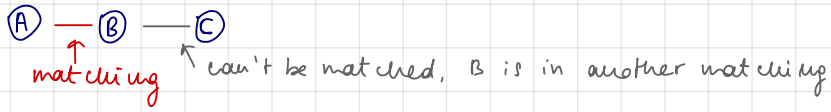
ANSWERS

MATCHING in ONLINE ENVIRONMENTS

Question 1

- **Mathematical formulation of a matching problem**

we are given a graph (set of edges and links) and the goal is to find a matching.
a MATCHING is a subset of edges of the graph such that no pair of edges share vertices
→ every vertex can occur once in a matching

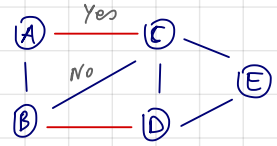


* maximal matching → matching that cannot be enlarged
* maximum matching → matching with largest cardinality (num. of edges)
* perfect matching → matching where all nodes are matched (it may no exist)

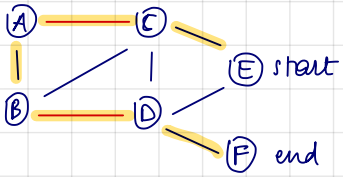
⇒ GOAL: find a maximum matching

- **Describe what an alternating and augmenting paths are**

* alternating path → path such that edges in the matching and edges of the complementary set (edges not in the matching) alternate



* augmenting path → alternating path starting and ending in unmatched vertices



- **Describe the alternating path algorithms for bipartite and arbitrary graphs**

The idea of the alternating-path algorithm is to find an augmenting path, do the complementary in order to find a new alternating path whose cardinality is strictly larger than the cardinality of the initial path
→ we find a maximum matching

- Alternating-path algorithm works iteratively until no augmenting path exists
  - look for an augmenting path
  - make the complementary

1. Consider every unmatched vertex (as root of a subtree)
2. Make a breadth-first search starting from every unmatched vertex
3. For every unexplored path which is not in the matching, 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 (even/odd)
5. If the path in the expansion contains some visited vertices then stop the branch and take a decision according to the specific case.

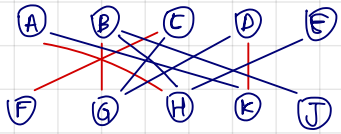
   case 1 If the expanding even node is directly connected to an even node (previously labeled) we found an augmenting path

   case 2 If the expanding even node is directly connected to an odd node (previously labeled) then stop the search

   case 3 (only for arbitrary graph) If the expanding even node is directly connected to a vertex previously labeled as even and belonging to the same subtree then we found a blossom.

Question 2 and 3

- Apply the alternating path algorithm to a bipartite graph

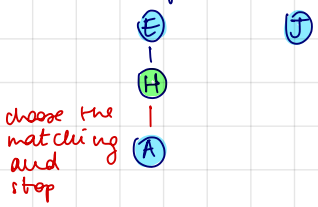


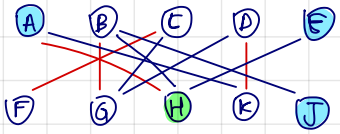
$M = \{(A,H), (C,F), (D,K), (B,G)\}$ links in the matching

