Online Learning Applications

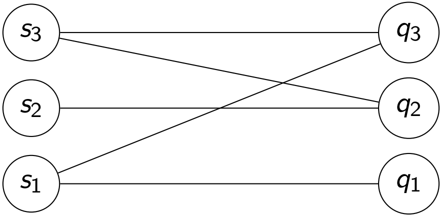
Part 9: Matching and combinatorial bandits

Introduction to matching

(Offline) Matching

Offline **matching** problem:

- A set of items $\mathcal{S}$
- A set of items $\mathcal{Q}$
- The goal is to match each item from $\mathcal{S}$ with (at most) an item from $\mathcal{Q}$
- The reward depends on the "quality" of the matching



Matching

We focus on a **weighted** matching problem:

- Weight $w(s, q) \in [0, 1]$ for each $s \in \mathcal{S}$ and $q \in \mathcal{Q}$
- A matching $\mathcal{M}$ is a set of couples (s, q) such that each item appears in at most one couple

goal

Find the matching $\mathcal{M}$ that maximizes

$$\sum_{(s,q)\in\mathcal{M}} w(s, q).$$

Example: find a perfect matching

We can model the problem of finding a **perfect matching** with a weighted matching problem.

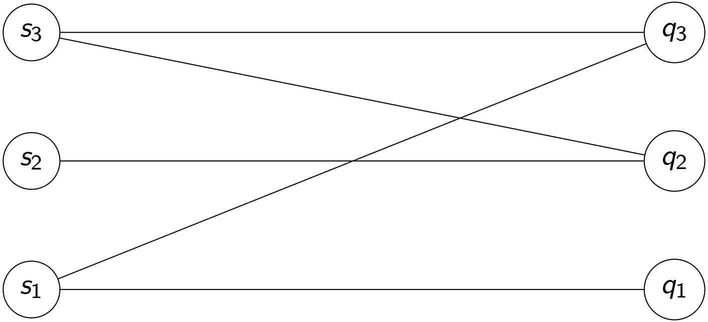
Perfect matching problem

- A set of items $\mathcal{S}$
- A set of items $\mathcal{Q}$
- The two sets have equal size $|\mathcal{S}| = |\mathcal{Q}|$
- A set of feasible matches $\mathcal{F} \subseteq \mathcal{S} \times \mathcal{Q}$, i.e., couples of items that can be matched
- **Goal**: find a feasible matching such that each item is matched (if it exists). Formally, find a matching $\mathcal{M} \subseteq \mathcal{F}$ such that each element is matched.

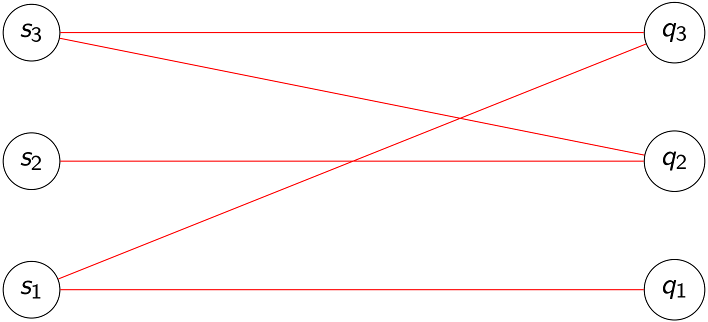
This is a special case of weighted matching in which:

- $w(s, q) = 1$ if $(s, q) \in \mathcal{F}$
- $w(s, t) = 0$ if $(s, q) \notin \mathcal{F}$

