PRICING in E-COMMERCE

E-Commerce Data

Electronic commerce is the main economic scenario where the dimension of pricing can lead to significant increase in revenue

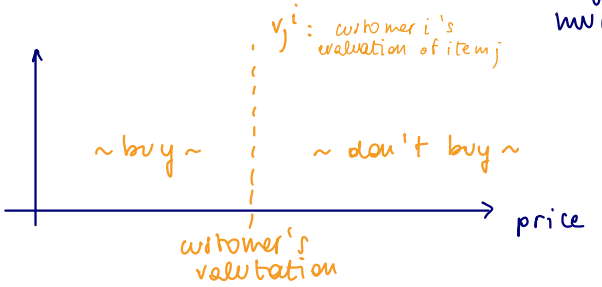
→ in last 5 years the sales due to electronic-commerce increased a lot

ECONOMIC PRELIMINARIES

Basic scenario

A single seller has to sell infinite units of a single good to multiple customers

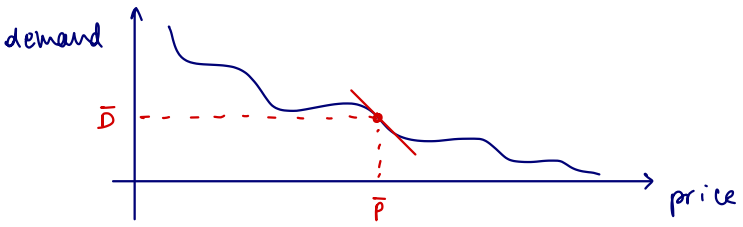
- SELLER MODEL: • the seller has a cost over the good
  • all units are sold at same price
- CUSTOMER MODEL: • every customer has a valuation over the good (says how much he wants to spend for a good)



the customer will accept any price smaller than his valuation

- demand curve

If we have many customers we construct the demand curve → different evaluation

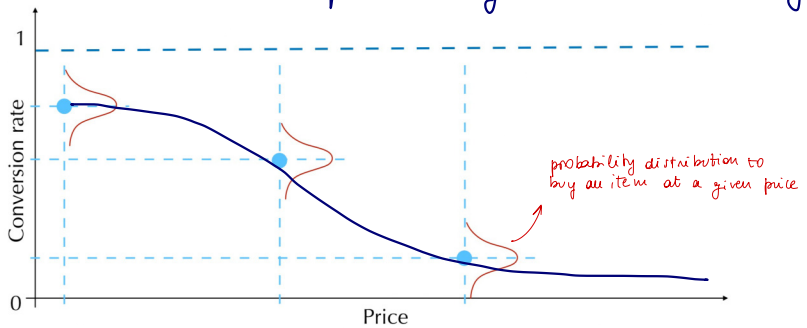


demand = number of customers that would buy an item at a given price

demand elasticity < 1 : the seller sells same number of items with an higher price

demand elasticity: $\frac{\bar{p}}{\bar{D}} \frac{dD}{dp}(\bar{p})$ → it corresponds to the normalization of the derivative of the demand wrt the price (look at the speed of change)

⇒ actually the demand curve is unknown: we rely on conversion rate
↳ which is the probability to sell an item given a price?



from conversion rate we build the average demand curve

Pricing

~How to choose the price of a good with economical tools~

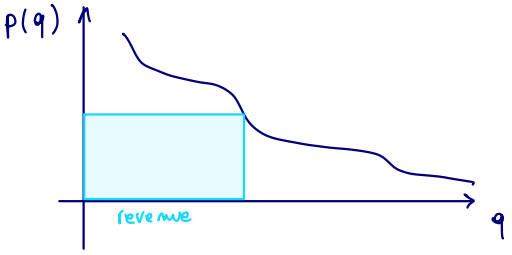
- Scenarios
  1. Monopoly → only one seller in the market
  2. Oligopoly → few sellers in the market
  3. Competitive market → many sellers in the market

- Definitions

p : price		q : demand
$q(p)$: demand at price p		$p(q)$: price of a demand
$c(q(p))$: cost of $q(p)$ items for the seller	(*)	$c(q)$: cost of q items for the seller
$p \cdot q(p)$: revenue at price p		$q \cdot p(q)$: revenue at demand q
$p \cdot q(p) - c(q(p))$: profit at price p		$q \cdot p(q) - c(q)$: profit at demand q

(*) the cost of every item is the same.
If the number of sold items is different, also the cost of every item is different

$\Rightarrow$ **the seller wants to maximize the profit**



the price to apply depends on the quantity we want to sell

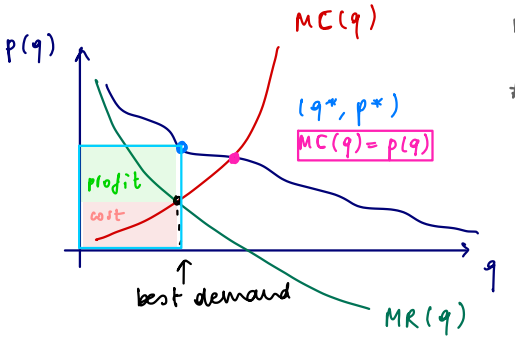
- profit maximization

we need to find $\bar{q} = \operatorname{argmax}_q \{ p(q) \cdot q - c(q) \}$

$$\frac{d}{dq}\left(p(q)\cdot q - c(q)\right) = 0 \quad \rightarrow \quad \frac{dp(q)}{dq}\cdot q + p(q) = \frac{dc(q)}{dq}$$

$\frac{dp(q)}{dq} \cdot q + p(q)$: marginal revenue $MR(q)$: revenue provided by selling the last unit of good when # items $= q$

$\frac{dc(q)}{dq}$: marginal cost $MC(q)$ cost due to selling the last unit of good when # items $= q$

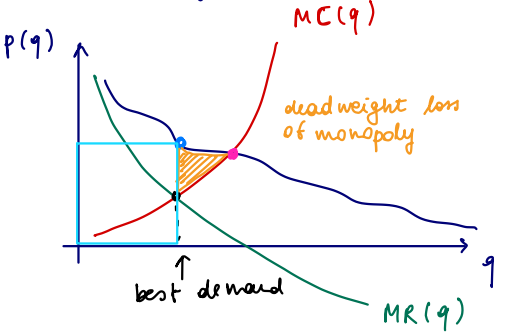


best price in the demand curve → the seller sells q items with an higher price

in order to sell more items (avoid negative profit)
→ OPTIMAL SOCIAL PRICE

→ deadweight loss of monopoly : inefficiency of the market due to the seller

$\Rightarrow$ the seller is selfish and he doesn't allow the maket to be efficient

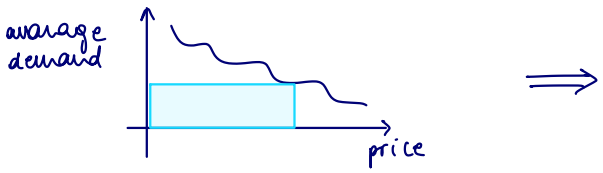


the seller increases his profit wrt the choice of the best demand → make happier more customers

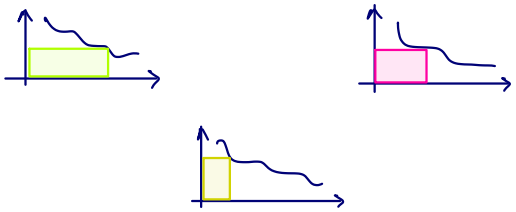
Price discrimination

- simple price inefficiency
  - Deadweight loss of monopoly is due to the adoption of a simple pricing strategy
    every item is sold at the same price
  - If the seller can personalize the price for each customer, the social efficiency of the market can be improved

⇒ we build different demand curves according to different customer classes



⇒

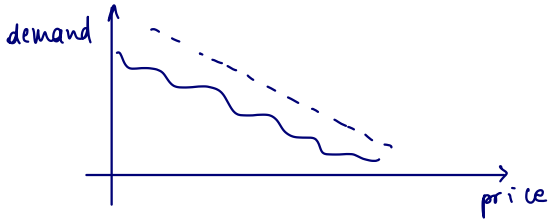


AGGREGATE DEMAND:
If we don't have information on customers we need to set the same price for each costumer

DISAGGREGATE MODELS: If the seller knows the class of the customers, we can separate the problem and apply different prices for every class
⇒ price discrimination increases the profit

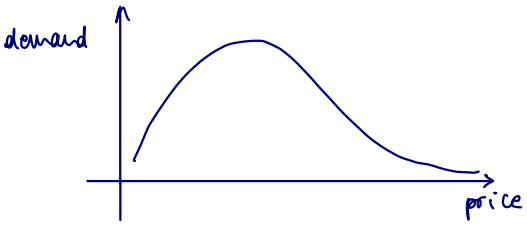
single product demand curves

- Negative slope demand curve



the demand monotonically decreases as the price increases
→ It is referred to as the "law of demand": people will buy more items as its price falls

- Non negative slope demand curve



the demand does not monotonically decreases as the price increases
→ goods for which the quantity demanded increases as the price increases

Interdependent product demand curves

~ Correlated Goods ~

A seller has to sell an infinite number of A and B to multiple customers
→ there is dependency between A and B

- Strategic dependency

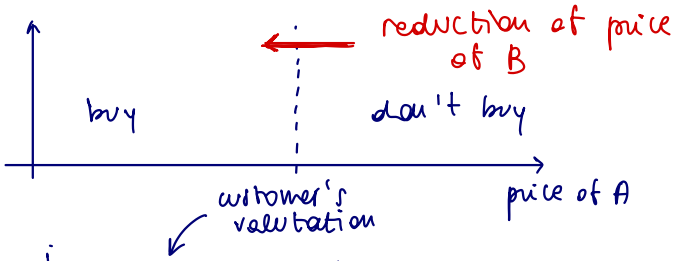
① Strategic substitutes
idea: good A can substitute good B (and viceversa)
effect: if the price of A reduces then the costumers' willingness to pay good B reduces

A similar to B
A ↑ ⇔ B ↑

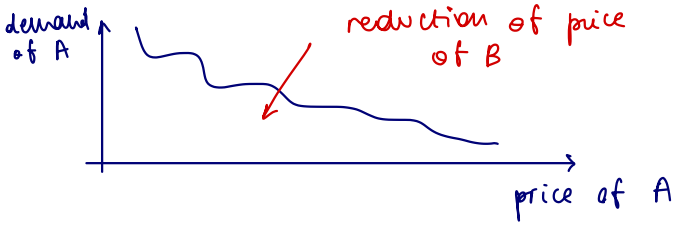
② Strategic Complements
idea: good A can complement good B (and viceversa)
effect: if the price of A reduces, then the costumers' willingness to pay good B increases

A and B used together
A ↓ ⇔ B ↑
{ A ↓ : sell more items
B ↑ : sell more items }

- Customer model with substitutes

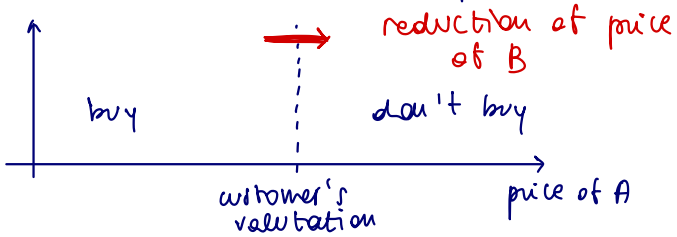


- demand curve with substitutes

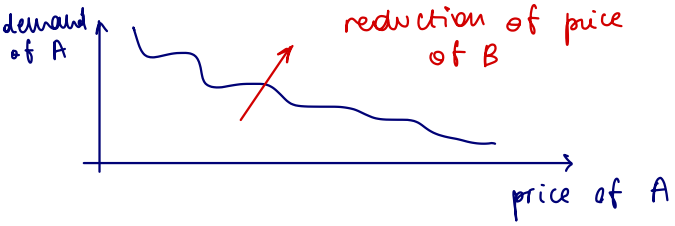


$v_i^j(p^k)$: customer i's evaluation of item j depending on price of good k

- Costumer model with complements



- demand curve with complements

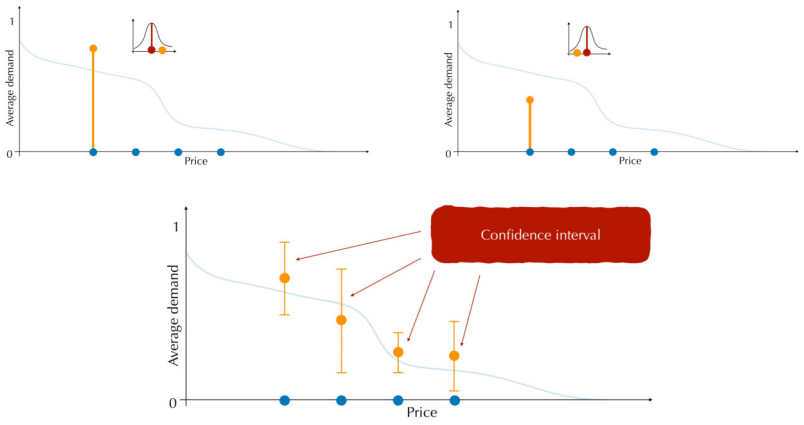


- pricing interdependent goods
  → two or more goods that are interpreTendent (due to strategic complements or strategic substitutes) must be priced together in order to maximize the profit

LEARNING THE DEMAND CURVE

The average expected demand curve in general is unknown
→ we have to estimate it

- Sampling the demand curve
  1. we fix some values of price and we estimate the demand
  2. observe the demand drawn from a probability distribution (in general is gaussian)
  3. collect samples and calculate the empirical mean (average)
  4. Estimate the confidence intervals



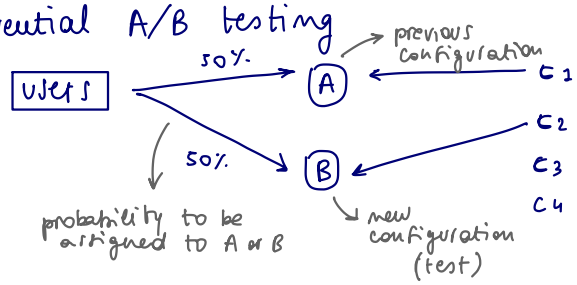
- Setting up
  given some candidates (prices) we associate an unknown expected reward (profit = price · demand)

Hp 1. rewards are stationary : the probability distribution of rewards is stationary and the demand curve doesn't change
2. we can use a single candidate per round
3. The feedback is not delayed : given a price we can't observe the samples
4. The number of rounds is bounded : finite time horizon

INTRODUCTION TO A/B/n TESTING

☑ Non-technical introduction to A/B/n testing
~ How to learn the best prices ~

- Sequential A/B testing

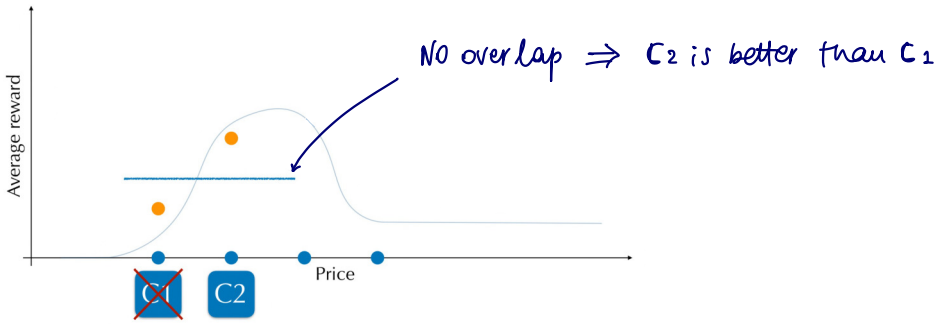


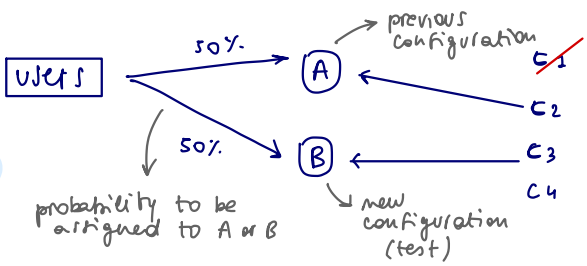
1° step

we want to understand if B is better than A

- Several definitions of reward
  1. profit maximization
     - demand curve has to be estimated (thanks to the fact that for every price it behaves as a truncated Gaussian distribution) and it has to be multiplied by the marginal profit
     - conversion rate has to be estimated (thanks to the fact that for every price it behaves as a binomial distribution (buy or not)) and it has to be multiplied by the marginal profit
  2. volumes maximization (market invasion) → fight with another company
     - demand has to be estimated
     - conversion rate has to be estimated

⇒ we collect many samples in order to have a small confidence interval.