$U = \{(E,J)\}$ unmatched nodes → even

$J = \{E, J\}$ nodes to be visited

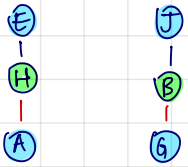
1° step: Expand E

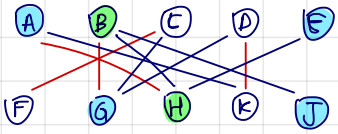




$J = \{J, A\}$

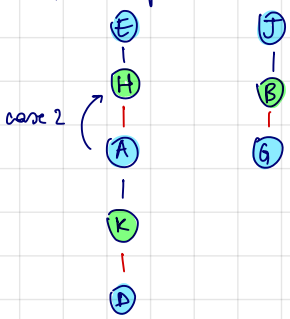
2° step: Expand J

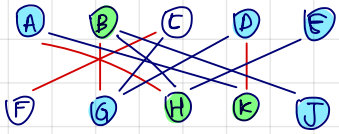




$J = \{A, G\}$

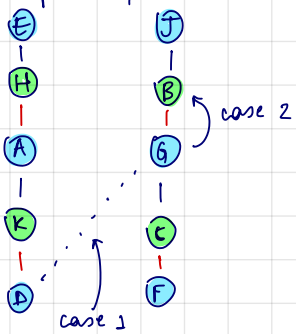
3° step: expand A

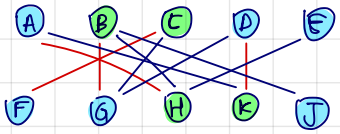




$J = \{G, D\}$

4° step: expand G





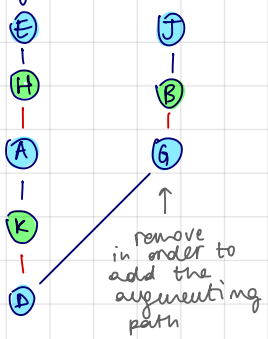
G - D augmenting path

$J = \{D, F\}$

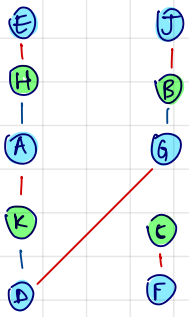
5° step: expand D D → K stop,
D → G already visited

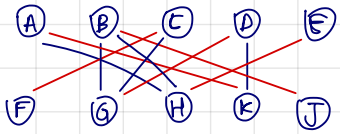
6° step: expand F F → C already visited

so we find

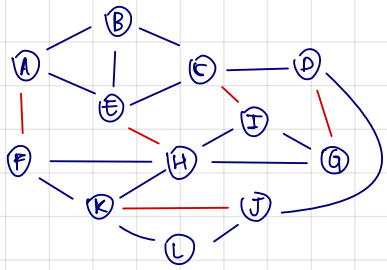


complementary





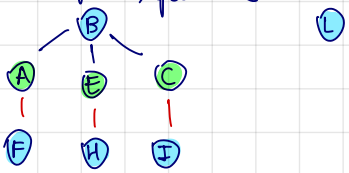
- Apply the alternating path for arbitrary graph