Example: find a perfect matching

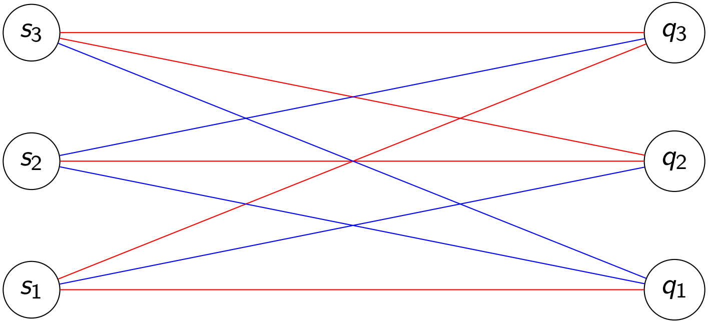


Example: find a perfect matching



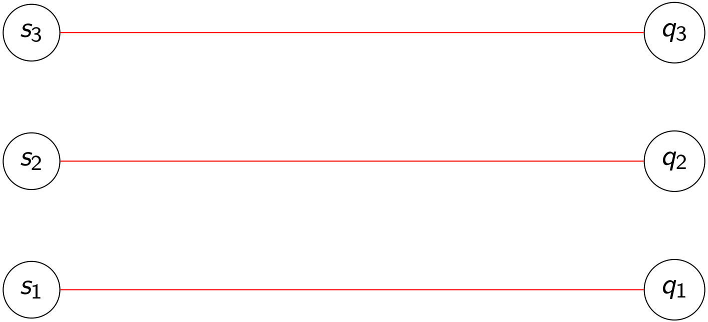
Red edges have weight 1

Example: find a perfect matching



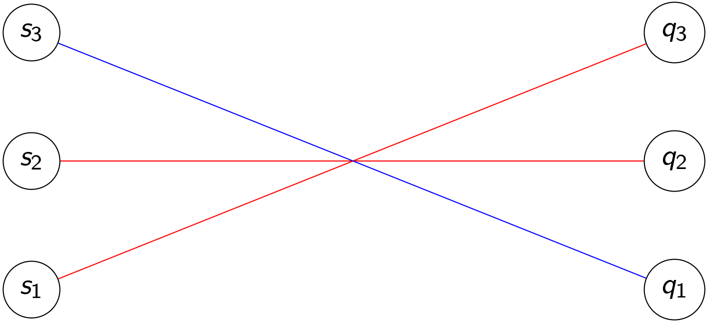
Blue edges have weight 0

Example: find a perfect matching



A **perfect** matching has value 3

Example: find a perfect matching



An **non-perfect** matching has value strictly smaller than 3 (in this case 2)

Example: find a maximum matching

We can model the problem of finding a **maximum matching** with a weighted matching problem.

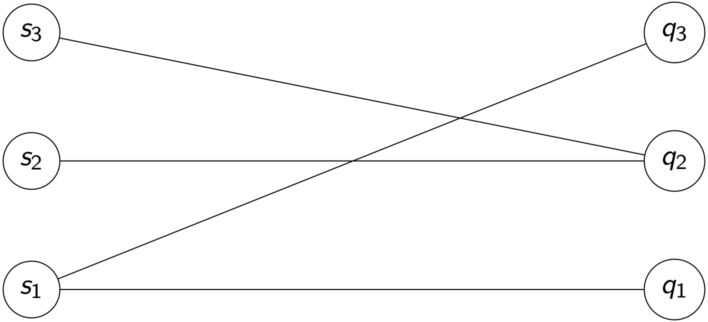
Feasible matching problem

- A set of items $\mathcal{S}$
- A set of items $\mathcal{Q}$
- A set of feasible matches $\mathcal{F} \subseteq \mathcal{S} \times \mathcal{Q}$
- **Goal**: find a matching of **maximum size**. Formally, find a matching $\mathcal{M} \subseteq \mathcal{F}$ with highest cardinality $|\mathcal{M}|$.

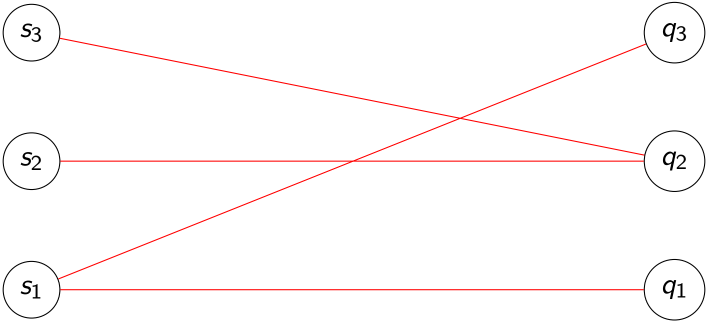
This is a special case of weighted matching in which:

- $w_{s,t} = 1$ if $(s, q) \in \mathcal{F}$
- $w_{s,t} = 0$ if $(s, q) \notin \mathcal{F}$