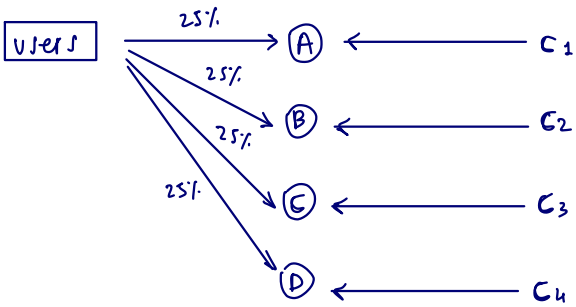


2° step

because we know c_2 is better than c_1

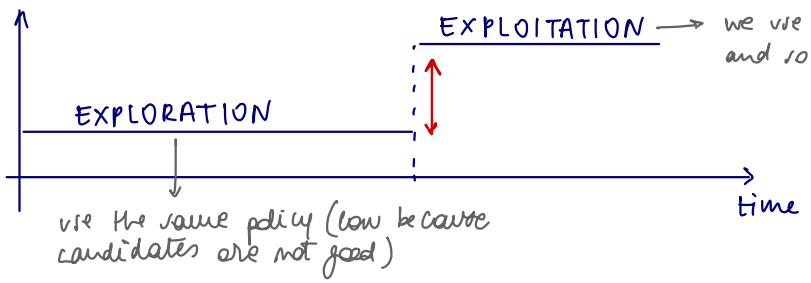
this method is very slow since we compare all couples

- Sequential A/B/n testing
  we decide to evaluate all together



A candidate is chosen from a uniform probability because no prior information and we calculate the expected reward. We do many times for all candidates at the end of the experiment we select the winner

- pure exploration → in the time horizon we consider all candidates with same weight
- pure exploitation → we we only the winner to find the expected reward



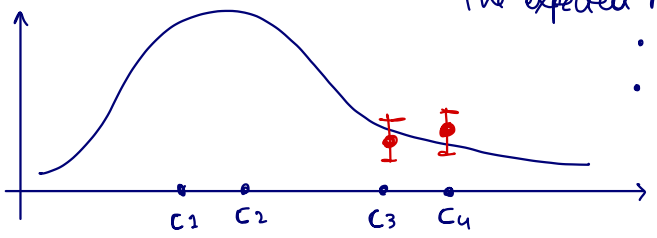
VERY SIGNIFICANT LOSS
offline learning → before collect data, then perform

→ A/B/n Testing depends on the length of time horizon
- LOW: lose a bit of expected reward
- LONG: lose the ratio half expected reward

⇒ How long should an A/B/n test be ?
- confidence to be guarantee
- reward to be collected

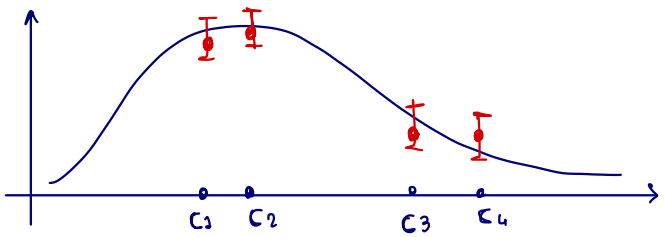
- A/B/n weaknesses
1. assumption of stationary process → probability distribution doesn't change during time, but actually it can be good in future
2. long time to identify the optimal candidate → time horizon is too short wrt the best solution
3. Discarding a potentially optimal candidate → if the required confidence needs a larger number of samples wrt time horizon ~ high confidence

Example (A/B testing)



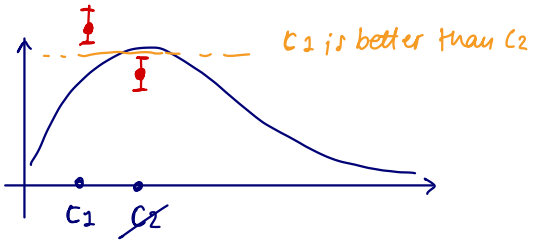
The expected reward of these two suboptimal solutions is very close
- need a larger number of samples
- lose time to find the best between c_3 and c_4 but maybe c_1 and c_2 are better

Example (A/B/n testing)



This time we test all candidates together
- long time to discriminate c_1 vs c_2 and c_3 vs c_4
- we can exclude immediately c_3 and c_4 but we have to wait the end of the episode

→ in order to decide the optimal candidate we rely on small confidence intervals but it may happen that we exclude a better solution
→ once a candidate is discarded, it cannot be reconsidered in the future
⇒ INACCURATE

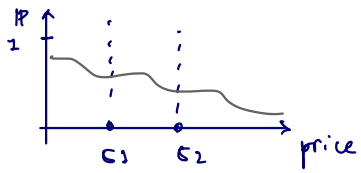


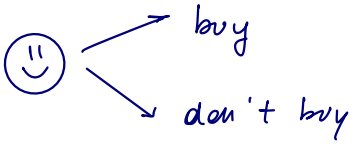
A/B/n TESTING TECHNICALITIES

Hypothesis testing

① Defining the problem

The data we collect is composed by customers who decide to buy or not an item given a price - we can model this decision with a Bernoulli distribution



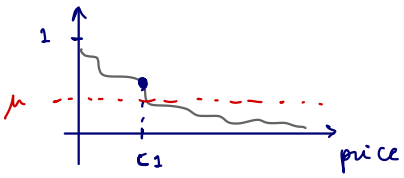


② Formulating an hypothesis to verify

1 candidate

$H_0: \mu_{c_1} = \mu$ vs $H_1: \mu_{c_1} \neq \mu$

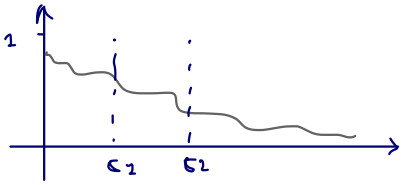
$H_1: \mu_{c_1} > \mu$



2 candidates

$H_0: \mu_{c_1} - \mu_{c_2} = \mu$ vs $H_1: \mu_{c_1} - \mu_{c_2} \neq \mu$

$H_1: \mu_{c_1} - \mu_{c_2} > \mu$



actually $\mu = 0$ in order to compare c_1 vs c_2

③ Selecting a test statistic

→ The test statistics defines the algorithm to use and verify the hypothesis

$$Z = \frac{\bar{X} - \mu_0}{\sqrt{\frac{\mu_0(1-\mu_0)}{n}}}$$ → test statistic random variable

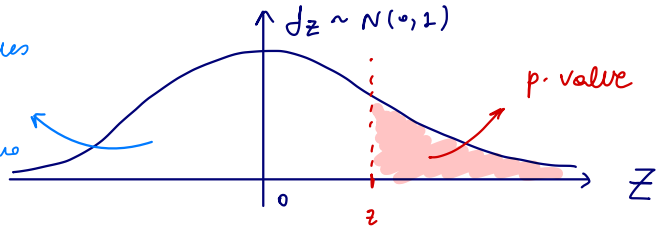
$\bar{X}$ sample mean rv

data ⟹

$$z = \frac{\bar{x} - \mu_0}{\sqrt{\frac{\mu_0(1-\mu_0)}{n}}}$$ → test statistic

$\bar{x}$ empirical mean

1 - p-value: probability that samples are drawn from a distribution whose mean is bigger than μ_0



$$Z = \frac{\bar{X_1} - \bar{X_2}}{\sqrt{\bar{Y}(1-\bar{Y})\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

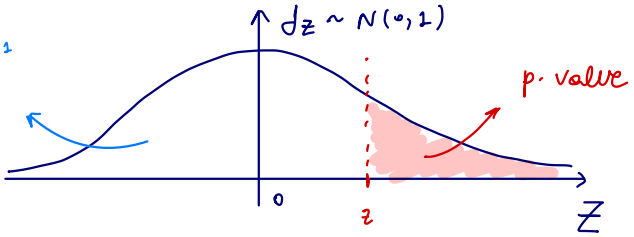
$$z = \frac{\bar{x_1} - \bar{x_2}}{\sqrt{\bar{y}(1-\bar{y})\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

data ⟹

$$Y = \frac{n_1 \bar{X_1} + n_2 \bar{X_2}}{n_1 + n_2}$$

$$y = \frac{n_1 \bar{x_1} + n_2 \bar{x_2}}{n_1 + n_2}$$

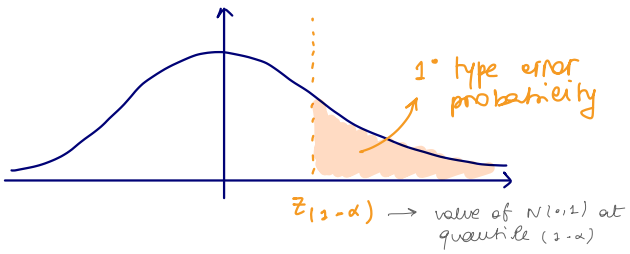
1 - p-value: probability that samples x_1 are drawn from a distribution whose mean is bigger than the mean of the distribution from which samples x_2 are drawn



④ Selecting the accuracy of the test statistic (before collecting data)

$\alpha = \mathbb{P}$(1° type error)

↑ reject H_0 when it's true



⑤ Selecting random samples

Fix the number of samples, but how many ?

	H_0 is true	H_1 is true
Not rejecting H_0	$1-\alpha$	β
Rejecting H_0	α	$1-\beta$

we want to collect a small number of samples but at the same time it has to be enough to find the winner.

⇒ The minimum number of samples to collect for A\B testing is :

$H_0 : \mu_{C1} = 0,7$

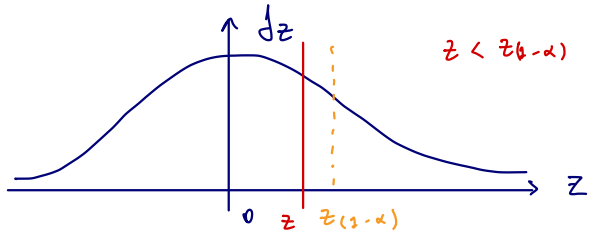
$H_1 : \mu_{C1} \geq 0,7 + \delta$

$$n = \frac{(z_{(1-\alpha)} + z_{(\beta)})^2 \sigma^2}{\delta^2}$$

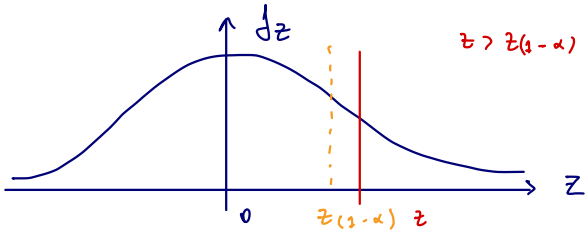
$\sigma \rightarrow$ standard deviation

$\delta \rightarrow$ minimal variation in the process

⑥ Calculating the test statistic on the given samples and Taking the decision



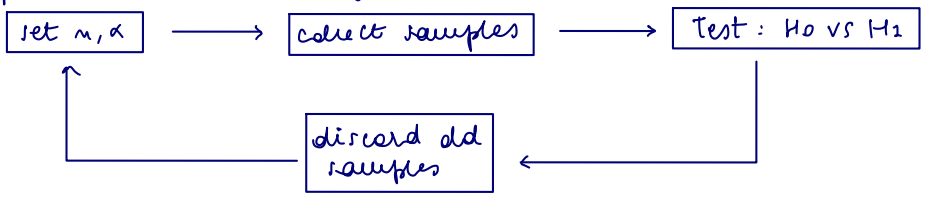
ACCEPT H_0



REJECT H_0

~ p-value < α ~

⇒ we repeat the test many times

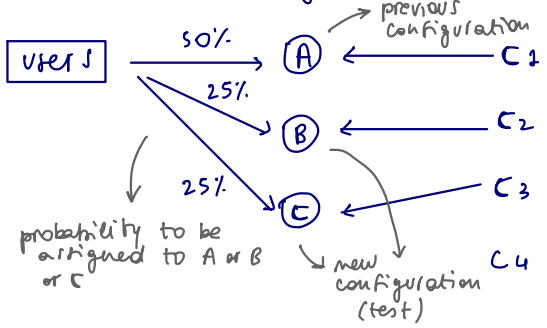


Rem A/B testing is not an iteractive tool

- we have to collect all data before taking the decision
- Once we decide, the old samples are discarded and not used anymore

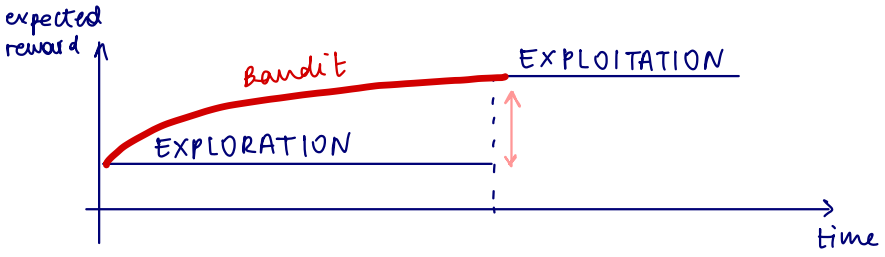
INTRODUCTION TO BANDIT ALGORITHMS

- from A/B/n testing to Bandit Algorithm



Bandit Algorithms don't use an uniform distribution to assign users to A, B, C

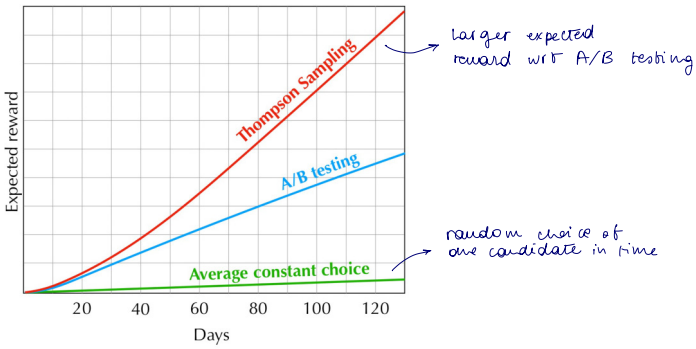
→ the probability changes wrt observations we collect



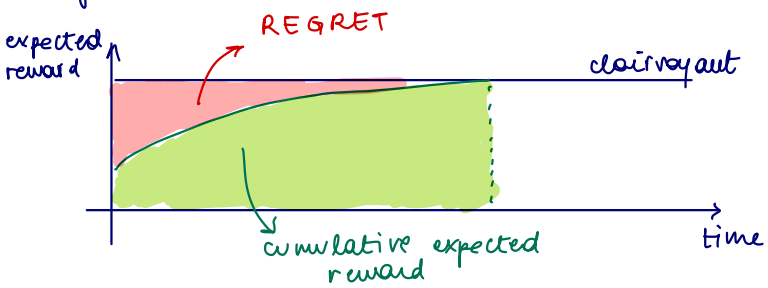
we perform better during the exploitation phase because we tune the probability wrt observations we get

online learning → use collected data while performing

- Scenarios: Pricing for E-commerce



- Regret Minimization



regret: what we're losing due to the fact we're learning (loss of reward)

CLAIRVOYANT ALGORITHM:
a priori the expected reward of all candidates is known and at each step the best candidate is chosen
~ perfect algorithm, not realizable ~

→ since a zero-regret algorithm doesn't exist (except clairvoyant), we want to provide a lower and upper bound to the regret
⇒ BANDIT ALGORITHMS

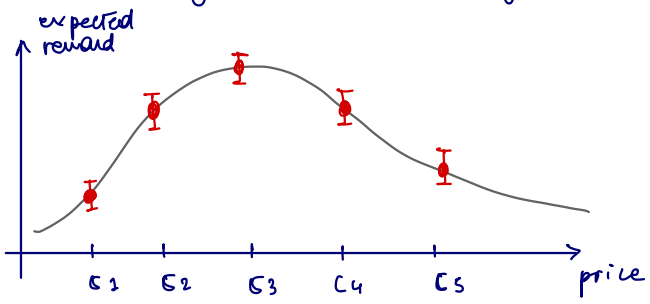
ps: the minimization of the regret corresponds to the maximization of the reward
regret ↓ ⇒ reward ↑

- Bandit Algorithm

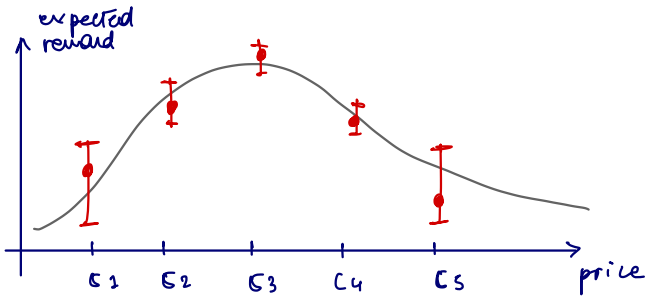
sequence of candidates chosen in the previous round + feedbacks → BANDIT ALGORITHM → candidate to use at current round

- Offline vs Online learning
  - → A/B/n testing is an offline learning → collect all data and then decide which candidate to play
  - → Bandit Algorithm is an online learning → decide at every step using the previous data

- A/B/n Testing vs Bandit Algorithms



A/B/n testing does an accurate estimation of all candidates



Bandit algorithms do an accurate estimation only for good candidates

BANDIT TECHNICALITIES

UCB1

frequentist and deterministic approach

- every arm is associated with an upper confidence bound
  → optimistic estimation of the reward provided by an arm
- the arm with the highest upper confidence bound is chosen at each step
- update of the rewards → update of the upper confidence bounds

hp : 1. the rv of every arm is a bernoulli with unknown mean
2. every arm has a different mean
3. The model captures the scenario in which one needs to learn a conversion rate
4. reward = conversion rate · net margin (UNK)

algorithm

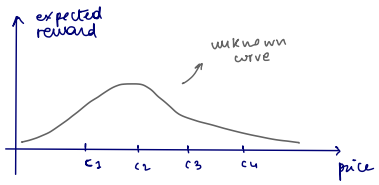
1° step : play once every arm $a \in A$

2° step : at every time t play arm a_t :

$$a_t = \underset{a \in A}{\operatorname{argmax}} \left\{ \bar{x}_a + \underbrace{\sqrt{\frac{2 \log t}{n_a(t-1)}}}_{\text{UCB (uncertainty)}} \right\}$$

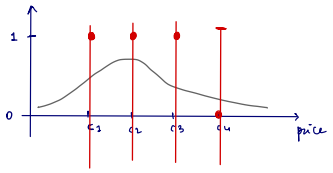
a^* : optimal arm
a_t : arm a chosen at time t
X_a : rv of reward of arm a
x_a : realization of X_a
$\bar{x}_a$: empirical mean of x_a
$n_a(t)$: number of times we chose arm a from 0 to t

Example

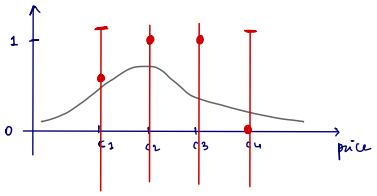


1° step : we pull $C_1 \rightarrow x_{C_1} = 1$ we pull $C_2 \rightarrow x_{C_2} = 1$
we pull $C_3 \rightarrow x_{C_3} = 1$ we pull $C_4 \rightarrow x_{C_4} = 0$

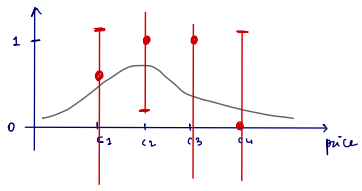
2° step : compute the empirical mean $x_a = \bar{x}_a$
and compute UCB
→ choose next $a_t = \operatorname{argmax} \left\{ \bar{x}_a + \sqrt{\frac{2 \log t}{n_a(t-1)}} \right\}$