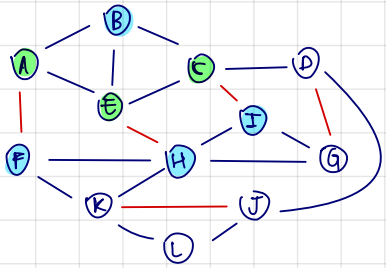


$M = \{(A,F), (E,H), (C,I), (D,G), (K,J)\}$

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

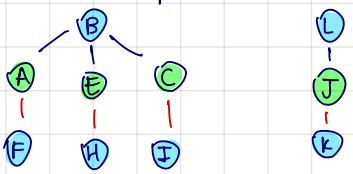
1° step: expand B

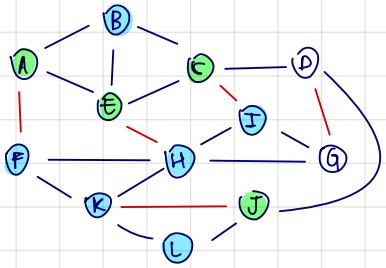




$\mathcal{Y} = \{L, F, H, I\}$

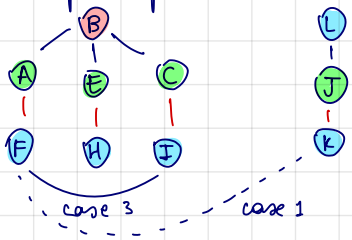
2° step: expand L





$\mathcal{Y} = \{F, H, I, K\}$

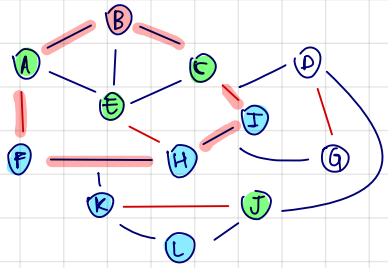
3° step: expand F



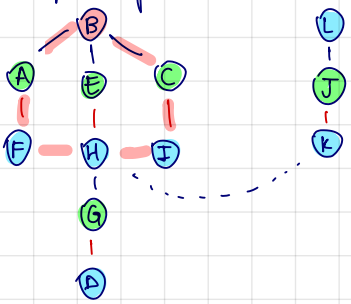
B → vertex of a blossom

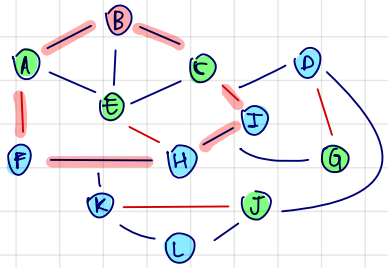
F − K → augmenting path

$\mathcal{Y} = \{H, I, K\}$



4° step: expand H

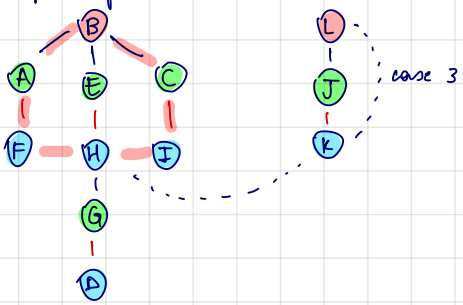




$\mathcal{Y} = \{I, K, D\}$

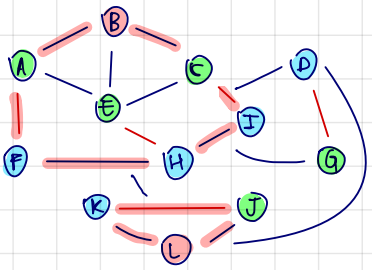
5° step: expand I → close

6° step: expand k

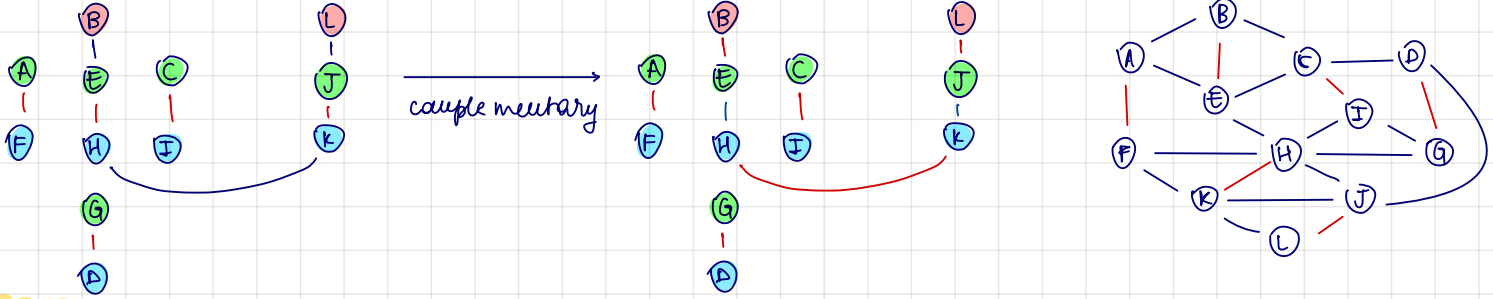


L → vertex of a blossom

$\mathcal{Y} = \{D\}$



7° step: expand D → stop



Question 4

- **Define what a combinatorial bandit problem is**

Given a set of candidates (arms = prices) and an unknown expected reward for each arm, a combinatorial bandit problem is a problem in which we have combinatorial constraints, so a subset of feasible arms, whose arms are pulled simultaneously. These arms are the solution of a combinatorial problem.