Example: find a maximum matching

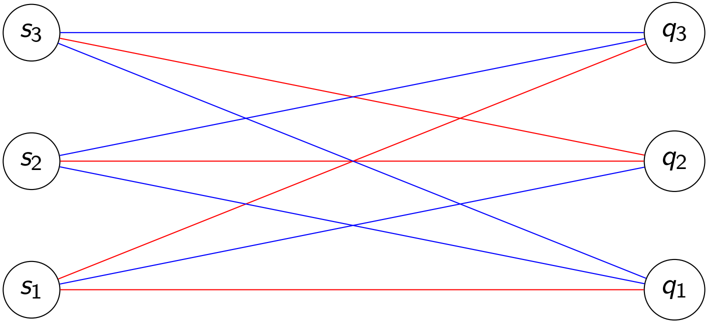


Example: find a maximum matching



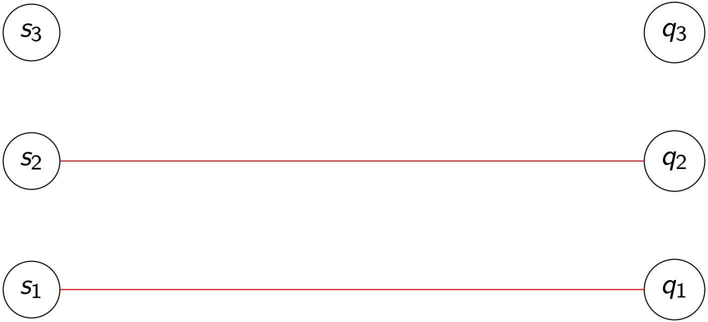
Red edges have weight 1

Example: find a maximum matching



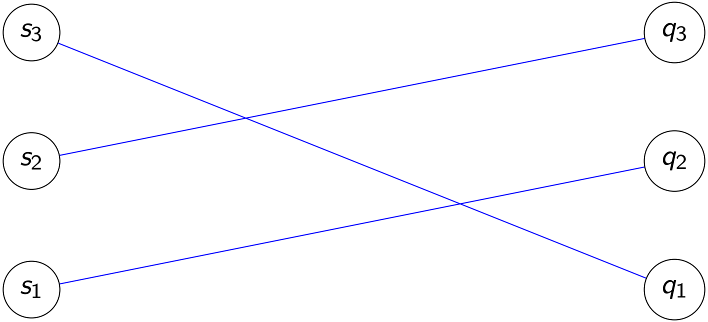
Blue edges have weight 0

Example: find a maximum matching



A maximum matching has value 2 (number of matches)

Example: find a maximum matching



A sub-optimal matching has value strictly smaller than 2 (in this case 0)

Some applications of matching

Some **applications** of matching are:

- Assign workers to tasks
- Match drivers with riders
- Match users with products
- Match patients with organs
- .....

Solving the weighted matching problem

The weighted matching problem can be solved **efficiently** (i.e., in polynomial time) by many algorithms such as:

- Hungarian algorithm
- Linear programming

Tools to solve the weighted matching problem are available in many programming languages **(more during lab)**.

The study of these optimization algorithms is out of the scope of this course. We assume to have access to an **optimization oracle** that solves the matching problem.

Online matching

Online matching problem

We consider an **online** version of the matching problem in which the weights are:

- **Unknown**, and
- **Stochastic**

At each time $t = 1, \ldots, T$:

1. The weight $w^t(s, q)$ of an **match** (s, q) is sampled from a distribution $\mathcal{D}_{s,q}$ supported on $[0, 1]$

Online matching problem

We consider an **online** version of the matching problem in which the weights are:

- **Unknown**, and
- **Stochastic**

At each time $t = 1, \dots, T$:

1. The weight $w^t(s, q)$ of an **match** (s, q) is sampled from a distribution $\mathcal{D}_{s,q}$ supported on $[0, 1]$ (**alternatively we can assume subgaussian noise**)

Online matching problem

We consider an **online** version of the matching problem in which the weights are:

- **Unknown**, and
- **Stochastic**

At each time $t = 1, \dots, T$:

1. The weight $w^t(s, q)$ of an **match** (s, q) is sampled from a distribution $\mathcal{D}_{s,q}$ supported on $[0, 1]$
2. The learner chooses an matching $\mathcal{M}$
3. The learner receives reward $\sum_{(s,q)\in\mathcal{M}} w^t(s, q)$
4. The learner observes the weight $w^t(s, q)$ of each match in the matching $\mathcal{M}$

Online matching problem

We consider an **online** version of the matching problem in which the weights are:

- **Unknown**, and
- **Stochastic**

At each time $t = 1, \dots, T$:

1. The weight $w^t(s, q)$ of an **match** (s, q) is sampled from a distribution $\mathcal{D}_{s,q}$ supported on $[0, 1]$
2. The learner chooses an matching $\mathcal{M}$
3. The learner receives reward $\sum_{(s,q)\in\mathcal{M}} w^t(s, q)$
4. The learner observes the weight $w^t(s, q)$ of each match in the matching $\mathcal{M}$ $\rightarrow$ **this is called semi-bandit feedback**

Combinatorial Bandits

Combinatorial Bandits

Online Learning provides many useful abstract models such as combinatorial bandits.