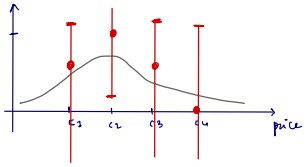
② we can choose among C_1, C_2, C_3
we pull $C_1 \rightarrow x_{C_1} = 0$
then we choose next at



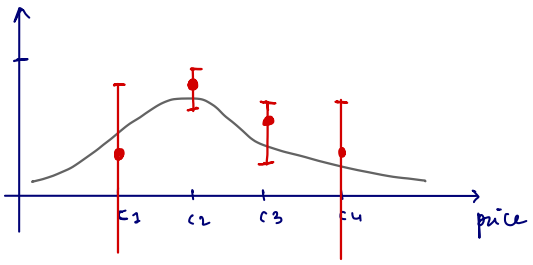
③ we can chose among C_2, C_3
we pull $C_2 \rightarrow x_{C_2} = 1$,
then we chose next at $= C_3$



④ we pull $C_3 \rightarrow x_{C_3} = 0$
then we choose C_2



⇒ after many samples



we have different confidence intervals

Regret

Def (regret) $R_T(U) = T\mu_{a^*} - \sum_{t=1}^{T} \mu_{a_t}$

- $R_T(U)$: expected regret after T steps
- $T\mu_{a^*}$: expected reward of clairvoyant
- $\sum_{t=1}^{T} \mu_{a_t}$: expected regret of algorithm U

Def (UCB1 Regret)

$$R_T(UCB1) \leq \sum_{a:\mu_a<\mu_{a^*}} \frac{4 \log T}{\mu_{a^*} - \mu_a} + 8(\mu_{a^*} - \mu_a)$$

- it increases logaritmically as T increases
- it increases linearly as |A| increases

Thompson Sampling

Bayesian and randomized approach

- For every arm, we have a prior on its expected value
  → if arms' rewards are Bernoulli then priors are Beta
- For every arm we drawn a sample according to Beta distribution
- the arm with the best sample is chosen at every step
- we update the Beta distribution according to the observation

{ observe 1 : Beta(α, β) → Beta$(\alpha+1, \beta)$
{ observe 0 : Beta(α, β) → Beta$(\alpha, \beta+1)$

algorithm

1° step : start with $\alpha = 1, \beta = 1$

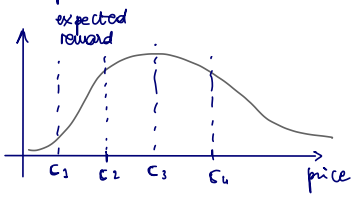
2° step : at every time t, for any arm
$\tilde{\vartheta}_a$ = sample $(\mathbb{P}(\mu_a = \vartheta_a))$ from Beta(α, β)

3° step : at every time t play arm a_t such that
$a_t = \underset{a \in A}{\text{argmax}} \{\tilde{\vartheta}_a\}$

4° step : update Beta distribution of arm a_t
$(\alpha_{a_t}, \beta_{a_t}) = (\alpha_{a_t}, \beta_{a_t}) + (x_{a,t}, 1 - x_{a,t})$

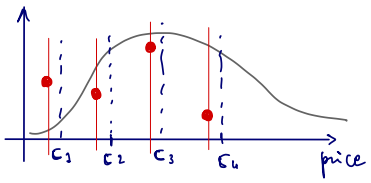
$x_{a,t}$ realization of X_a at time t
$\mathbb{P}(\mu_a = \vartheta_a)$: prior on $\mathbb{E}[X_a]$
$\mathbb{P}(\mu_a = \vartheta_a)$ = Beta(α_a, β_a)

Example

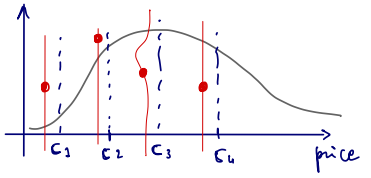


at the beginning $\alpha = 1$, $\beta = 1$.
No prior information : Beta$(1,1)$ = Uniform !

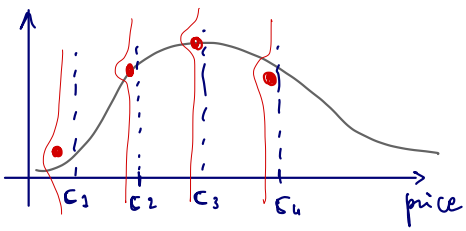
① sample all candidates from Beta
then choose the arm with the highest reward
→ c_3 has the highest expected reward ⇒ $x_{c_3,t} = 1$
→ update c_3 beta distributions accordingly



② sample all candidates from Beta and choose the best
→ c_2 has the highest expected reward ⇒ $x_{c_2,t} = 0$
→ update c_2 beta distribution accordingly



⇒ after many samples



we can see that almost the probability is close to the expected value wrt the most promising candidates (because they were pulled more)

Def (TS Regret)

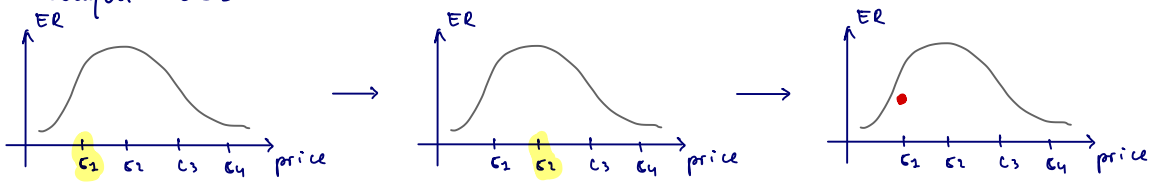
$$R_T(TS) \leq (1+\varepsilon) \sum_{a:\mu_a<\mu_{a^*}} \frac{(\mu_{a^*} - \mu_a)(\log T + \log(\log T))}{K\,\mathcal{D}(\mu_{a^*}, \mu_a)} + C(\varepsilon, \mu_{a_1} \dots \mu_{|A|})$$

- it increases logaritmically as T increases
- it increases linearly as |A| increases

DELAYED FEEDBACK

Bandit algorithm are very fast but the realization needed for the decisions are observed after some time

- delayed - UCB1



we pull c_1, then we pull c_2 and we observe the realization of c_1

algorithm

1° step : play once every arm $a \in A$

2° step : at every time t play arm a_t such that

$$a_t = \underset{a:\, \hat{n}(t-1)>0}{\operatorname{argmax}} \left\{ \bar{x}_a + \frac{2 \log T}{\hat{n}_a(t-1)} \right\}$$

$\hat{n}_a(t)$: number of feedbacks for arm a at time t

$\bar{G}_{a,T} = \max_{t \le T} G_{a,t}$ max number of missing feedbacks for an arm a $[0,t]$

regret

$$R_T(UCB1) \le \sum_{a:\, \mu_a < \mu_{a^*}} \left[\frac{8 \log T}{\mu_{a^*} - \mu_a} + 3{,}5 (\mu_{a^*} - \mu_a) \right] + \sum_a (\mu_{a^*} - \mu_a) \mathbb{E}[\bar{G}_{a,T}]$$

- it increases logarithmically as T increases
- it increases linearly as $|A|$ increases

⇒ it increases linearly as the delay increases

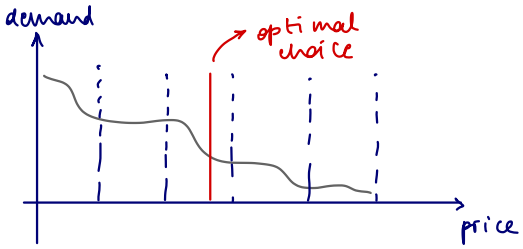
- delayed - TS

It has the same idea of UCB1, but it's not discussed in literature.
We expect that the regret increases linearly as the delay increases

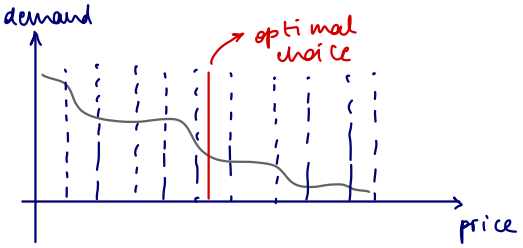
HOW MANY CANDIDATES?

We need different candidates in order to do a discretization of the price space

5 candidates

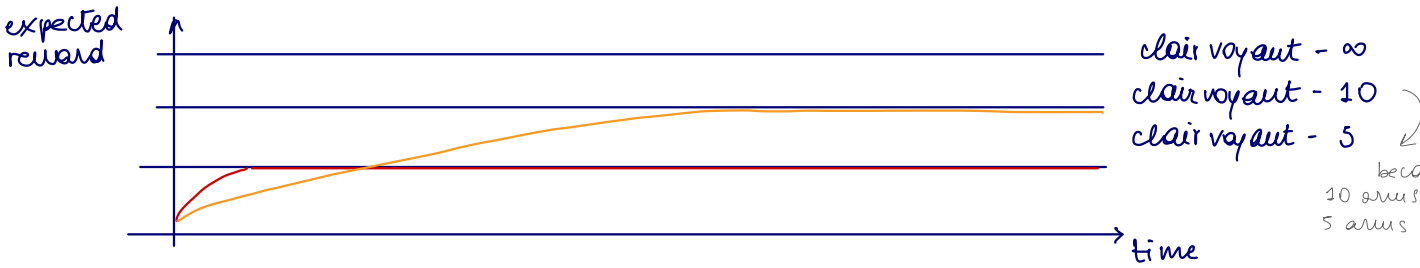


10 candidates



- Expected reward

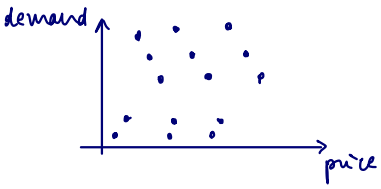
When we evaluate the regret we have to evaluate the expected reward archieved by the clairvoyant algorithm (the infinite case and the finite case)



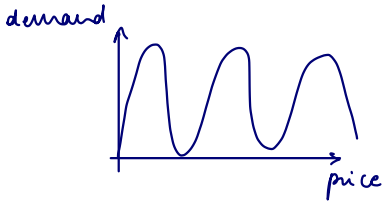
The scenario with more arms reaches an higher expected reward after a long time because we need to collect more data

⇒ if we can collect few data we need a small number of arms
if we can collect many samples we need an higher number of arms
⇒ The optimal number of candidates depends on the length of the horizon over which we want to maximize the expected reward

- Discretization
In finite time we can only discretize the space of arms. So we need some assumptions to bound the cumulative regret
→ the reward function needs to have some regularities



exclude discontinuous functions



bound first derivative

→ when the second derivative of the reward is < 0, then the regret is minimized.
The number of candidates should be:

start from this number of arms $k = \lceil (T \log T)^{1/4} \rceil$ ⟶ $R_T(UCB1) \leq \sqrt{T \log T}$

NON STATIONARY ENVIRONMENTS

Now we want to consider the change of rewards' probability distribution during time.
→ the mean expected value changes during time

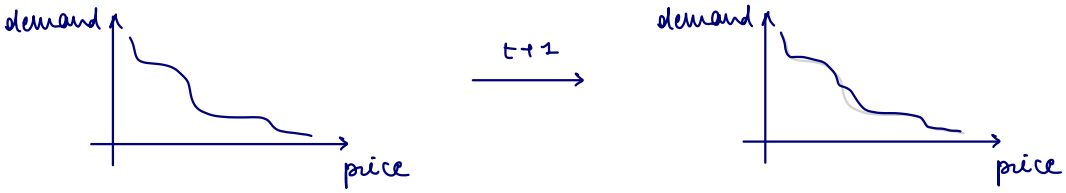
⇒ without any information about the change of the expected value,
a non-stationary environment is equivalent to an adversarial environment
↳ non stationary environments provide better guarantees in some situations
(zero-sum game)

- Non-stationary environments

① Abrupt changes ~ a new product enters the market
- time is divided in phases
- in every phase the reward functions are constant
(the change is between the end of a phase and the beginning of a new one)

the length of phases may change and it's described by a breakpoint.
phases can also be periodic

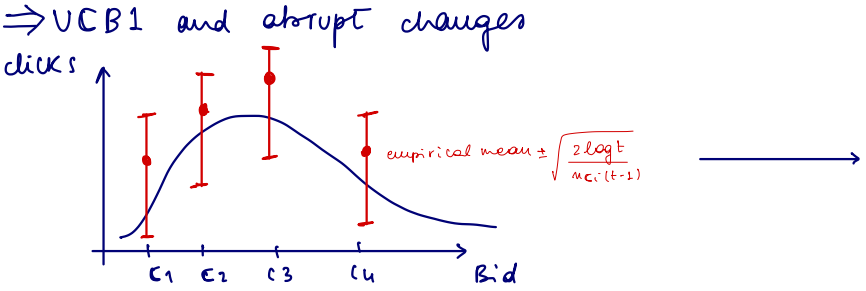
② Smooth change ~ a product looses attractiveness during time
- Reward functions continuously change during time

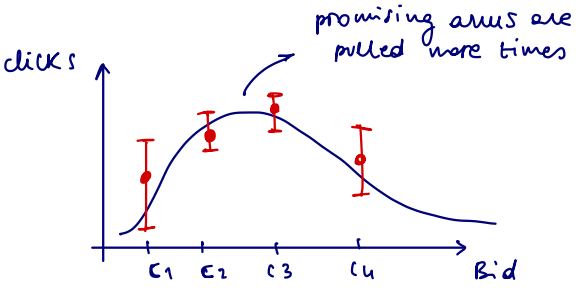


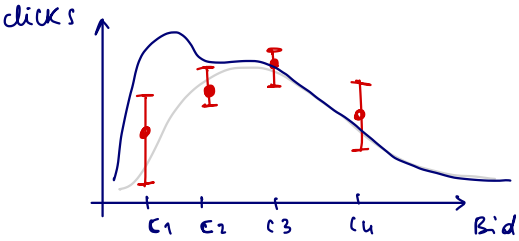
Failure of standard algorithms

- standard bandid algorithms fail in non-stationary environments because they exploit confidence bands which reduce monotonically
- The main reason is that their bands reduce: bands get closer linearly as the number of observed samples increases and logarithmically in time

⇒ UCB1 and abrupt changes





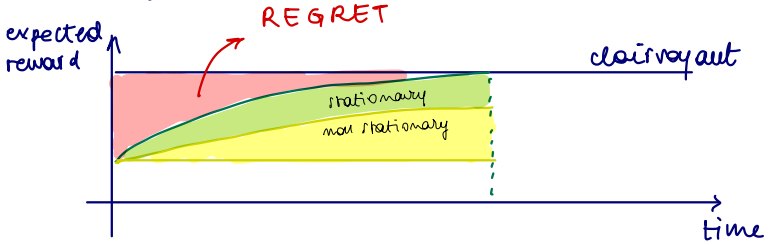


if there is a change of the demand curve, then UCB1 needs a long time to understand that c_1 is the best and during all this time we continuously collect regret
→ exponential time, because the confidence bound increases logarithmically

⇒ We need that confidence bands don't monotonically reduce in time.
A solution could be that we can use a sliding window to forget old samples
- ↳ confidence bands are computed by looking at last samples
- ↳ we get less accurate estimates and suboptimal decisions

- Regret lower bound

The regret lower bound is larger than in stationary environments



SW-UCB1 algorithm

step 1: play once every arm $a \in A$

step 2: at t play arm a_t with $n_{a_t}(t-1, \tau) = 0$, otherwise play arm a_t such that

$$a_t = \underset{a \in A}{\operatorname{argmax}} \left\{ \bar{x}_{a,t,\tau} + \sqrt{\frac{2\log t}{n_a(t-1,\tau)}} \right\}$$

$\bar{x}_{a,t,\tau}$: empirical mean of x_a at time t computed with last τ samples

$n_a(t,\tau)$: number of samples of arm a in τ steps

SW-TS algorithm

step 1: start with $\alpha = 1$, $\beta = 1$

step 2: at every time, for every arm a

$$\tilde{\vartheta}_a = \text{sample}\,(\mathbb{P}(\mu_a = \vartheta_a))$$

step 3: at every time t play arm a such that

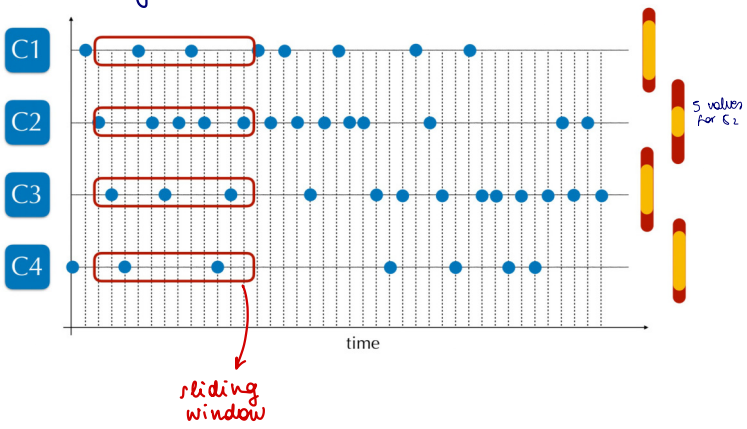
$$a_t = \underset{a \in A}{\operatorname{argmax}} \{\tilde{\vartheta}_a\}$$

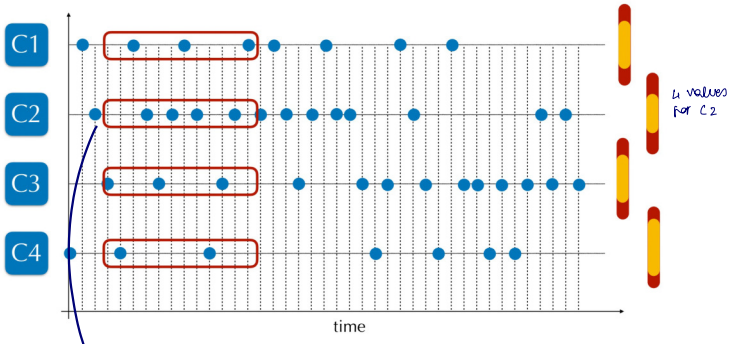
step 4: update Beta parameters of arm a_t

