB Criteria
    a = np.argmax(self.k_reward + self.c * np.sqrt
       (
            (np.log(self.n)) / self.k_n))

    reward = np.random.normal(self.mu[a], 1)

    # Update counts
    self.n += 1
    self.k_n[a] += 1

    # Update total
    self.mean_reward = self.mean_reward + (
        reward - self.mean_reward) / self.n

    # Update results for a`k
    self.k_reward[a] = self.k_reward[a] + (
        reward - self.k_reward[a]) / self.k_n[a]

def run(self):
    for i in range(self.iters):
        self.pull()
        self.reward[i] = self.mean_reward

def reset(self, mu=None):
    # Resets results while keeping settings
```

```
        self.n = 1
        self.k_n = np.ones(self.k)
        self.mean_reward = 0
        self.reward = np.zeros(iters)
        self.k_reward = np.zeros(self.k)
        if mu == 'random':
            self.mu = np.random.normal(0, 1, self.k)
```

Aquesta és la classe en la qual definim el problema. Per tal utilitzar-la i obtenir-ne resultats, podem executar el seguent còdi:

```
k = 10 # number of arms
iters = 1000
ucb_rewards = np.zeros(iters)
# Initialize bandits
ucb = ucb_bandit(k, 2, iters)
episodes = 1000
# Run experiments
for i in range(episodes):
    ucb.reset('random')
    # Run experiments
    ucb.run()

    # Update long-term averages
    ucb_rewards = ucb_rewards + (
        ucb.reward - ucb_rewards) / (i + 1)

plt.figure(figsize=(12,8))
plt.plot(ucb_rewards, label="UCB")
plt.legend(bbox_to_anchor=(1.2, 0.5))
plt.xlabel("Iterations")
```

```
plt.ylabel("Average␣Reward")
plt.title("Average␣UCB␣Rewards␣after␣"
          + str(episodes) + "␣Episodes")
plt.show()
```

Aquest còdi ens farà un plot en el qual podem veure l'evolució de l'agent:

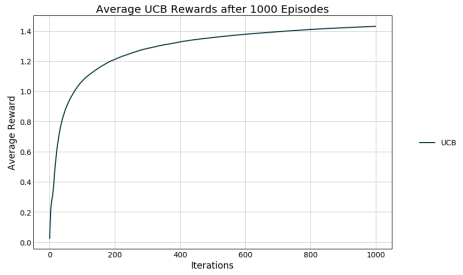

Figure 1: Result plot simulation multi-armed bandit

4 Conclusions

5 Bibliografia