A **combinatorial bandit** is defined as follows:

- A set of arms A
- A set of superarms $S \subseteq 2^A$, i.e., each superarm is a subset of A
- each arm a has a reward $r_t(a) \sim D_a$ at each round $t \in [T]$
- the reward of a superarm $\mathbf{a}_t \in S$ chosen at time t is $\sum_{a \in \mathbf{a}_t} r_t(a)$, i.e., the sum of the rewards of the arms in $\mathbf{a}_t$
- the feedback is the reward $r_t(a)$ of each arm $a \in \mathbf{a}_t$ (**semi-bandit** feedback)

Combinatorial Bandits

Let $\mu(a)$ be the expected reward of arm a, i.e., $\mu(a) = \mathbb{E}_{r(a)\sim D_a} r(a)$.

Pseudo-Regret

The **pseudo-regret** $\mathcal{R}_T$ of an algorithm is:

$$\mathcal{R}_T = T \max_{\mathbf{a}\in S} \sum_{a\in\mathbf{a}} \mu(a) - \mathbb{E}\left[\sum_{t\in[T]} \sum_{a\in\mathbf{a_t}} r_t(a)\right],$$

where the expectation is over the randomness of the algorithm.

Goal

Design an algorithm that achieves sublinear pseudo-regret ($\lim_{T\to\infty} \frac{R_T}{T} = 0$).

Applications of combinatorial bandits

Online matching

Online matching can be modeled as a combinatorial bandit.

- The set of arms A is the set of couples $(s, q) \in \mathcal{S} \times \mathcal{Q}$
- The set of superarms S includes all the matching

Online matching

Online matching can be modeled as a combinatorial bandit.

- The set of arms A is the set of couples $(s, q) \in \mathcal{S} \times \mathcal{Q}$
- The set of superarms S includes all the matching $\rightarrow$ **a matching is a set of couples (s, q), i.e., arms**

Online matching

Online matching can be modeled as a combinatorial bandit.

- The set of arms A is the set of couples $(s, q) \in \mathcal{S} \times \mathcal{Q}$
- The set of superarms S includes all the matching $\rightarrow$ **a matching is a set of couples (s, q), i.e., arms**
- Each match (s, q) has a reward $w^t(s, q) \sim D_{s,q}$ at each round $t \in [T]$
- The reward of a superarm $\mathcal{M}_t$ chosen at time t is $\sum_{(s,q)\in\mathcal{M}_t} w^t(s, q)$
- The feedback is the reward $w^t(s, q)$ of each match $(s, q) \in \mathcal{M}_t$

Online knapsack

Online knapsack

Each arm has a cost and a superarm cannot have cumulative cost larger than a given budget.

Online knapsack

Online knapsack

Each arm has a cost and a superarm cannot have cumulative cost larger than a given budget.

Example: managing advertising campaigns

Budget B=10000$

Campaign 1 2000$	Campaign 2 3000$	Campaign 3 5000$	Campaign 4 8000$

Online knapsack

Online knapsack

Each arm has a cost and a superarm cannot have cumulative cost larger than a given budget.

Example: managing advertising campaigns

Budget B=10000$

Campaign 1 2000$

Campaign 2 3000$

Campaign 3 5000$

Campaign 4 8000$

Online knapsack

Online knapsack

Each arm has a cost and a superarm cannot have cumulative cost larger than a given budget.

Example: managing advertising campaigns

Budget B=10000$

Campaign 1 2000$

Campaign 2 3000$

Campaign 3 5000$

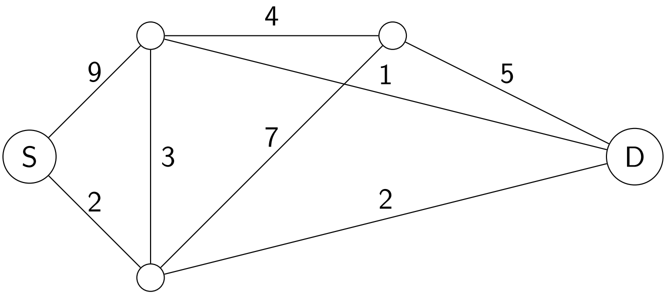
Campaign 4 8000$

Online shortest path

Online shortest path

Goal: go from a source S to a destination D as fast as possible.