$t \le \tau$ $\quad (\alpha_{a_t}, \beta_{a_t}) = (\alpha_{a_t}, \beta_{a_t}) + (x_{a,t}, 1 - x_{a,t})$

$t > \tau$ $\quad (\alpha_{a_t}, \beta_{a_t}) = \max\left\{(1,1), (\alpha_{a_t}, \beta_{a_t}) + (x_{a,t}, 1 - x_{a,t}) - (x_{a_{t-\tau}(t-\tau)}, 1 - x_{a_{t-\tau}(t-\tau)})\right\}$

- Sliding window and confidence bounds





we discard this sample ⇒ the confidence band becomes larger (less accuracy)

☒ Abrupt change
- the number of breakpoints must be small wrt the time horizon

$$M = |\{b_1 .. b_{M+1}\}| = O(T^\alpha) \qquad \alpha \in [0,1)$$

- the regret is of the order $O(|A| T^{\frac{\alpha+1}{2}})$ → linear in the number of arms
  $\alpha = 1$ regret linear in T
  $\alpha = 0$ number of breakpoints constant in T

☒ Smooth change
- The change is sufficiently smooth
  $\sigma > 0 : |\mu_{it} - \mu_{it'}| < \sigma |t - t'|$
- The separation between the expected rewards of two arms is arbitrarily small only for a limited number of rounds
- the regret is of the order $O(|A| T^{\frac{\alpha+1}{2}})$

⇒ we can consider also the case of abrupt + smooth change
- time horizon is divided in phases (corresponding to abrupt changes)
- in every phase the change is smooth
- the regret is of the order $O(|A| T^{\frac{\alpha+1}{2}})$

→ in practice we don't know
- number of breakpoints
- coefficient of smoothness
- the size of the sliding window cannot depend on unknown parameters

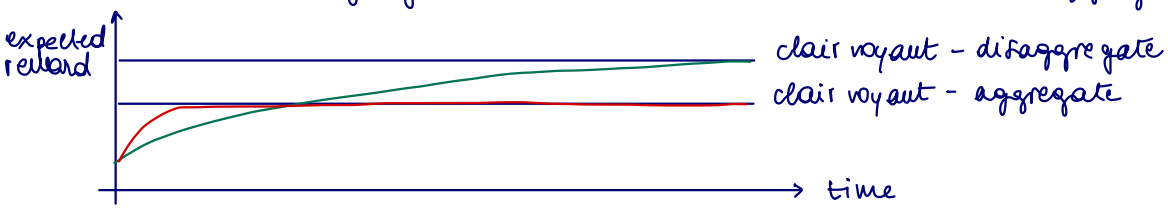
$\boxed{\tau \propto \sqrt{T}}$ length of τ

Rem • sliding windows force every candidate to be chosen repeatedly during time and can be re-evaluated in order to get the optimal one.
- Non stationary environments require <u>permanent exploration</u> (otherwise the algorithm suffers from a constant regret
  ⇒ the regret is much higher

CONTEXT GENERATION

→ We want to learn the classes of users, in particular users are divided through a ratio

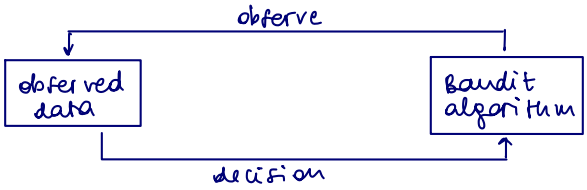
Consider users belonging to different classes. We can use an aggregate o disaggregate model

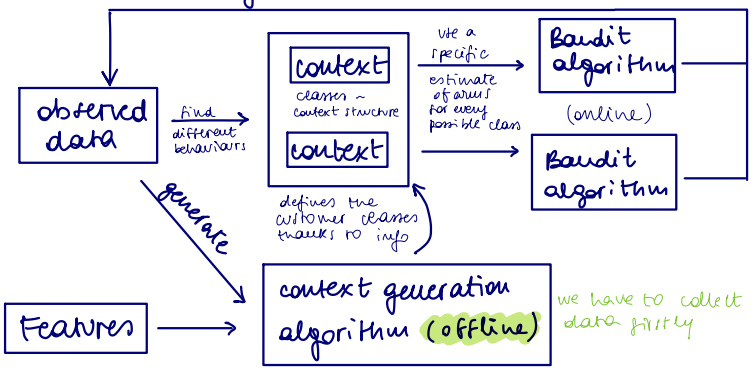


The performance depends on the amount of data we have : { many data → disaggregate model / few data → aggregate model

The best we can do is to start from an aggregate model and when we collect many samples we adopt a disaggregate model.

⇒ we have to learn the disaggregate model (so how many classes) from data





- Context generation algorithm

  Suppose to collect a lot of data and to be able to characterize data with different features. We're able to partition the space of features and to compare different partitions in order to evaluate the best one (male vs female → 2 different models ⇒ we split)

  → the algorithm learns the classes !

- How many partitions ?

  The number of partition is really huge ⇒ all the possible partitions can't be evaluated

  ↳ some heuristics guiding the search is needed

- How to compare the partitions ?

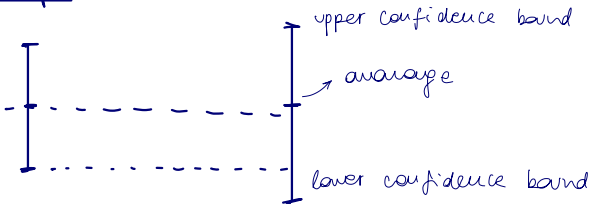
  If we have many partitions we need a lot of data in order to learn. The idea instead is to generate the context according to the data we get

  Actually with few data, the convergence of the bandit algorithms is too slow

  ⇒ partitioning makes sense when we can provide a strong evidence that is better to do

  ⇒ the comparison is made by looking a lower confidence bound

Example



Here we have two possible context structures.

A: one context → we have an accurate confidence bound

B: two contexts → we have more uncertainty in the context, so we have a larger confidence interval

→ the best we can do is to split many times in order to have a bigger average mean, but splitting too much means the learning slower

⇒ we look at lower confidence bound (average and confidence intervals are useless from a computational point of view)

- Formal model

  space of attributes : $F \subseteq \{0,1\}^t$

  context : $c \subseteq F$

  context structure : $P = \{ c : c \subseteq F, \bigcap_i c_i = \emptyset \ \bigcup_i c_i = F \}$ → partition

- How many contexts?

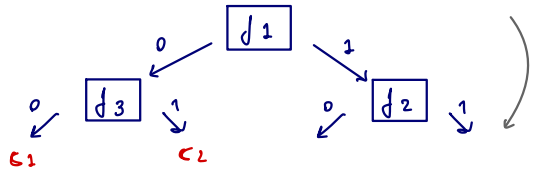
since the number of possible contexts is exponential, then the number of possible context structures is huge as well

→ the expected value of a context structure is $\sum_{c \in \mathcal{P}} p_c \cdot \mu_{a_c^*, c}$ → sum of expected rewards pulling the best arm in each context

probability to choose context c

- A greedy approach

since we have an exponential number of contexts, we iteratively identify the most promising feature and we split the rest accordingly with a feature tree (here binary features)



every step we choose a feature and we decide if splitting is worth

SPLIT $\iff$ the lower bound of the expected reward when we split $>$ lower bound of the expected reward when we don't split

if we look at lower bound we're sure that the whole performance is better

1° context 2° context

$$p_{c_1} \mu_{a_{c_1}^*, c_1} + p_{c_2} \mu_{a_{c_2}^*, c_2} > \mu_{a_{c_0}^*, c_0}$$

lower bound of original context

p_{c_1}: probability the user belongs to context c_1 · $\mu_{a_{c_1}^*, c_1}$: lower bound on the best expected reward for c_1 choosing the best arm

p_{c_2}: probability the user belongs to context c_2 · $\mu_{a_{c_2}^*, c_2}$: lower bound on the best expected reward for c_2 choosing the best arm

The choice of lower bound depends on the distribution. If it is finite, we can use the Hoeffding bound (with Bernoulli distribution)

$$\bar{x} - \sqrt{\frac{-\log \delta}{2|z|}}$$ where δ is the confidence and z the set of data

algorithm

step 1: for every feature evaluate the expected value after the split by using the lower bound

step 2: select the feature with the maximum marginal value if larger than the non split case

→ properties:

- NO optimality is guaranteed if we use a greedy algorithm
- The tree provides an immediate interpretation about the most important feature that affects the problem

MATCHING IN ONLINE ENVIRONMENTS

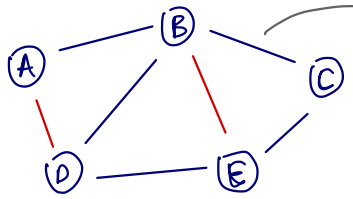
The online advertising data is an example of matching a user with an advertisement

MATCHING PROBLEMS AND ALTERNATING-PATH ALGORITHM

- matching problem formulation

We are given a graph composed by nodes and edges.
Each node represents a person and a link exists if 2 people share something

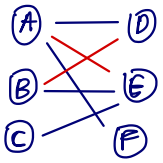
Def (matching)
A matching is a subset of edges of the graph such that no pair of edges share vertices
⇒ every vertex can occur in the matching only once



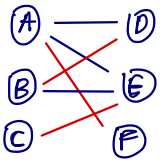
if it's red, then it would be an unfeasible matching since B would have joined 2 matchings

→ Actually most of our problems are explained by bipartite graphs

Def (maximal matching)
A maximal matching is a matching which cannot be enlarged
⇒ we cannot add a new edge



Def (maximum matching)
A maximum matching is a matching with largest cardinality
⇒ largest possible number of edges



Def (perfect matching)
A perfect matching is a matching in which all vertices are matched
⇒ all nodes are in the matching

⇒ a maximum matching always exists
a perfect matching may not exist

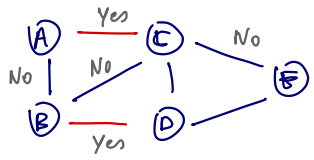
Matching problem: Given a graph, the goal is finding a maximum matching

- Alternating path algorithm

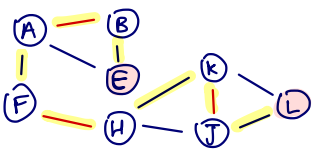
The most effective algorithms are
→ Hopkroft-Karp algorithm for bipartite graphs $O(m\sqrt{n})$
- Edmund algorithm for arbitrary graph $O(mn^2)$ // works with weighted graphs and it's an extention of HK
→ Micali-Vatirani algorithm for arbitrary graph $O(m\sqrt{n})$

All these algorithms are based on the concept of alternating path

Def (alternating path)
An alternating path is a path such that edges of the matching and edges of the complementary set (edges not in the matching) alternate

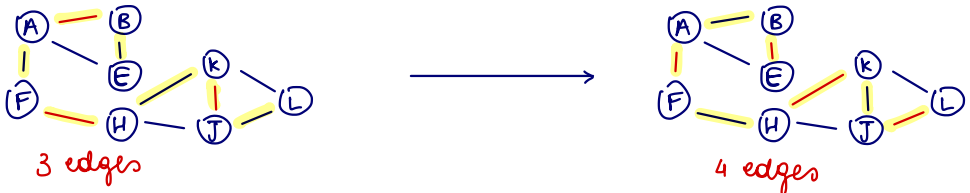


Def (augmenting path)
An augmenting path is an alternating path starting and ending in unmatched vertices



- Complementing an augmenting

→ Complementing an augmenting path results in an alternating path where the number of edges is increased by 1



→ A matching is maximum if it does not admit any augmenting path

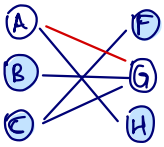
Alternating - path algorithm

1. look for an augmenting path ↕ repeat until no augmenting path exists
2. make the complementary ↳ we find a maximum matching

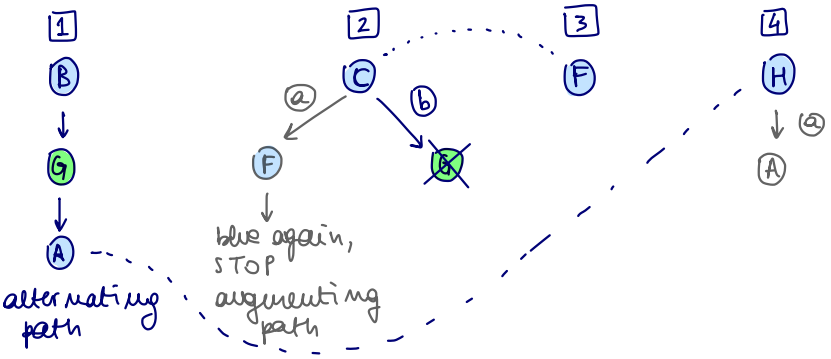
→ problem: looking for an augmenting path - we are given a matching + graph and we look for all the shortest augmenting path

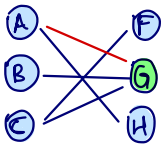
1. consider every unmatched vertex
2. make a breadth-first search starting from every unmatched vertex
3. Given the unmatched node, make an expansion adding that path and the connected path in the matching if the vertex is matched
4. label vertices as blue/green according to the level of search tree
5. If expanding the path we visit some already visited nodes then we stop the branch and we make an action according to the specific case

Example



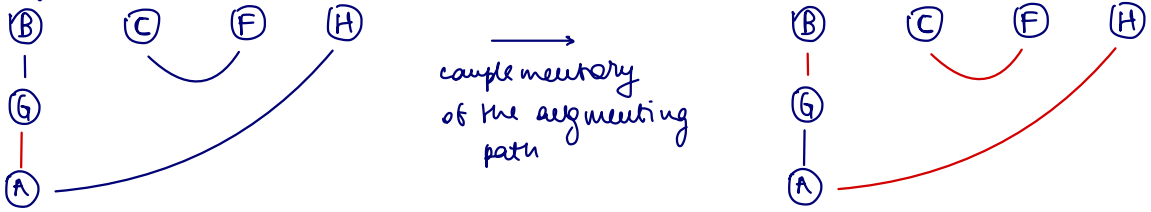
given a matching, we start from unmatched nodes B, F, C, H and do breadth-first search

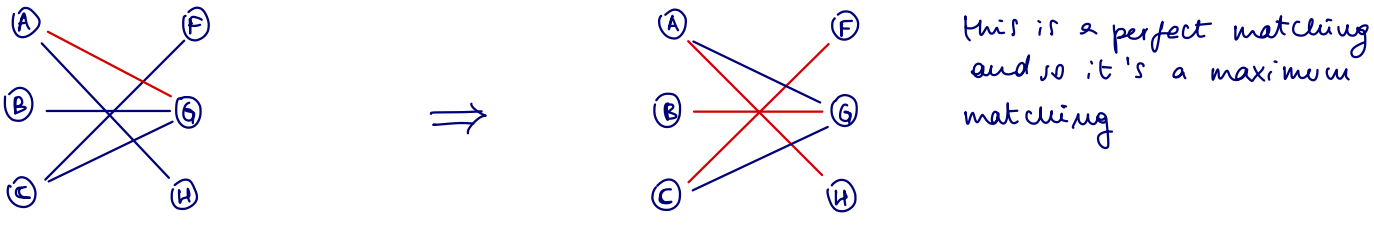




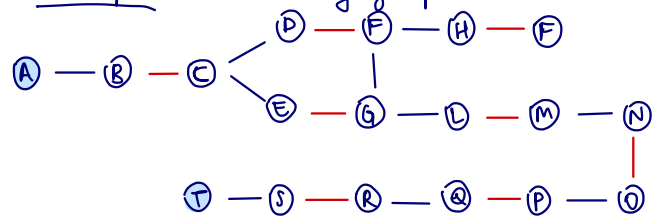
ⓐ If the expanding even node is directly connected to another even node (previously labeled) then we have found an augmenting path
⇒ such edge can be added to the matching

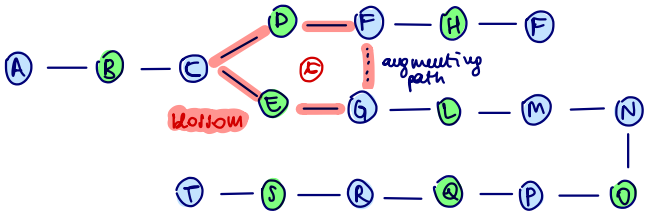
ⓑ If the expanding even node is directly connected to an odd node (previously labeled) belonging to another subtree (root), then stop the search



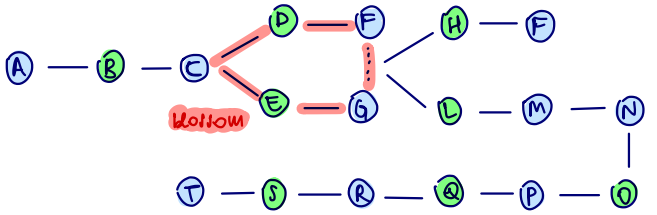


Example (arbitrary graph)



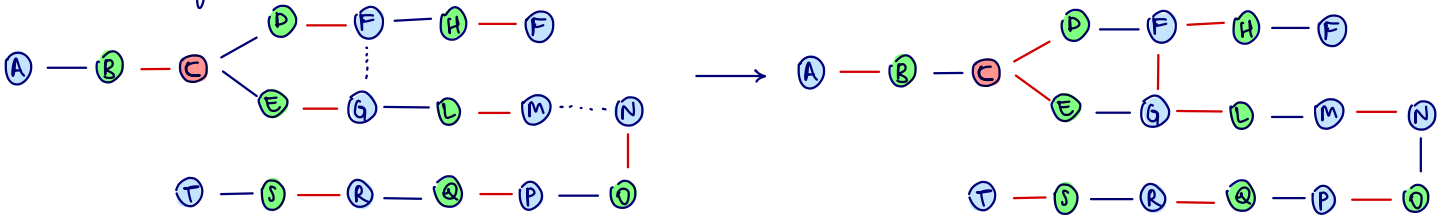


(E) If the expanding even node is directly connected to a vertex previously labeled as even and belonging to the same subtree then we have found a blossom
⇒ shrink the blossom (make it as one object)

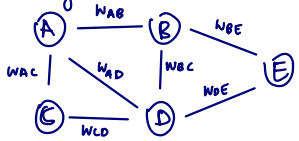


- How to deal with a blossom ?