We define: superarm → collection of candidates ($\underline{a}$)

feasible superarm → superarm satisfying the (combinatorial) constraint $C(\underline{a}) = 0$

reward of a superarm → sum of the reward included in a superarm

→ Goal: maximize the cumulative in time expected reward

- **Describe how a matching problem can be formulated as a combinatorial bandit problem**

→ In weighted matching problems the value associated to an edge is unknown, and we need to estimate it by formulating the problem as a combinatorial bandit problem

A matching problem can be formulated as a combinatorial bandit problem as follows:

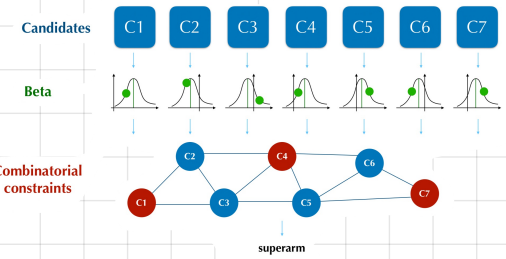
- a superarm is a matching (a set of edges)
- arms are nodes of the graph

- **Describe the combinatorial TS algorithm**

Assume that the random variable of every arm is a Bernoulli in $\{0,1\}$ with unknown mean

1. At every time t and for every arm a
   $\tilde{\vartheta}_a \leftarrow$ sample $(\mathbb{P}(\mu_a = \vartheta_a))$
2. At every time t, play superarm $\underline{a}_t$ (solve the combinatorial problem)
   $\underline{a}_t \leftarrow \underset{C(\underline{a})=0}{\operatorname{argmax}} \left\{ \sum_{a \in \underline{a}} \tilde{\vartheta}_a \right\}$
3. Update the Beta distribution
   $(\alpha_{a_t,t}, \beta_{a_t,t}) \leftarrow (\alpha_{a_t,t}, \beta_{a_t,t}) + (X_{a_t,t};\ 1 - X_{a_t,t})$

X_a: random variable of the reward of arm a
μ_a: expected value of X_a
$x_{a,t}$: realization of X_a at time t
α, β: parameters of Beta distribution



- each arm has a beta distribution
- we drawn a sample
- we solve the associated combinatorial problem
- find the superarm $C(\underline{a}) = \{C_1, C_4, C_7\}$
- pull the arms simultaneously
- update the rewards of all these arms

- **Define the regret in case of combinatorial bandit problem and compare with non combinatorial bandit problems**

* Regret of combinatorial bandit problems $R_T(TS) \in \vartheta\left(\frac{|A| \log T}{\Delta_{min}}\right)$
  - increases linearly as the number of arms $|A|$ increases
  - increases logarithmically as T increases
* Regret of non combinatorial bandit problems is a lower bound which
  - increases linearly as the number of arms $|A|$
  - increases logarithmically as T increases

⇒ combinatorial bandit problems have in general a lower regret → better!

Question 5

- **Define what an online matching problem is and how it distinguishes from the non-online case**

A matching problem is said to be "online" if the graph is discovered during time (also if it's weighted) (in bipartite graph one side of nodes is known a priori)

At each round a single node enters the problem and its edges are discovered and we have to decide to match or not the new node.

A non-online matching problem is a completely known graph a priori

- **Define the competitive factor of an online problem**

online matchings are inefficient, because the optimal choice of a possible matching depends on what happens in the future. In order to understand if the algorithm is doing the best, we compare the number of matchings with the number of matchings of the clairvoyant solution, where the clairvoyant solution is the best solution we have with complete information about the problem

$$\alpha_A = \min_P \frac{\Gamma_A(P)}{\Gamma^*(P)}$$

α_A: competitive factor of the algorithm

$\Gamma_A(P)$: value provided by the online algorithm

$\Gamma^*(P)$: value provided by the clairvoyant algorithm

it can't be larger than 1 because the clairvoyant solution is the best

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

- **Show that a basic online matching problem does not admit any deterministic algorithm with competitive factor larger than 1/2**

Match a node with an arbitrary unmatched node → algorithm has $\alpha_A = 1/2$ (the best we can do)

The proof relies on 2 steps:

1. The cardinality of any maximal matching is at least 1/2 of the cardinality of the maximum matching

   Assume all weights $w_i = 1$ and a bipartite graph.