- The edges are the arms
- The superarms are the paths from S to D
- The cost (opposite of the reward) is the cumulative travel time

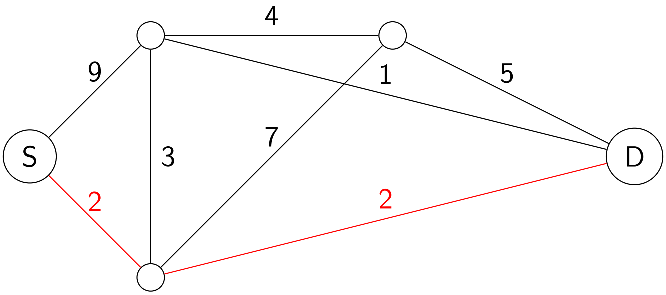


Online shortest path

Online shortest path

Goal: go from a source S to a destination D as fast as possible.

- The edges are the arms
- The superarms are the paths from S to D
- The cost (opposite of the reward) is the cumulative travel time



Algorithms for combinatorial bandits

Combinatorial-UCB

Idea: as for (non-combinatorial) stochastic bandits we can use **optimism** to incentivize **exploration**.

The general idea of Combinatorial-UCB is to:

- Define an Upper Confidence Bound (UCB) on the expected mean of each arm
- At each round, play the superarm with the higher cumulative UCB $\rightarrow$ **here the algorithm is optimistic about the mean incentivizing exploration)**
- The UCB of the played arms is updated

Upper Confidence Bound

For each arm $a \in A$, we build a confidence interval around its empirical mean.

- $N_t(a)$ is the number of times we pick arm a in the first t rounds:

$$N_t(a) := \sum_{t'=1}^{t} \mathbb{I}[a \in \mathbf{a}_{t'}].$$

- $\mu_t(a)$ is the empirical mean of arm a:

$$\mu_t(a) = \frac{1}{N_{t-1}(a)} \sum_{t'=1}^{t-1} r_{t'}(a)\mathbb{I}[a \in \mathbf{a}_{t'}]$$

Upper Confidence Bound

For each arm $a \in A$, we build a confidence interval around its empirical mean.

At each time $t \in [T]$, we define an UCB of the arm average reward $\mu_t(a)$:

$$UCB_t(a) = \underbrace{\mu_t(a)}_{exploitation\ \ term} + \underbrace{\sqrt{\frac{2\log(T)}{N_{t-1}(a)}}}_{exploration\ \ term}$$

- The term $\mu_t(a)$ incentivizes to play arms with large empirical mean
- The term $\sqrt{\frac{2\log(T)}{N_{t-1}(a)}}$ incentivizes to play arms with low $N_{t-1}(a) \rightarrow$ played a small number of times

Upper Confidence Bound

For each arm $a \in A$, we build a confidence interval around its empirical mean.

At each time $t \in [T]$, we define an UCB of the arm average reward $\mu_t(a)$:

$$UCB_t(a) = \underbrace{\mu_t(a)}_{exploitation\ \ term} + \underbrace{\sqrt{\frac{2\log(T)}{N_{t-1}(a)}}}_{exploration\ \ term}$$

- The term $\mu_t(a)$ incentivizes to play arms with large empirical mean
- The term $\sqrt{\frac{2\log(T)}{N_{t-1}(a)}}$ incentivizes to play arms with low $N_{t-1}(a) \rightarrow$ played a small number of times

⚠ We can also take the smaller exploration term $\sqrt{\frac{2\log(t)}{N_{t-1}(a)}}$.

Combinatorial-UCB

Algorithm: Combinatorial-UCB

set of arms A, set of superarms S, number of rounds T;
for $t = 1, \ldots, T$ **do**
  for $a \in A$ **do**
    $\mu_t(a) \leftarrow \frac{1}{N_{t-1}(a)} \sum_{t'=1}^{t-1} r_{t'}(a)\mathbb{I}[a_{t'} = a]$;
    $UCB_t(a) \leftarrow \mu_t(a) + \sqrt{\frac{2\log(T)}{N_{t-1}(a)}}$;
  play superarm arm $\mathbf{a}_t \in \arg\max_{\mathbf{a} \in S} \sum_{a \in \mathbf{a}} UCB_t(a)$;