If an augmenting path is found, the blossom is expanded and we do the complementary of the graph



Weighted matching problem ~ assignment problem
each edge has a weight : $w_{ij} > 0 \ \forall i,j$

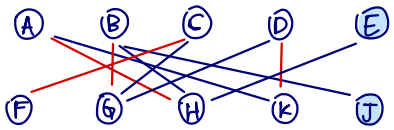


Def a maximum weighted matching is a matching with the largest cardinality, so the one that maximizes the sum of weights

- alternating path algorithm
  - Hungarian algorithm for bipartite graphs $O(n^2 m)$
  - Edmund algorithm for arbitrary graphs $O(mn^2)$

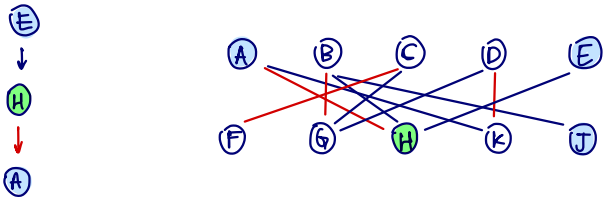
MATCHING IN BIPARTITE GRAPHS

~ Alternating path algorithm applied to a bipartite graph ~

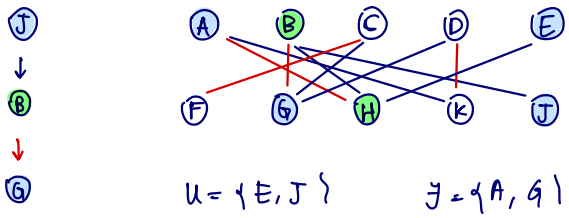


$M = \{(A,H), (B,G), (E,F), (D,K)\}$

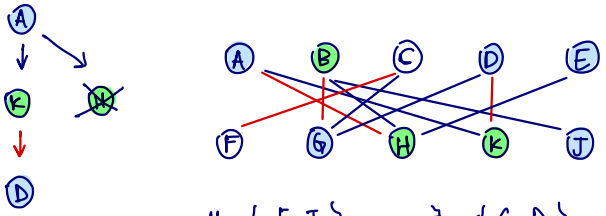
$U = \{E, J\}$ $\quad V = \{E, J\}$



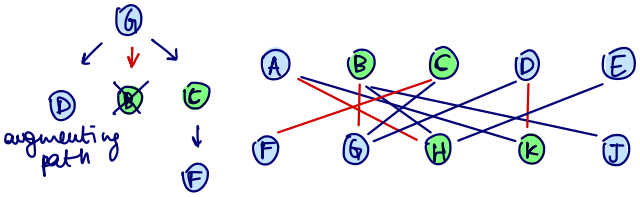
$U = \{E, J\}$ $\quad V = \{J, A\}$



$U = \{E, J\}$ $\quad Y = \{A, G\}$



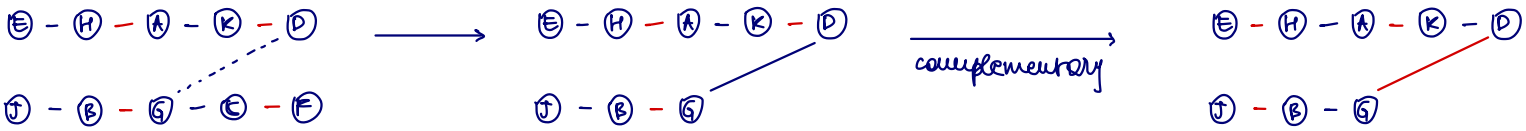
$U = \{E, J\}$ $\quad Y = \{G, D\}$



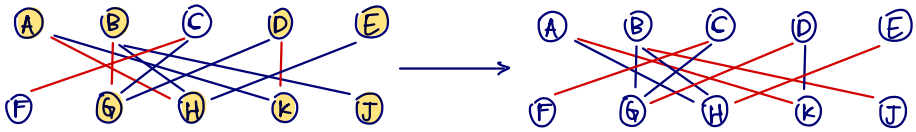
$U = \{E, J\}$ $\quad Y = \{D, F\}$

D → stop

F → C → stop



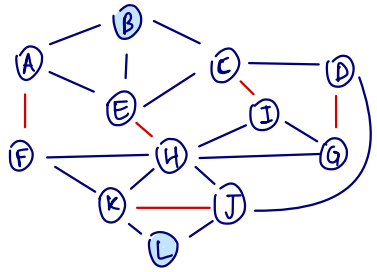
and so we find



$M = \{(A,K), (B,I), (C,F), (D,G), (E,H)\}$

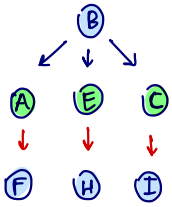
MATCHING IN ARBITRARY GRAPHS

~ Alternating path algorithm applied to an arbitrary graph ~

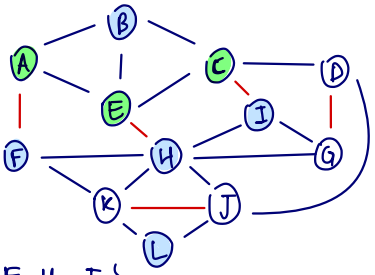


$M = \{(A,F), (C,I), (D,G), (K,J), (E,H)\}$

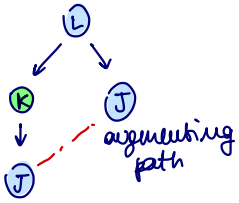
$U = \{B, L\}$ $\quad Y = \{B, L\}$



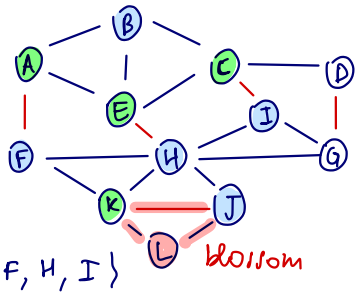
$U = \{B, L\}$



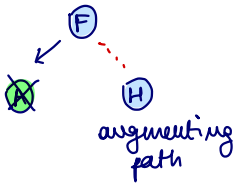
$Y = \{L, F, H, I\}$



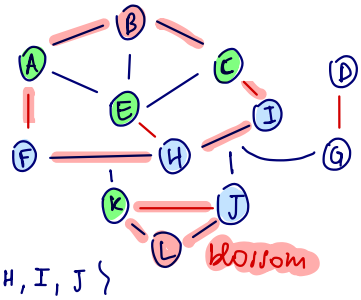
$U = \{B, L\}$



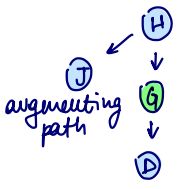
$Y = \{J, F, H, I\}$



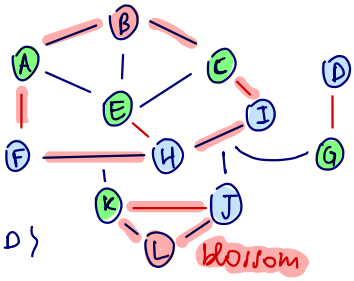
$U = \{B, L\}$



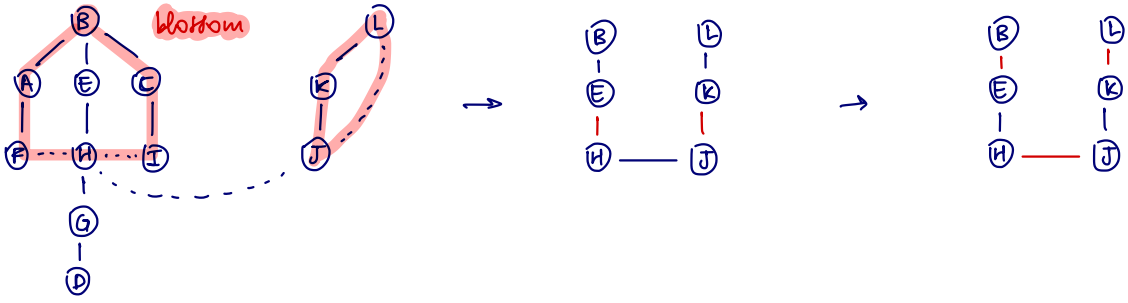
$Y = \{H, I, J\}$

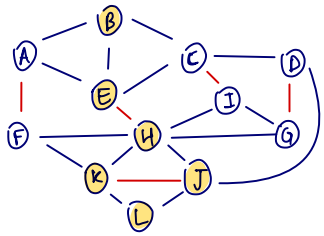
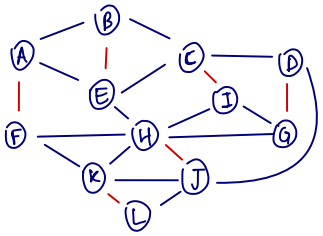


$U = \{B, L\}$



$Y = \{1, J, D\}$



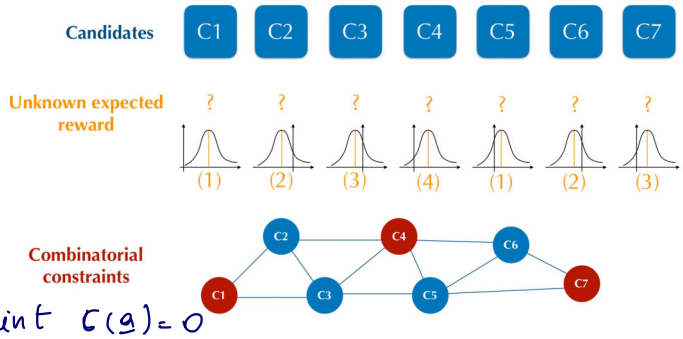



$M = \{(A,F), (B,E), (C,I), (D,G), (H,J), (K,L)\}$

COMBINATORIAL BANDITS AND MATCHING PROBLEMS

- Combinatorial constraints

Until now we said that candidates (arms) are drawn from an unknown probability distribution.
Now we add combinatorial constraints by pulling a subset of arms simultaneously



Def superarm: collection of candidates (arms)
feasible superarm: superarm satisfying the constraint $C(\underline{a}) = 0$
reward of a superarm: sum of the rewards included in a superarm

→ Goal: maximize the cumulative in time expected reward

- Assumption

The random variable of every arm is Bernoulli in $\{0,1\}$ with unknown mean

Thompson Sampling pseudocode

1. at every time t and for every arm a

$\tilde{\vartheta}_a \leftarrow \text{Sample}(\mathbb{P}(\mu_a = \vartheta_a))$

$\underline{a}$: superarm

2. at every time t, play superarm $\underline{a}_t$

$\underline{a}_t \leftarrow \underset{C(\underline{a})=0}{\text{argmax}} \left\{ \sum_{a \in \underline{a}} \tilde{\vartheta}_a \right\}$ → solve a combinatorial problem

3. Update the Beta distribution

$(\alpha_{a_t}, \beta_{a_t}) \leftarrow (\alpha_{a_t}, \beta_{a_t}) + (X_{a_t,t}, 1 - X_{a_t,t})$

optimization problem

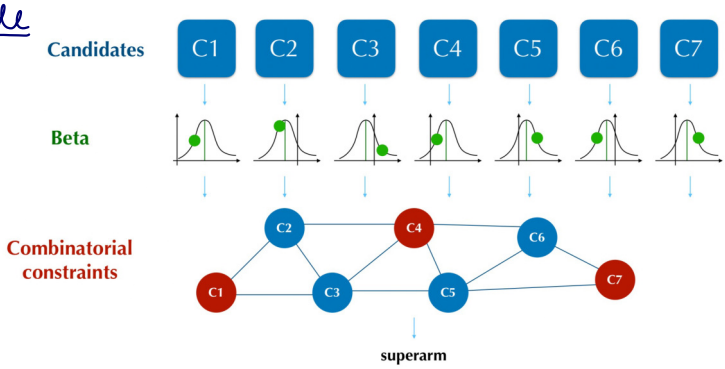
$\max f(x)$
x
$s.t.: g(x) = 0$

we have uncertain parameters
- they're Bernoulli
- drawn from a Beta

] 1.

obtain the estimate of parameters and update in order to find x

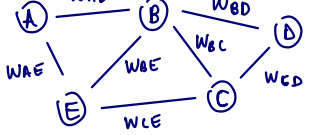
Example



- a beta distribution is associated to each arm.
- we draw a sample
- we solve the associated combinatorial problem
- find the superarm: $C(\underline{a}) = \{C_1, C_4, C_7\}$

Def (TS Regret) $R_T(TS) = \vartheta\left(|A| \frac{\log T}{\Delta \min}\right)$

- weighted matching problem and TS



$w_{ij} > 0 \quad \forall ij$ unknown → The value of each edge is rv whose expected value is unknown

→ matching ~ Be(p) p = probability to pair

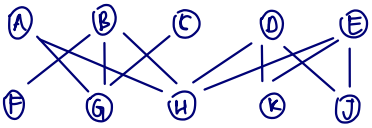
Steps:
1. choose a superarm (edges)
2. Drawn samples
3. Collect rewards
4. Update Beta parameters

↕ iterate for T rounds.
we converge to the best matching because combinatorial bandit guarantees to minimize the regret

ONLINE MATCHING PROBLEM

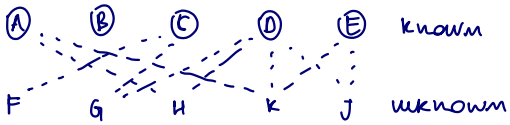
~ Matching problems which are online in time ~

classical matching problem



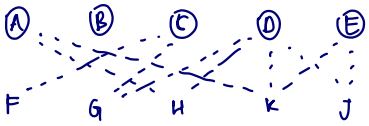
graph is completely known a priori

Online matching problem



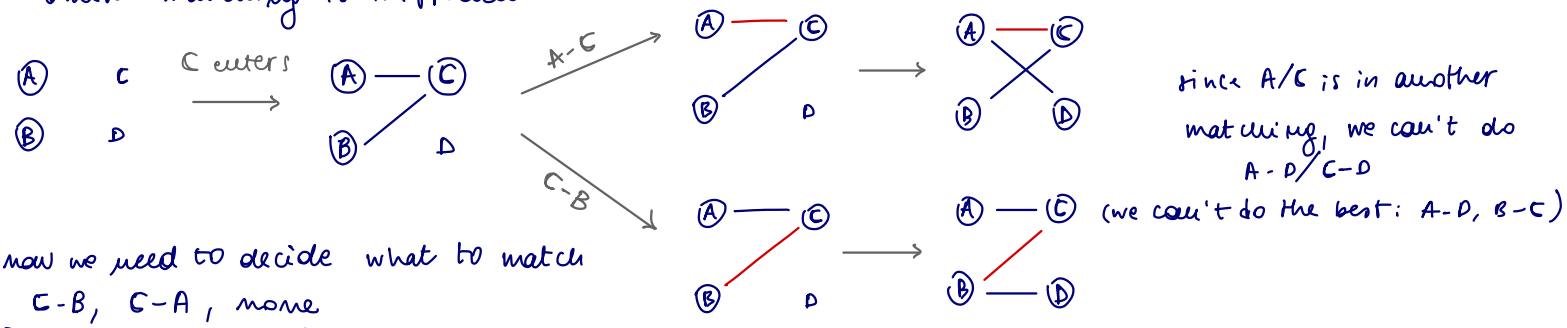
graph is discovered during time

☒ Basic bipartite online matching



1. One side of the bipartite graph is known a priori
2. The other side and edges are unknown at the beginning
3. at each round a single node enters the problem and its edges are discovered
4. at each round we decide if the entering node is matched with another node or it leaves the problem

- Online matching is inefficient



now we need to decide what to match
C-B, C-A, none

inefficiency: the optimal choice depends on what happens in the future

☒ Clairvoyant vs online optimal solution

clairvoyant optimal solution → best solution we have with complete information about the problem

Def competitive factor of an algorithm $\alpha_A = \min_P \frac{\Gamma_A(P)}{\Gamma^*(P)} = \frac{\#\text{ matchings of the algorithm}}{\#\text{ matchings of clairvoyant solution}}$

Rem
- No deterministic algorithm can have a competitive factor > 1/2 in online bipartite matching problems
- No randomized algorithms can have an expected competitive factor > $\frac{e-1}{e}$ in online bipartite matching problem

☒ Deterministic algorithm

Match a node with an arbitrary unmatched node
→ competitive factor = 1/2

proof

1° step: the cardinality of any maximal matching is at least 1/2 of the cardinality of the maximum matching

let M^* the maximum matching (largest possible matching)
let M the maximal matching (cannot be enlarged)

$\forall e = (u,v)$ edge in M^*, either u or v is matched in M, otherwise M is not maximal
$\Rightarrow$ # matched nodes in $M \geq |M^*|$ more edges involved

since # matched nodes in M^* is $2|M^*| \Rightarrow |M| \geq |M^*|/2$

2° step: the greedy algorithm always find a maximal matching

Let M be maximal
$\forall e=(u,v)$ edge not in M, consider v:
- v has been matched to some node other than u
- v has not been matched since all the connected nodes (also u) were already been matched

→ in both cases (u, v) cannot be added to M

Randomized algorithm
1. Generate randomly an ordering over the nodes that are initially available
2. match a node with first unmatched node in the ordering that we generated initially
   → competitive factor of $1/1-e$

GENERALIZATION OF ONLINE MATCHING PROBLEMS

Problem
1. The graph is discovered during time
2. Every node may wait for at most k rounds before leaving
3. k may be node specific

hp: graph is weighted
graph could be arbitrary or bipartite

Prop (Negative result)
No deterministic competitive algorithm even when k is the same for all nodes

competitive: $\alpha_k > 0$

Proof suppose all nodes stay in the problem for 2 rounds

1. A enters the problem (A)
2. B enters the problem → Do we match A and B?
   - ↳ match: (A) — 1 — (B) ---- >>1 (C) → $\frac{APX}{OPT} = \frac{1}{>>1} \to 0$ — match A-B / best clairvoyant sol B-C
   - ↳ don't match: (B) → $\frac{APT}{OPT} = \frac{0}{1} = 0$ — don't match / best clairvoyant sol A-B

⇒ independently from the choice we do, we have the worst case in which the competitive factor = 0, so there's no deterministic algorithm returning a strictly competitive factor.