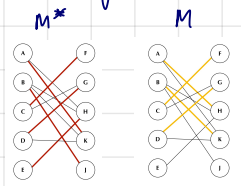
   A. Given M^* (maximum matching) and M (maximal matching)

   B. $\forall$ edge $e = (u, v)$ in M^*, either u or v is matched in M, otherwise M is not maximal (we can add e)

   C. So the number of matched nodes in M is not smaller than $|M^*|$ = # edges

   D. The number of matched nodes in M^* is $2|M^*|$

   E. The number of edges in M is at least 1/2 the number of edges in M^* : $|M| \geq 1/2 |M^*|$



2. This greedy algorithm always find a maximal matching

   A. Given M (maximal)

   B. $\forall$ edge $e = (u, v)$ not in M, consider the round in which v arrived

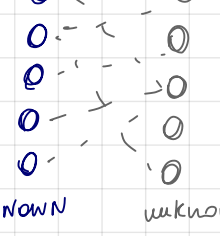
   C. { v has been matched to some other node instead of u
      { v is not matched since all other nodes are matched

   D. In both cases (u, v) cannot be added to M, not being feasible

- **Describe a greedy algorithm for a basic online matching problem with $\alpha_A = \frac{e-1}{e}$**

1. generate randomly an ordering over the nodes that are initially available
2. match a node with the first unmatched node in the ordering that we generated initially

   → competitive factor = $\frac{e-1}{e} > 1/2$



~ Randomized algorithm (Ranking) ~

Question 7

- **Show when both sides of a bipartite matching problem enter dynamically, there not online algorithm with strictly positive competitive factor**

There are some istances where the competitive factor is 0, so there's no deterministic algorithm returning any strictly competitive factor

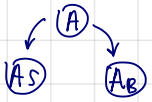
1. A enters the problem (A)
2. B enters the problem : do we match A and B ?
   - ↳ match: (A) —1— (B) ···>>1··· (C) $\quad \frac{APT}{OPT} = \frac{1}{>>1} \to 0$ $\quad$ match A-B / best solution B-C
   - ↳ don't match (B) $\quad \frac{APT}{OPT} = \frac{0}{1} = 0$ $\quad$ don't match / best solution A-B

- **Describe the functioning of the Postponed Dynamic Deferred Acceptance and report an example**

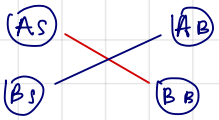
There's a randomized algorithm with an expected competitive factor of 1/4 when the waiting time of every node is the same (k)

→ every time a node arrives, it stays the same number of rounds and it leaves (when it's critical or at its deadline)

1. When the node arrives : the algorithm generates the node as seller and buyer



2. If a node has to leave :
   a. we match it
   b. it leaves the problem



A. Select A as seller ($P = 1/2$)
the matched B is labelled as buyer
match A_S with B_B because it's the best matching
remove virtual nodes that are different from labelling

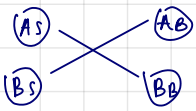
B. Select A as buyer ($P = 1/2$)
the matched B is labelled as seller
since A_B was optimally matched with another node, then A leaves the problem without being matched

→ worst case example (A) —1— (B) —>>1— (C)

1. A enters the problem :
2. B enters the problem :

3. A reached its deadline → we have to match it with B or it leaves the problem

A. A is seller → B is buyer (AS) —1— (BB)
C enters the problem but it's unmatched



B. A is buyer → B is seller (AB) (BS) ⇒ (BS) (BB)
A is not matched with B and it leaves the problem → matching of value 1

↳ 3. C enters the problem
B is seller → C is buyer (BS) —>>1— (CB) → matching of value >1

→ competitive factor is 1/2 in both cases