Combinatorial-UCB

Algorithm: Combinatorial-UCB

1 set of arms A, set of superarms S, number of rounds T;
2 **for** $t = 1, \dots, T$ **do**
3   **for** $a \in A$ **do**
4         $\mu_t(a) \leftarrow \frac{1}{N_{t-1}(a)} \sum_{t'=1}^{t-1} r_{t'}(a)\mathbb{I}[a_{t'} = a]$;
5         $UCB_t(a) \leftarrow \mu_t(a) + \sqrt{\frac{2\log(T)}{N_{t-1}(a)}}$;
6   play superarm arm $\mathbf{a}_t \in \arg\max_{\mathbf{a} \in S} \sum_{a \in \mathbf{a}} UCB_t(a)$;

At Line 6, we use an optimization oracle that solves the optimization problem $\rightarrow$ **We can use the Hungarian algorithm in a matching problem.**

Regret guarantees

Similarly to UCB1, combinatorial-UCB provides $\log(T)$ instance-dependent regret.

Theorem

combinatorial-UCB achieves pseudo-regret:

$$\mathcal{R}_T \leq C \log(T),$$

where C is independent from T but depends on the instance.

- Similarly to UCB1, the constant C can be arbitrarily large in some problem instances

Combinatorial-TS

Idea: Use a Bayesian approach.

We can generalize **Thompson sampling** to combinatorial bandits:

- Focus on Bernulli reward distributions (i.e., with support $\{0, 1\}$) for each arm
- Use a Beta distribution as prior distribution

The rewards are Bernulli in many applications. For example, in the online maximum matching problem.

Update the distribution

- We start with prior $Beta(1, 1) \rightarrow$ uniform distibution over $[0, 1]$ for each arm
- When we observe a new sample from arm a we increase α if we observe $r_t(a) = 1$ or β if we observe $r_t(a) = 0$

$$(\alpha_a, \beta_a) \leftarrow (\alpha, \beta) + (r_t(a), 1 - r_t(a))$$

- Differently from non-combinatorial bandits we update the distribution of every arm in the superarm $\mathbf{a}_t$
- The Beta distribution is a probability distribution over the expected value of the arm

Choose which arm to play

- For each arm a, sample $\theta_a \sim Beta(\alpha_a, \beta_a)$
- Play the superarm with the largest cumulative sampled mean:

$$\mathbf{a}_t \in \arg\max_{\mathbf{a} \in S} \sum_{a \in \mathbf{a}} \theta_a$$

Combinatorial-TS

Algorithm: Combinatorial-TS

set of arms A, number of rounds T;
for $a \in A$ **do**
  $\alpha_a = \beta_a = 1$;
for $t = 1, \ldots, T$ **do**
  for $a \in A$ **do**
    $\theta_a \sim Beta(\alpha_a, \beta_a)$;
  play superarm $\mathbf{a}_t \in \arg\max_{\mathbf{a} \in S} \sum_{a \in \mathbf{a}} \theta_a$;
  for $a \in \mathbf{a_t}$ **do**
    update$(\alpha_a, \beta_a) \leftarrow (\alpha_a, \beta_a) + (r_t(a), 1 - r_t(a))$;

Combinatorial-TS

Algorithm: Combinatorial-TS

1 set of arms A, number of rounds T;
2 **for** $a \in A$ **do**
3 $\quad \alpha_a = \beta_a = 1$;
4 **for** $t = 1, \dots, T$ **do**
5 $\quad$ **for** $a \in A$ **do**
6 $\quad\quad \theta_a \sim Beta(\alpha_a, \beta_a)$;
7 $\quad$ play superarm $\mathbf{a}_t \in \arg\max_{\mathbf{a} \in S} \sum_{a \in \mathbf{a}} \theta_a$;
8 $\quad$ **for** $a \in \mathbf{a_t}$ **do**
9 $\quad\quad \text{update}(\alpha_a, \beta_a) \leftarrow (\alpha_a, \beta_a) + (r_t(a), 1 - r_t(a))$;

At Line 6, we use an optimization oracle that solves the optimization problem $\rightarrow$ **We can use the Hungarian algorithm in a matching problem.**

Regret guarantees

Similarly to combinatorial-UCB, combinatorial-TS provides $\log(T)$ instance-dependent regret.

Theorem

Combinatorial-TS achieves pseudo-regret:

$$\mathcal{R}_T \leq C \log(T),$$

where C is independent from T but depends on the instance.

- Similarly to combinatorial-UCB, the constants C can be arbitrarily large in some problem instances
- Usually combinatorial-TS provides better empirical performances than combinatorial-UCB