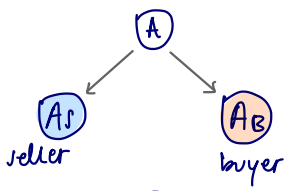
Positive result → If we resort randomized algorithm we can get a strictly competitive factor which is an expectation wrt the randomization of the algorithm

→ Postponed Dynamic Refered Acceptance (randomized) algorithm has an expected competitive factor of (1/4) when nodes stay in the problem for the same number of rounds

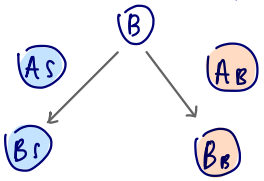
1/4 for arbitrary
1/2 for bipartite

Postponed Dynamic Refered Acceptance
idea of the algorithm: 2-sided market (explore with a market approach)

1° step: every time a node enters the problem, the algorithm generates a fictitious bipartite graph in which we have buyers and sellers

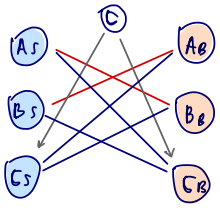


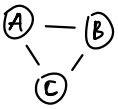
2° step: suppose A doesn't leave the problem. Then node B enters the problem



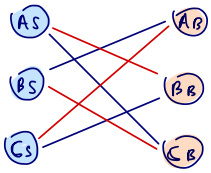
→ If A-B in original graph ⇒ $A_S - B_B$ and $B_S - A_B$ and we have to find the best matching among those 2

3° step : suppose A and B don't leave the problem.
Then node C enters the problem





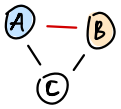
4° step : A gets its deadline, so
a. we match it
b. otherwise it leaves without being matched
⇒ find the best matching



a) select A as seller (IP = 1/2)

A — B

A and B are matched



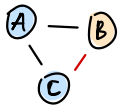
b) select A as buyer (IP = 1/2)

A — B

since the optimal matching is the red one, then we remove A without matching it

5° step : No other nodes enters the problem.
B gets its deadline.

B — C



■ worst case example

A —(1)— B —(>> 1)— C

a) select A as seller

A —(1)— B C

match A - B with IP = 1/2

the algorithm returns a matching of value 1 with probability 1/2

b) select A as buyer

~~A~~ — B —(≥ 1)— C

match B - C with IP = 1/2

the algorithm returns a matching of value >> 1 with probability 1/2

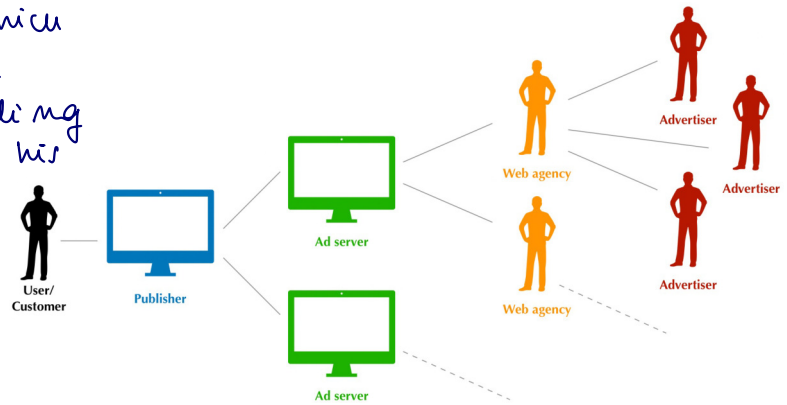
⇒ competitive factor : $\frac{1}{2}\left(1 + \gg 1\right) / \gg 1 \simeq 1/2$

→ The competitive factor is about 1/2 (both if C enters or not the problem)

⇒ In worst case example we get a strictly positive competitive factor, here we have a strictly competitive factor in expectation wrt randomization of the algorithm

ONLINE ADVERTISING DATA

- A user / customer visits a webpage proposed by a publisher (owner of website) in which some advertisers want to show an ad.
- The advertiser wants to publish an appealing ad, so he pays the web agency to make his ad optimized.
- The web agency tries to optimize the target and the budget of each ad



- Performance indices
  1. number of impressions → awareness
  2. number of clicks → traffic
  3. number of leads → potential sales
  4. number of conversions → sales

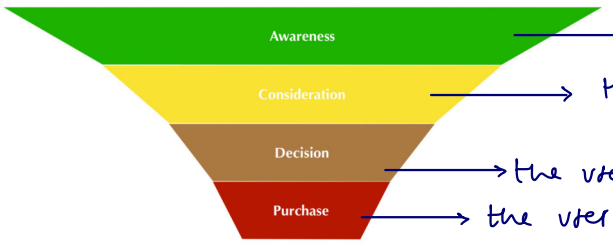
The goal is to maximize the conversions

- Format
  1. Display advertising → based on cost per impression
  2. Search advertising → based on pay-per-click
  3. Social advertising → based on both

- Payment scheme: when an advertiser pays the publisher he has to pay for
  1. cost per impression
  2. cost per click

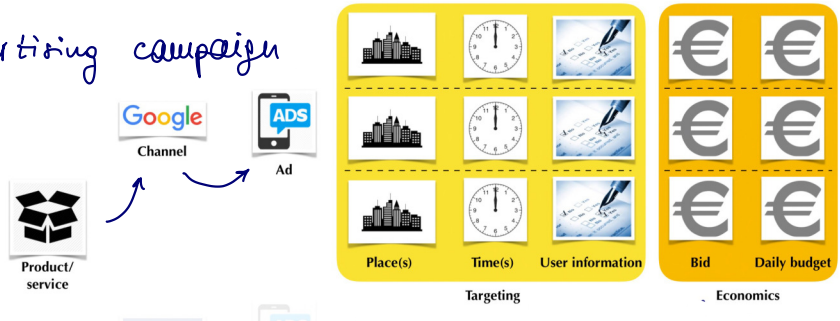
→ in order to evaluate the payments we need to consider all performance indices

Marketing funnel



→ user discovers the company and the product
→ the user compares the product with the product of other companies
→ the user looks for additional information about the product
→ the user buys the product

Advertising campaign



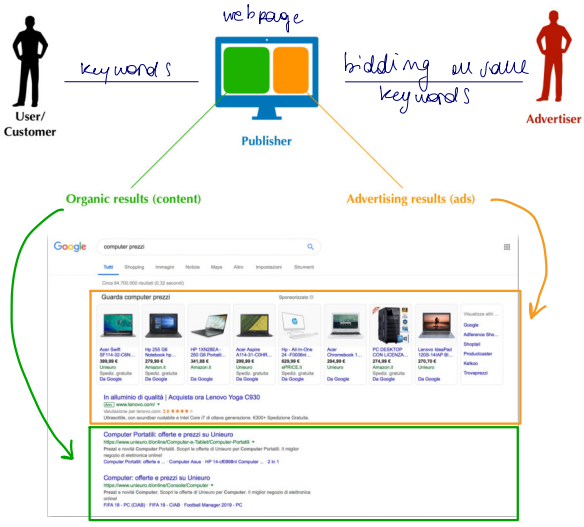
1. choose the channel (FB or Google)
2. choose the ads for each channel
3. choose the targeting → we get place, time and user information
   (for same ad we have multiple targeting)
4. choose the bid and the daily budget
   - bid: amount that the advertiser will pay for a single click
   - daily budget: max amount of money that an advertiser would pay in a day for a given sub campaign

Channel	Ad	Targeting			Economics	
		Place	Time	User's information	Bid	Daily budget
Google	Ad1	Milan	Morning	car	1 Euro	10 Euro
Google	Ad1	Italy	Afternoon	car	2 Euro	1 Euro
Google	Ad2	...	...	sport	...	...
subcampaign						

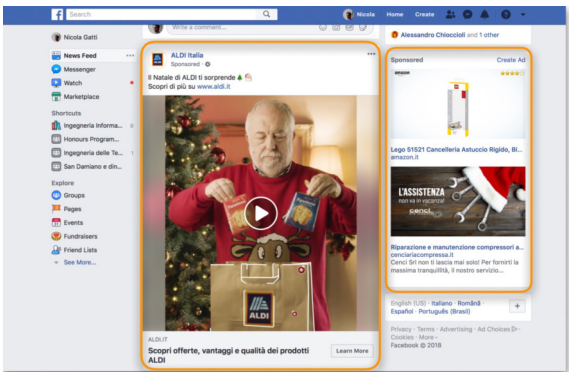
☑ User's information
channels distinguish user's information available
- display advertising → cookies (behaviour of users)
- social advertising → interests, friends, previous behaviour
- search advertising → keywords, IP, query

PAY PER CLICK ADVERTISING

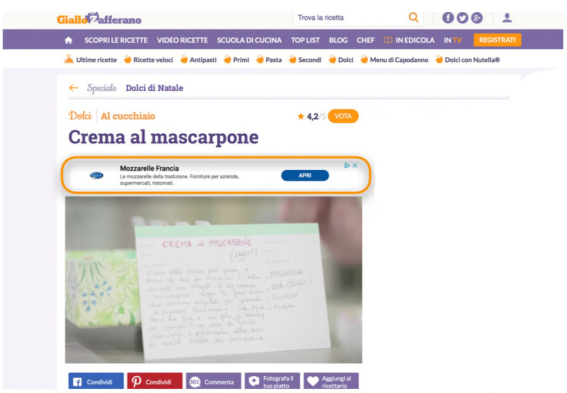
☑ search advertising
The advertiser submits an ad to the publisher
The publisher sets the bid (on some keywords)
The user makes the query



Once the user makes a click on the ad, the advertiser pays for such a click



☑ social advertising
It's similar to search advertising but it uses social networks as Facebook



☑ contextual advertising
It scans a page, looks for keywords and generate a set of pertinent ads

Publisher side problem

We focus on the optimization problem of the publisher

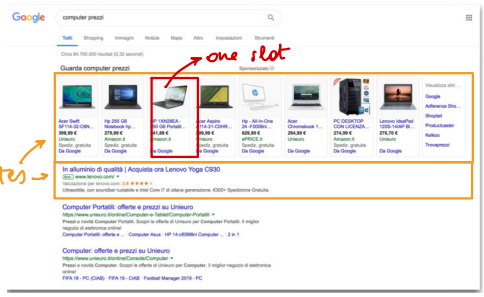
→ The publisher must produce the webpage including the content and ads and he defines the pay-per-click payment

- Webpage generation

The publisher can display ads in different slates and each slate can be divided in one or more slots.

In each slot there's a single ad.

→ search engines allow a single ad to span over multiple slots



Mathematical model

The publisher has to maximize the best location of ads in slots

⇒ the optimization problem is to maximize the value of displaying an ad into a specific slot wrt the budget and bid

expected value of having an ad "a" in slot "s" $= \Lambda_s \, q_a \, v_a$

- Λ_s: probability that the user observes the slot "s" — slot prominence
- q_a: probability to click the ad "a" observed in slot "s" by the user — ad quality
- v_a: value per click of ad "a" — ad value

$\Lambda_s q_a$ → probability to click an ad "a"

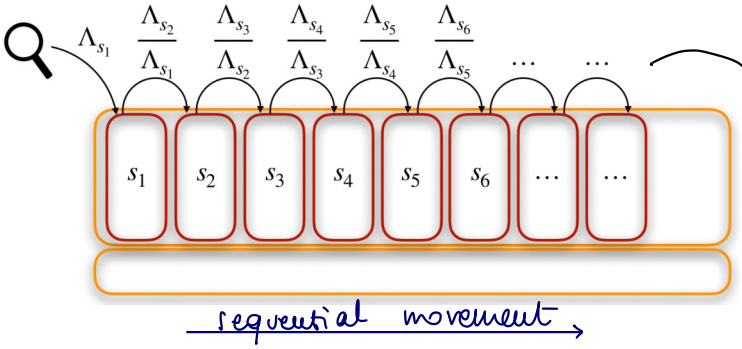
non è il contrario?

→ expected value of an allocation $\sum_a \Lambda_{s(a)} \, q_a \, v_a$

$s(a)$: function that returns the slot in which each ad is allocated (otherwise $s(a)=0$)

* cascade

The idea behind the prominence of the slot is that the user follows a cascade model



probability to move from one slot to the following one $\frac{\Lambda_{s_{i+1}}}{\Lambda_{s_i}}$

→ Goal: find a location that maximizes the cumulative value of the slot

In order to find the best allocation we have to maximize the expected value

$$\underset{s(a)}{\operatorname{argmax}} \left\{ \sum_a \Lambda_{s(a)} \, q_a \, v_a \right\}$$

* algorithm

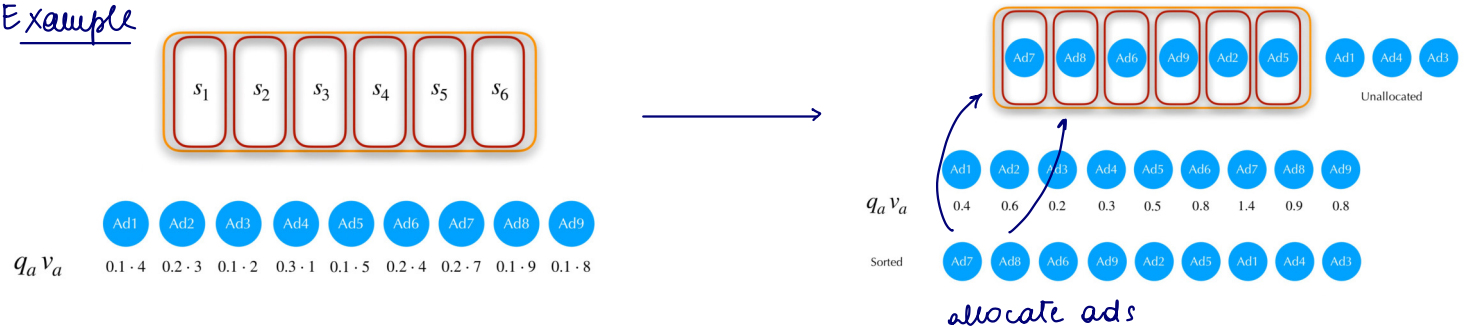
1° step: sort the ads in decreasing order in $q_a \, v_a$

2° step: allocate according to such order

order of $O(n \log k)$

- linear in the number of ads
- logarithmic in number of slots

Example



- known and unknown parameters
  $q_a v_a$ is crucial to find the best allocation, but in general allocation mechanism doesn't know it

  λ_s → estimated by search engine using data of all allocated ads
  q_a → estimated by the search engine by means of bandit algorithms (with upper bounds)
  v_a → private information of the advertiser
  The advertiser communicates v_a by making a bid to search engine which can be the true evaluation or not

⇒ The advertiser cannot have a correct estimation of the true value because v is estimated by the behaviour of activities of website of advertiser

Auction mechanism

When the search engine generates a page, it runs an auction in order to find the best allocated ads and the pay-per-click-payments

- every ad has a different advertiser
- every advertiser makes a bid which determines the max amount of money the advertiser will pay for a click
- the auctioneer chooses → payments (different ways)
  ↘ allocation using bids v_a

* payments

1° method → GPS (Generalized Second Price)

$$p_a = \frac{q_{a+1} v_{a+1}}{q_a}$$

→ the ads are sorted in decreasing order in the expected value

$p_a < \frac{q_a}{q_a} v_a = v_a$: pay-per-click-payment < bid ⇒ if an advertiser doesn't make a bid larger than its value he will never pay more than its value

2° method → VCG (Vickrey-Clarke-Groves)

$$p_a = \frac{1}{\lambda_{s(a)} q_a} (x_a - y_a)$$

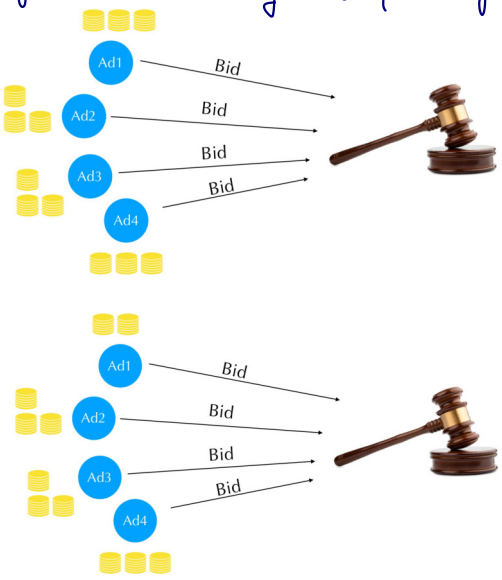
$$x_a = \max_{s(a')} \sum_{a' \neq a} \lambda_{s(a')} q_{a'} v_{a'}$$

$$y_a = \sum_{a' \neq a} \lambda_{\bar{s}(a')} q_{a'} v_{a'}$$

$$\bar{s}(a') = \operatorname{argmax}_{s(a')} \sum_{a'} \lambda_{s(a')} q_{a'} v_{a'}$$

Repeated auctions

In practice every advertiser has a daily budget and take part in auctions only if the remaining daily budget is not expired



every time a new user enters_

all advertisers who have budget make a bid and the auctioneer chooses allocation and payments

the user clicks on ad 1 → ad 1 is the winner and it pays the click ⇒ its daily budget reduces until it ends

- During the day
  1. the bid for a subcampaign can be changed during the day (with an unknown delay)
  2. the budget for a subcampaign can be set <u>once per day</u>

- properties

  GSP → bidding the actual value may not be the optimal strategy even without any budget constraint

  VCG → bidding the actual value is the optimal strategy without any budget constraint
  ⇒ truthful when played once but in repeated scenario is not truthful anymore
  → bidding the actual value may be not the optimal strategy with budget constraints

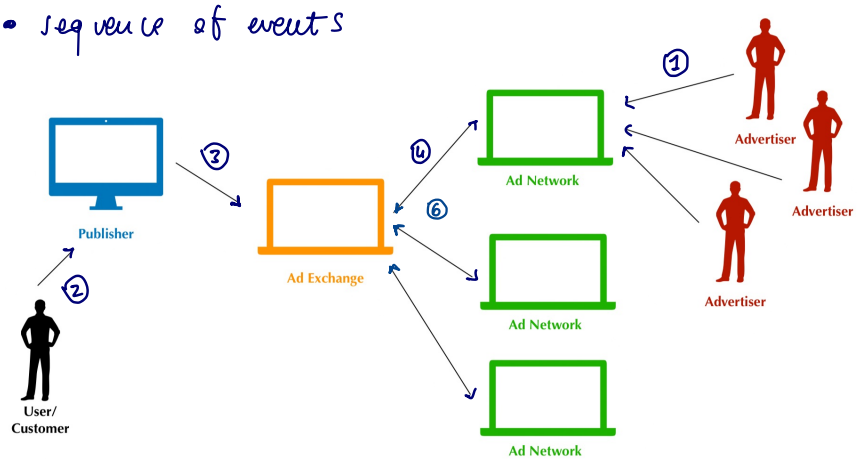
⇒ optimization problem:

Given a daily budget for a sub-campaign, what is the best (constant) bid?

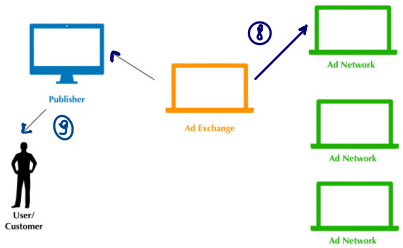
‖ Given a daily budget for a campaign, what is the best constant bid and daily budget for every sub-campaign?

DISPLAY ADVERTISING

- sequence of events



1. the advertisers communicate their ads to the ad network with all data in terms of targeting bids
2. an user "u" visits webpage "w" of publisher p(w)
3. before generating the webpage, the publisher p(w) contacts ad exchange E with $(w, I(u), r)$ = (webpage, info(user), context)
4. Ad exchange E contacts the ad network with $(I(w), I(u))$ = users
5. Every ad network runs an auction to determine the winning ad
6. Ad network "i" communicates the ad with the corresponding bid and the budget to the exchange → (b_i, d_i) = (bid, budget)
7. Ad exchange determines the winner via an auction
8. Ad exchange communicates the winner (b_i, d_i) to the publisher and the price to the winning ad network
9. The publisher serves the webpage to the user with the impression of the ad



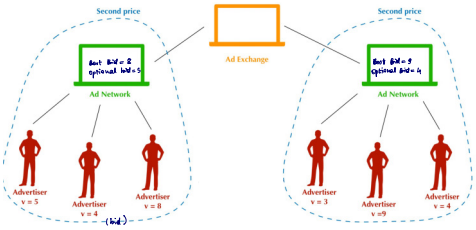
Ad network auction mechanism

Every ad network runs a second price auction to determine the winner of the network and its payment

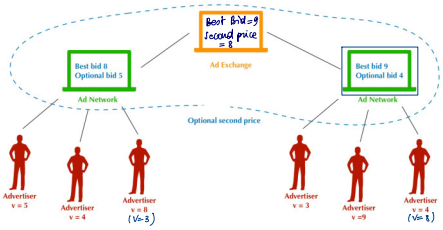
→ every ad network communicates to the ad exchange
- The winning bid and the ad (of the auction)
- Another optional bid (potentially the second price)

→ the ad exchange runs a second price auction to determine the winner and the payment

Example



both ad networks run second price option
→ the winner is the one with the highest bid and it pays for the second highest bid



the ad exchange runs the optional second price and it selects the best bid and the second price

OPTIMIZATION PROBLEM

Channel	Ad	Targeting			Economics	
		Place	Time	User's information	Bid	Daily budget
Google	Ad1	Milan	Morning	...	?	?
Google	Ad1	Italy	Afternoon	...	?	?
Google	Ad2	...	...	...	?	?
Google	Ad2	...	...	...	?	?
Bing	Ad3	...	...	...	?	?
Bing	Ad4	...	...	...	?	?
Facebook	Ad5	...	...	...	?	?
Facebook	Ad5	...	...	...	?	?
Facebook	Ad6	...	...	...	?	?
Facebook	Ad6	...	...	...	?	?

- Assumptions
  1. The performance of every subcampaign is independent from the performance of the other subcampaigns
     → not true in general, good approximation
  2. The values of bid and daily budget are finite and given
     → not true, but reasonable

$$\max \sum_{j=1}^{N} v_j m_j(x_{jt}, y_{jt})$$

max value per click (impression) · number of clicks (depending on bid and budget)

$$\sum_{j=1}^{N} y_{jt} \leq y_t \quad \forall t$$ → daily budget

$\underline{x}_{jt} < x_{jt} < \overline{x}_{jt} \quad \forall t,j$ → range of bid for each subcampaign

$\underline{y}_{jt} < y_{jt} < \overline{y}_{jt} \quad \forall t,j$ → range of budget for each daily budget

- j = subcampaign
- x_{jt} → bid of subcampaign j at day t
- y_{jt} → budget of subcampaign j at day t
- v_j → value per click
- m_j → number of clicks

algorithm

step 1 : remove the dependencies of the values of the bid (?)

$vn(x_i, y_j)$	bid x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	max
budget y_1	100	110	115	100	max	95	90	90	115
y_2	120	130	140	132	125	120	115	115	
y_3	135	140	143	145	138	130	125	120	
y_4	148	153	156	160	152	145	137	130	
y_5	152	155	158	163	165	157	140	135	
y_6	158	160	163	165	168	165	150	145	
y_7	164	168	170	174	178	170	163	158	
y_8	166	170	174	178	181	184	171	160	

we find the max bid for each daily budget

If these values are larger than the upper bound or smaller than the lower we have an unfeasible case

step 2 : find the best allocation for each budget

subproblem	budget 0	10	20	30	40	50	60	70
0	0	0	0	0	0	0	0	0
+ c_1	$-\infty$	90	100	105	110	$-\infty$	$-\infty$	$-\infty$
+ c_2	$-\infty$	90	172	182				
+ c_3								
+ c_4								
+ c_5								
c_2	0	82	90	92	$-\infty$	$-\infty$	$-\infty$	$-\infty$

we have c_1 and c_2

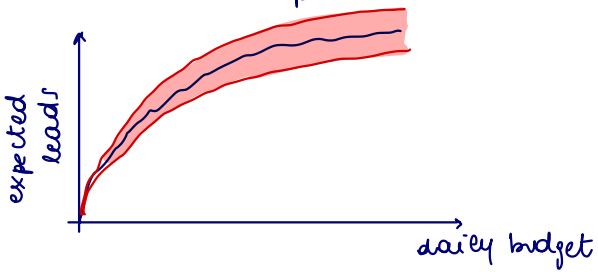
best solution and we take the max

budget = 10 { c_1 has budget = 10 → 90 ; c_2 has budget = 0 } { c_2 has budget = 10 → 82 ; c_1 has budget = 0 }

budget = 20 { c_1 has budget = 20 → 100 ; c_2 has budget = 0 } { c_1 has budget = 10 → 90 ; c_2 has budget = 10 → 82 } → 172 { c_1 has budget = 0 ; c_2 has budget = 20 → 90 }

budget = 30 { c_1 has budget = 30 → 105 ; c_2 has budget = 0 } { c_1 has budget = 20 → 100 ; c_2 has budget = 10 → 82 } → 182 { c_1 has budget = 10 → 90 ; c_2 has budget = 20 → 90 } → 180 { c_1 has budget = 0 ; c_2 has budget = 30 → 92 }

Real-world example



it shows how many leads the advertiser can get for every cumulative daily budget that is allocated on a campaign

→ a campaign can decide to increase or decrease the daily budget spent wrt the improvements of the number of leads

→ complexity : $O(NK^2)$

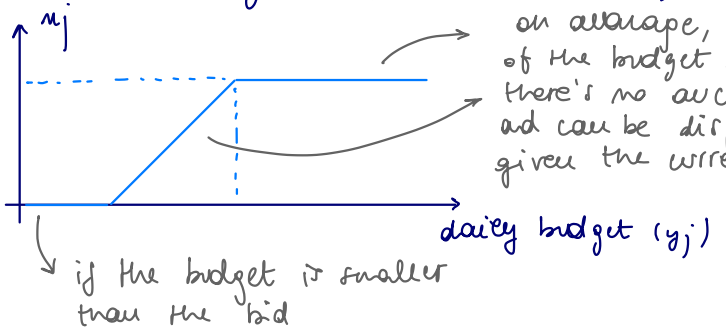
- linear in the number of subcampaigns
- quadratic in the number of values for the daily budget

REGRESSION BY GAUSSIAN PROBLEM

The optimization problem can be solved once we know $m_j(x_{jt}, y_{jt})$ $\forall\, x_{jt}, y_{jt}$ which are the number of clicks given a value of bid and daily budget
→ learning every possible combination is unfeasible (it requires too many samples)

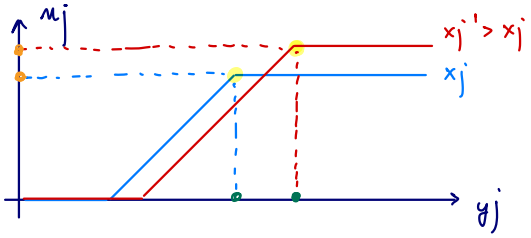
- Approach:
  1. the function is smooth
  2. The dependency of the function on bid and budget can be factorized

☒ understanding the phenomenon (x_j = constant)



on average, the dependency of the budget is linear until there's no auction in which the ad can be displayed in a slot given the current bid

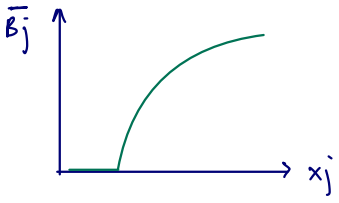
if the budget is smaller than the bid



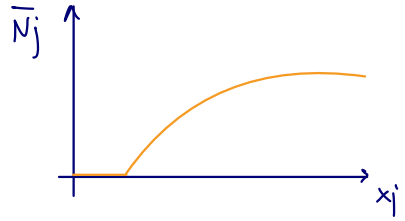
we expect with x_j' to have a smaller slope, because the spent per click is higher, but it can reach larger number of clicks as the budget increases

⇒ if we increase the bid, the number of options in which the ad is displayed increases because it's more competitive, but it requires more daily budget

→ we want to learn the daily budget which makes the number of clicks constant



budget beyond which the number of clicks is constant

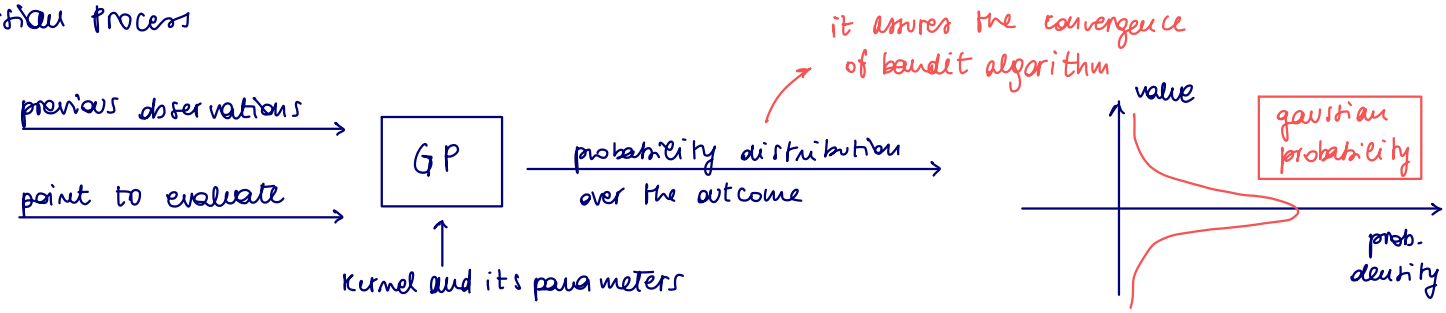


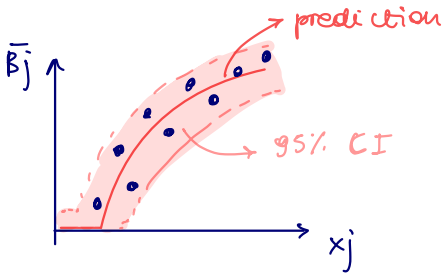
max number clicks achievable

$$\Rightarrow m_j = \max\left\{0, \min\left\{y_j, \bar{B}_j(x_j)\right\} - x_j\right\} \frac{\bar{N}_j(x_j)}{\bar{B}_j(x_j) - x_j}$$

☒ Regression problem
we want to estimate $\bar{B}_j(x_j)$ and $\bar{N}_j(x_j)$, but even if we discretize x_j we need too many samples
→ we assume smoothness (correlation among points)
⇒ we use a Gaussian Process (GP)

- Gaussian process





by increasing the number of samples the CI has more accuracy

COMBINATORIAL GP BANDITS

- standard bandit algorithms
  - we are given a set of arms
  - we can pull a single arm
  - we observe the reward of the pulled arm
- GP bandit algorithms
  - arms are correlated
  - the reward of an arm provides information on the reward of the arms closed to it

* GP-TS algorithm

1° step : at every time t, for every arm a

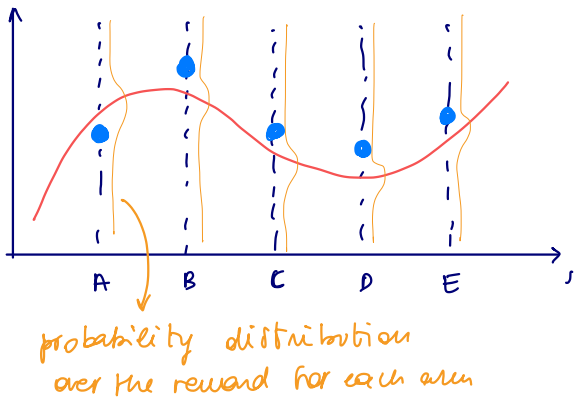
$$\tilde{\vartheta}_a = \text{sample}\left(\mathbb{P}(\mu_a = \vartheta_a)\right) \quad \text{(by GP)}$$

2° step : at every time t, play arm a_t such that

$$a_t = \underset{a \in A}{\text{argmax}} \{\tilde{\vartheta}_a\}$$

3° step : update the GP according to the observed reward

Example (Non combinatorial)



select the highest sample

- Regret bound

$$R_T \leq \sqrt{\frac{8}{\log\left(1 + \frac{1}{\sigma^2}\right)} T B \gamma_T} \qquad \text{where } B = 8 \log\left(\frac{T^4 M}{6\delta}\right)$$

γ_T: info gained at time t
σ^2 : variance of GP
M : number of arms

- Why GP regression
  - The GP plays a crucial role in bandit algorithms because it returns a measure of uncertainty of the regression (CI) (wrt linear regression)
  - The probability distribution is necessary for the exploration and convergence to the optimal solution

Combinatorial GP bandits

We can pull any set of arms satisfying some combinatorial constraint

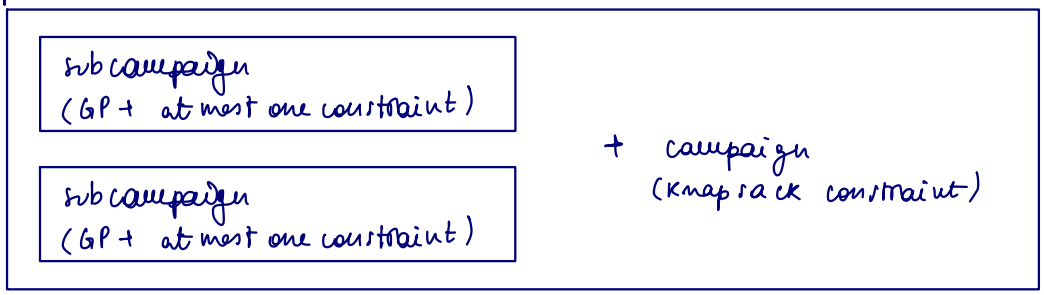
- Regret

$$R_T \leq \sqrt{\frac{2\Lambda^2}{\log\left(1+\frac{1}{\sigma^2}\right)} T B \gamma_T M} \qquad \text{where } B = 8 \log\left(\frac{2T^2M}{\delta}\right)$$

Λ : lipschitz constant

- Combinatorial GP bandits for advertising

Every subcampaign is modelled by means of a GP with the constraint that at most a pair of (bid, budget) can be chosen



constraint: cumulative budget < value

1. sample for every pair the reward
2. solve the optimization problem

- Regret

$$R_T \leq \sqrt{\frac{2\Lambda^2}{\log\left(1+\frac{1}{\sigma^2}\right)} C T B \sum_{k=1}^{C} \gamma_{k,T}} \qquad \text{where } B = 8 \log\left(\frac{2T^2MC}{\delta}\right)$$

C : number of GP

SOCIAL

Information cascades

We focus on scenarios in which a social influence phenomenon is due to information effects.

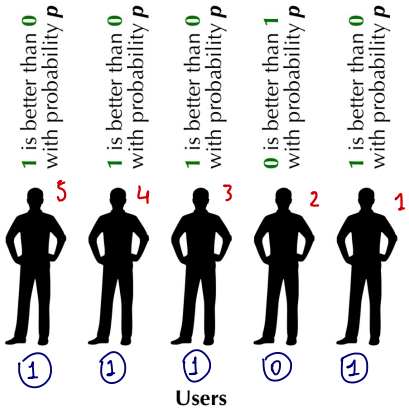
- Scenario

An user has to choose among different proposals (restaurants), but he doesn't have any information about the quality, so he looks at other users' recommendations
→ IDEA: the user builds his beliefs looking at others.

- Model

we are given a set of users and every user can make one action among a given set of actions

1. users don't know which action is the best, at the beginning all actions are equal
2. Each users at the beginning receives a signal saying which is the best action with a probability $p > 1/2$
3. A part from the first user, all users can observe the taken actions by previous users and they update their belief accordingly



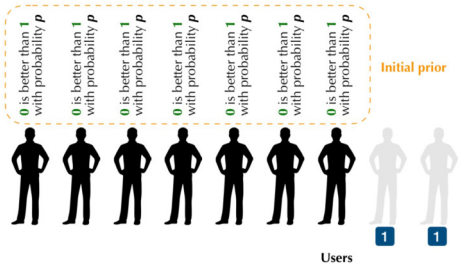
user 1: his belief depends only on the signal he received
→ he chooses 1

user 2: he sees that user 1 chose 1 but the signal he received is that 0 is better than 1. The two beliefs nullify: ties are broken in favour of the received signal
→ he chooses 0

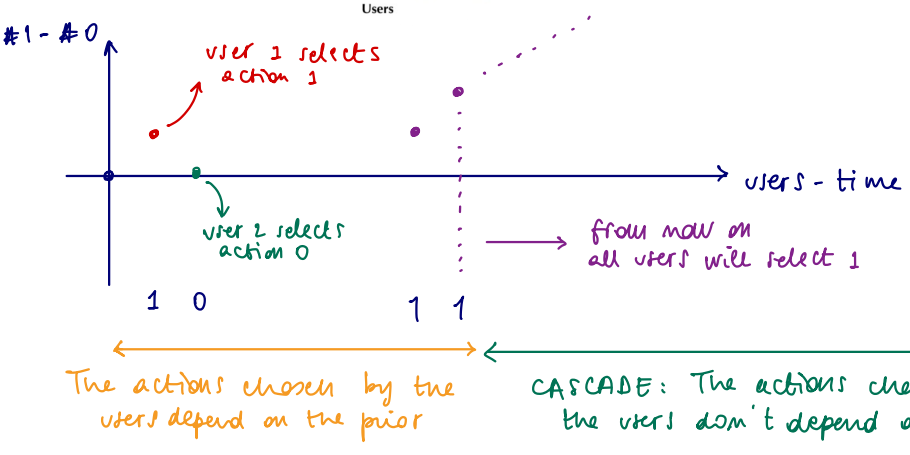
⇒ all users update their beliefs

The belief of users depends on the number of users that in the past selected action 1 or 0
⇒ it depends on the difference between the number of observed 1 and 0.

- cascade activation



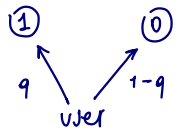
if the first two users select action 1, then they generate an information cascade because all other users will select 1 regardless their signal (prior information)



$|\#1 - \#0| \leq 1$

- Cascade activation probability

Probability to have a cascade when we have many users



The probability of a cascade is larger than the probability to have 3 consecutive 1 signals or 0 signals

$$\bar{p} = q^3 + (1-q)^3$$

q^3: prob. to have 3 1-signals; $(1-q)^3$: prob. to have 3 0-signals

$\Rightarrow$ if the time horizon $T \to +\infty$ we have the certainty to reach a cascade

$$1 - (1-\bar{p})^{T/3} \xrightarrow[T\to\infty]{} 1$$

* Mathematics

PRIOR: $\mathbb{P}(1 \succ 0) = \mathbb{P}(0 \succ 1) = 1/2$ same probability for 1 better than 0 and viceversa

prob. 1 is better than 0 — prob. 0 is better than 1

SIGNAL: $\mathbb{P}(\{1\} \mid 1 \succ 0) = \mathbb{P}(\{0\} \mid 0 \succ 1) = p > 1/2$

prob. user 2 receives the signal "1 is better than 0" knowing that user 1 chose 1

$$\mathbb{P}(\{0\} \mid 1 \succ 0) = \mathbb{P}(\{1\} \mid 0 \succ 1) = 1-p < 1/2$$

prob. user 2 receives the signal "0 is better than 1" knowing that user 1 chose 1

Bayesian Rule: $\mathbb{P}(1 \succ 0 \mid \{1\}) = \dfrac{\mathbb{P}(\{1\} \mid 1 \succ 0)\,\mathbb{P}(1 \succ 0)}{\mathbb{P}(\{1\} \mid 1 \succ 0)\,\mathbb{P}(1 \succ 0) + \mathbb{P}(\{1\} \mid 0 \succ 1)\,\mathbb{P}(0 \succ 1)} = p$

prob. action 1 is better than 0 knowing user 1 chose 1

$$\mathbb{P}(\{0,1\} \mid 1 \succ 0) = \mathbb{P}(\{0\} \mid 1 \succ 0) \cdot \mathbb{P}(\{1\} \mid 1 \succ 0) = (1-p)p$$

$$\mathbb{P}(1 \succ 0 \mid \{0,1\}) = 1/2 \rightarrow \text{the two signals nullify}$$

prob. action 1 is better than 0 knowing user 1 chose 1 and user 2 chose 0

$$\Rightarrow \mathbb{P}(1 \succ 0 \mid (n_0, n_1)) = \frac{1}{1 + \dfrac{(1-p)^{n_1-n_0}}{p^{n_1-n_0}}} > 1/2 \quad \text{if } n_1 - n_0 > 0$$

$\Rightarrow$ if $\Delta \to \infty$ then $\mathbb{P} \to 1$, so many users will select 1

- Model extentions
  1. Multiple actions
  2. Different priors
  3. Non uniform signals
  4. Users with different preferences

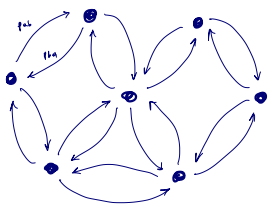
→ a manipulator could always start a cascade by introducing fake reviews

↳ solution: rate the reviews to block adversarial cascades

DIRECT EFFECTS

We consider a social influence phenomenon in which users collect information on possible auctions and the action of a user depends on previous users' action

- Social Network



node = user, edge = connection among users

→ direct graph: the way a user affects the other may be different and it's weighted because it describes how a node affects the other node

$p_{a,b} \in [0,1]$: probability that a node a affects node b

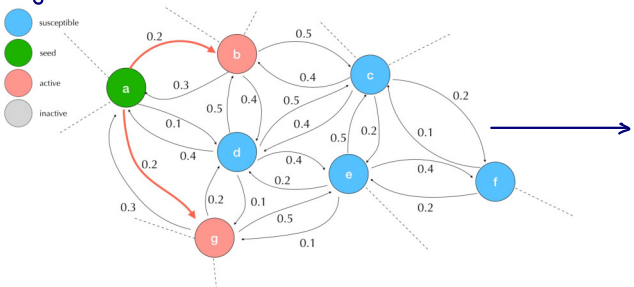
- Model
  1. Every node can be of different states {susceptible, active, inactive}
     - susceptible: can be activated by the influence of some neighbourhood
     - active: is activated by a neighbourhood
     - inactive: has been previously activated
  2. Time is discrete
  3. If a node a becomes active at time t, at time t+1 it can activate neighbour b with probability $p_{a,b}$
  4. If node b is not activated by node a, then node a cannot activate node b in the future
  5. If a node activates at time t, at time t+1 it becomes inactive

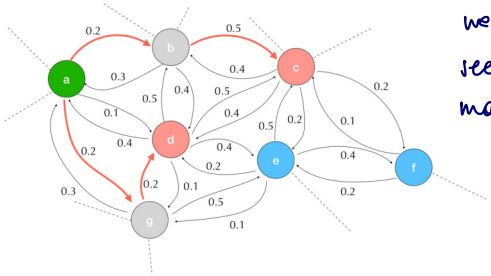
- Seeding

Def seeds are a subset of nodes which are able to generate a cascade of influence to the other nodes

⇒ the goal of a social influencer is to buy seeds in order to influence (generate a cascade)

- Dynamic diffusion

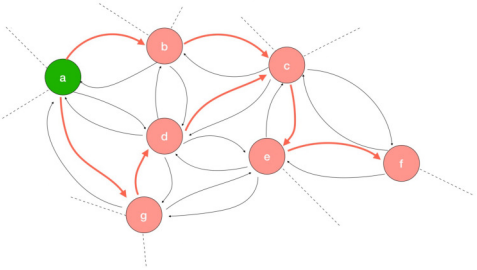




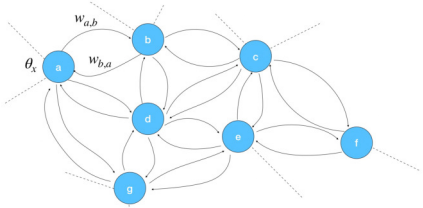
we need to select the seeds properly in order to make the whole network active

- Live-edge graph

Def a live edge graph is the subgraph composed by all nodes that can be reached by the seed (so which can be active in the future)



- Linear threshold

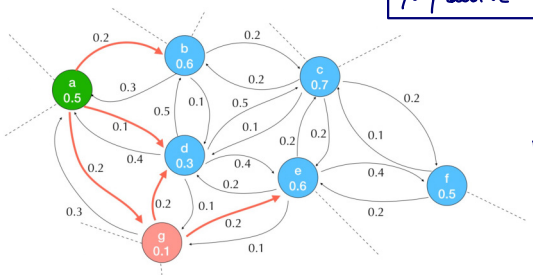


$\vartheta_x \in [0,1]$ threshold

$\begin{cases} w_{ij} \in [0,1] \\ \sum_i w_{ij} < 1 \end{cases}$

- activation
  1. once a node is active, it has the possibility to influence the neighbours in every time in the future
  2. A node activates if $\sum_{y:\, y \text{ active}} w_{yx} \geq \vartheta_x$

the sum of weights of edges coming to that node from active nodes is larger than the threshold



d: $0.1 + 0.2 \geq 0.3$ → activates

b: $0.2 \geq 0.6$ → NO

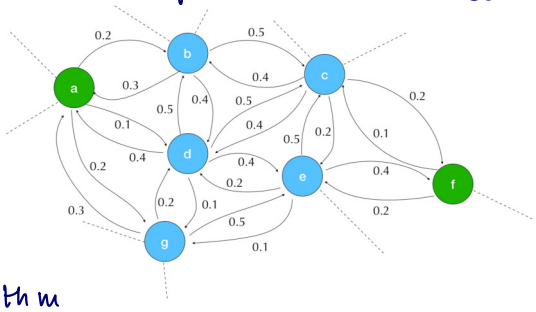
e: $0.2 \geq 0.6$ → NO

- extentions
  - Contextual problems: the probabilities associated with the edges depend on messages characterized by features
  - Negative effects: active/inactive nodes can change the state, returning to susceptible, due to negative influence of another node
    - active → linear models
    - inactive → cascades

INFLUENCE MAXIMIZATION

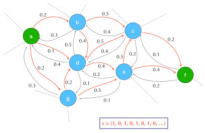
- influence maximization problem (IMP)
  1. Input
     - Network
     - Influence probabilities
     - Budget (in terms of number of nodes we have to buy)
  2. Action: select a subset of nodes (seeds) to select simultaneously with cardinality not larger than the budget

→ GOAL: find the best subset of nodes subject to budget constraints to maximize the expected number of nodes that have been activated during the cascade

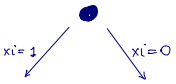


given these seeds we want to know the probability to activate a node

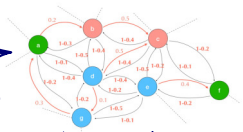
- algorithm



  1. Call E the set of m edges $\{e_1 \dots e_m\}$
  2. Call $\underline{x} = (x_1 \dots x_m)$ where $x_i = 1$ if edge e_i is present, 0 otherwise
  3. Enumerate every $\underline{x}$ (it's a live edge graph)
     ↳ the enumeration can be done by using a tree with branching factor 2



  4. $\forall \underline{x}$ find the set of nodes which have been activated during the cascade
  5. calculate the probability of each $\underline{x}$ as the product of all probabilities



  6. Then compute the activation probability of a node as the sum of the probabilities of $\underline{x}$ with a path connecting the node with some seed

* Computational consideration:
the computation of the activation probabilities associated with each node is impratical (exponential time in the number of edges)
↳ solution: approximate these probabilities with Monte Carlo sampling (polynomial time)

Monte Carlo sampling
1. For each node i assign $z_i = 0$ (number of times node i is activated during MC simulations)
2. Generate randomly a live-edge graph according to the probability of every edge (*)
3. For each node that is active assign $z_i = z_i + 1$
   → repeat 2. and 3. for k times
4. For each node return z_i/k → frequency of a node which has been activated in the simulations

(*) some live edge graphs may not be present, while others too many times

- How many repetitions?
  With probability of at least $1-\delta$, the estimated activation probability of every node is subject to an additive error $\pm \varepsilon m$ when the number of repetitions is

$$R = \mathcal{O}\left(\frac{1}{\varepsilon^2} \log(|S|) \log\left(\frac{1}{\delta}\right)\right)$$

- IM exact solution
  - The problem cannot be solved exactly in polynomial time (even with MonteCarlo)
    ↳ every exact algorithm requires exponential time
  - The basic exact algorithm enumerates all the possible subsets of k nodes (seeds) and for each subset it evaluates the expected number of active nodes

- Approximating IM

A simple greedy algorithm provides an approximation of $1 - \frac{1}{e} \simeq 0.63$

idea of algorithm: 1. For each node that is not a seed, we need to evaluate the marginal increase in the objective function if that node is not a seed
2. Select the node for which the marginal increase is maximized and add it to the set of seeds

→ Considerations:

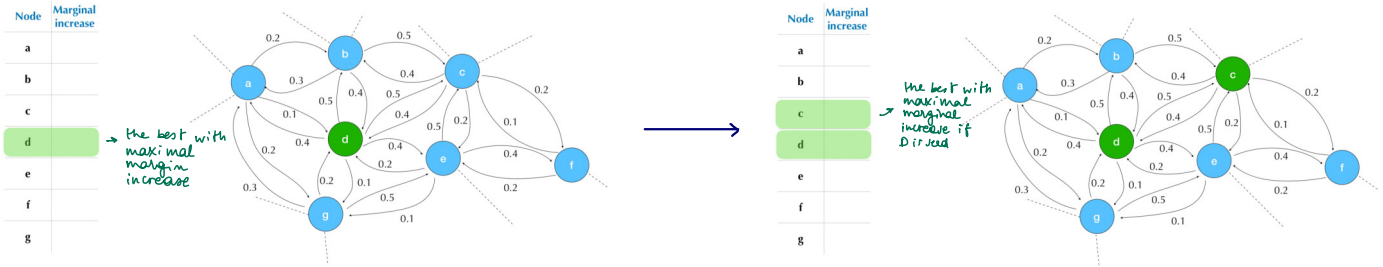
1. The approximation bound follows from the sub-modularity property:

$$\forall X \subseteq Y,\ x \notin Y : f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y)$$

adding a seed to a set of seeds gives a marginal increase in the utility function larger than adding the same node to a strictly larger set of seeds

2. The greedy algorithm works incrementally (add seed by seed) in order to decide the set of seeds, not to simulate the fact that nodes become a seed

Example



- Model generalization

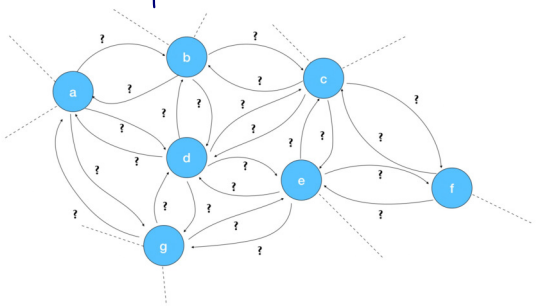
1. Each seed requires a different cost in terms of monetary resources
2. The budget is expressed in monetary resources
3. Every node can have a different weight

→ GOAL: the greedy algorithm maximizes the marginal increase in utility wrt the cost of the seed

LEARNING ISSUES IN INFLUENCE MAXIMIZATION

Some information in social network is not known and we need to learn it in online session

- Uncertainty in social networks



the network is known
weights are unknown

→ in order to learn the weights we can repeat the problem many times

activated nodes ~ cascade termination

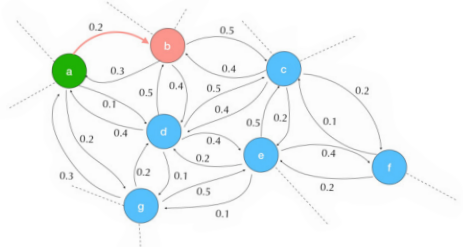
- Three learning scenarios

① Fully observable edges

~ we can observe the fully activation of all edges ~

For each edge in the network we can observe whether the edge activated or not (we can see the node which has been activated and the node which activated)

⇒ we observe that node b makes a retweet of a tweet of node a



Example: standard combinatorial MAB problem

- every edge $\sim Be(p)$ with p unknown
- For every edge we can draw a sample from a Beta with TS or use an upper confidence bound with UCB1
- Once we have a value for each edge we use a greedy algorithm to find the optimal solution (or an approximation)

② Partially observable edges

~ we can observe the activation of a small portion of the edges ~
we want to predict all possible information of edges,
but we can't collect a huge number of samples.
since the cascade due to a seed involves a small ratio of nodes,
we do many repetitions of the problem.

⇒ idea: reuse information collected in a part of the network also for other parts

1. exploit the information on no observable edges to infer the probabilities of non-observable edges
2. it implies a constraint on the probability of edges

Def
- features $\mathcal{Y} = \{1 .. L\}$
- values of the features: $x_{ij} \in [0,1]$ where $x_{ij} = 1$ means that feature j is very important for edge i
- parameters of the features: $\vartheta_j \in [0,1]$
- linear dependency $p_i = \sum_{j=1}^{L} \vartheta_j x_{ij}$ → probability of an edge

Example: If the features are the interest of users, a value of 1 for a feature means that both users share that interest

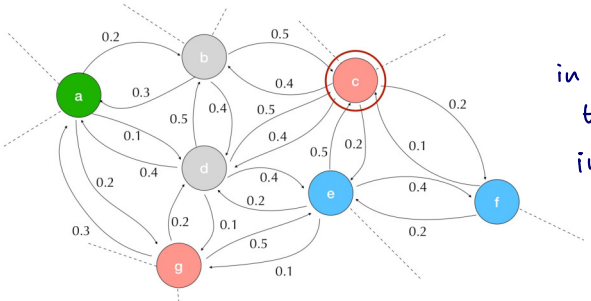
- parameters of features are estimated by using linear regression (features' value is known)
- Given the values of the probabilities a term is added to obtain an upper confidence (UCB1 idea)
- The optimization is performed by using the greedy algorithm

③ Non-observable edges

~ we can observe only the activation of the nodes ~

⇒ we only observe the action of the nodes without observing the activation of the edges

Time	Active
0	a
1	b, d
2	c, g
3	
4	



in order to solve the problem we need to observe the evolution of the cascade by providing additional information by taking trace of the active nodes

→ when a node could have been activated by multiple nodes, maximum likelihood techniques are used to estimate which is the edge that most likely activated that node or alternative frequentist techniques should be used