Online algorithms for dummies

Nicolas Bousquet

Journées CALAMAR

(Journées Combinatoires des Alpes, des Littoraux Atlantique et Méditéranéen, d'Auvergne et du Rhône)

January 2024

Today : online algorithms

- Talk 1 : Introduction to online algorithms - Nicolas Bousquet (LIRIS).
- Talk 2 : Online algorithms with predictions - Bertrand Simon (IN2P3).
- Talk 3 : Online edge coloring - Clément Legrand-Duchesne (LaBRI).

What is an online algorithm ?

- Input arrives sequentially over time (arrival order).
- Decisions must be taken without the knowledge of the future input.
- Decisions are irrevocable.

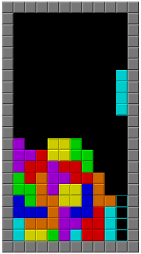


Illustration : Graph coloring on trees

Greedy Algorithm : Give to each vertex the smallest possible color.

Illustration : Graph coloring on trees

Greedy Algorithm : Give to each vertex the smallest possible color.

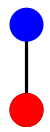


Illustration : Graph coloring on trees

Greedy Algorithm : Give to each vertex the smallest possible color.

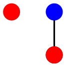


Illustration : Graph coloring on trees

Greedy Algorithm : Give to each vertex the smallest possible color.

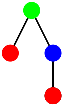


Illustration : Graph coloring on trees

Greedy Algorithm : Give to each vertex the smallest possible color.

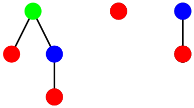


Illustration : Graph coloring on trees

Greedy Algorithm : Give to each vertex the smallest possible color.

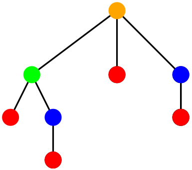
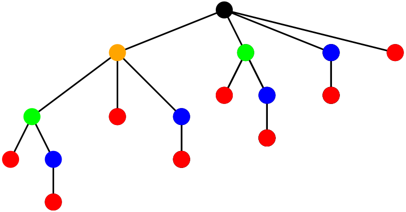


Illustration : Graph coloring on trees

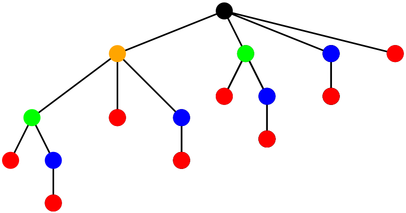
Greedy Algorithm : Give to each vertex the smallest possible color.



$\rightarrow$ This algorithm may output a $(\Delta + 1)$ coloring. (while there exists a 2-coloring)

Illustration : Graph coloring on trees

Greedy Algorithm : Give to each vertex the smallest possible color.



$\rightarrow$ This algorithm may output a $(\Delta + 1)$ coloring. (while there exists a 2-coloring)

Typical question : Can we find an algorithm that "approximates" the quality of the best offline algorithm ?

Types of adversaries

You shall choose an instance of that type.

e.g. "a graph", "a planar graph", "an interval graph"

Types of adversaries

You shall choose an instance of that type.

e.g. "a graph", "a planar graph", "an interval graph"

This will be my (deterministic / randomized) algorithm.

Types of adversaries

You shall choose an instance of that type.

e.g. "a graph", "a planar graph", "an interval graph"

This will be my (deterministic / randomized) algorithm.

Héhéhé ! I send you the worst possible instance (and ordering) ! I am evil !

Types of adversaries

You shall choose an instance of that type.

e.g. "a graph", "a planar graph", "an interval graph"

This will be my (deterministic / randomized) algorithm. 

Héhéhé ! I send you the worst possible instance (and ordering) ! I am evil !

Oblivious Adversary. Knows the algorithm and choose -**once for all**- the instance. (weaker adversary)

Types of adversaries

You shall choose an instance of that type.

e.g. “a graph”, “a planar graph”, “an interval graph”

This will be my (deterministic / randomized) algorithm.

Héhéhé ! I will send you the worst possible **1st** vertex of an instance and I’ll see next... I am super evil !

Oblivious Adversary. Knows the algorithm and choose -**once for all**- the instance. (weaker adversary)

Types of adversaries

You shall choose an instance of that type.

e.g. “a graph”, “a planar graph”, “an interval graph”

This will be my (deterministic / randomized) algorithm.

Héhéhé ! I will send you the worst possible **1st** vertex of an instance and I’ll see next… I am super evil !

This is my decision for the 1st vertex.

Oblivious Adversary. Knows the algorithm and choose -**once for all**- the instance. (weaker adversary)

Types of adversaries

You shall choose an instance of that type.

e.g. "a graph", "a planar graph", "an interval graph"

This will be my (deterministic / randomized) algorithm.

Héhéhé ! Now, I will send you the worst possible **2nd** vertex of an instance and I'll see next... I am super evil !

This is my decision for the 1st vertex.

Oblivious Adversary. Knows the algorithm and choose -**once for all**- the instance. (weaker adversary)

Types of adversaries

You shall choose an instance of that type.

e.g. “a graph”, “a planar graph”, “an interval graph”

This will be my (deterministic / randomized) algorithm.

Héhéhé ! Now, I will send you the worst possible **2nd** vertex of an instance and I'll see next... I am super evil !

This is my decision for the 2nd vertex.

Oblivious Adversary. Knows the algorithm and choose -**once for all**- the instance. (weaker adversary)

Types of adversaries

You shall choose an instance of that type.

e.g. “a graph”, “a planar graph”, “an interval graph”

This will be my (deterministic / randomized) algorithm.

Héhéhé ! Now, I will send you the worst possible **2nd** vertex of an instance and I’ll see next... I am super evil !

This is my decision for the 2nd vertex.

Oblivious Adversary. Knows the algorithm and choose -**once for all**- the instance. (weaker adversary)

Adaptive adversary. Knows the algorithm and all the choices performed so far and chooses the **next action**. (stronger adversary)

Two levels of such adversaries

Deterministic vs randomized

Two types of online algorithms : deterministic or randomized !

Deterministic vs randomized

Two types of online algorithms : deterministic or randomized !

Remark :
Oblivious and adaptive adversaries are equivalent for deterministic algorithms.

Performance of online algorithms

Performace of an online algorithm Given a maximization problem, I an instance, an algorithm is :

- α-competitive the algorithm outputs a solution of (expected) size $\geq \alpha \cdot OPT(I) + c$ where $OPT(I)$ denotes the size of the optimal solution.

Performance of online algorithms

Performace of an online algorithm Given a maximization problem, I an instance, an algorithm is :

- α-competitive the algorithm outputs a solution of (expected) size $\geq \alpha \cdot OPT(I) + c$ where $OPT(I)$ denotes the size of the optimal solution.
- α-strictly competitive the algorithm outputs a solution of (expected) size $\geq \alpha \cdot OPT(I)$ where $OPT(I)$ denotes the size of the optimal solution.

Performance of online algorithms

Performace of an online algorithm Given a maximization problem, I an instance, an algorithm is :

- α-competitive the algorithm outputs a solution of (expected) size $\geq \alpha \cdot OPT(I) + c$ where $OPT(I)$ denotes the size of the optimal solution.
- α-strictly competitive the algorithm outputs a solution of (expected) size $\geq \alpha \cdot OPT(I)$ where $OPT(I)$ denotes the size of the optimal solution.

Remark :

- $\alpha \leq 1$ and if $\alpha = 1$ we have an almost optimal algorithm.
- For a minimization function we can twist the definition
- For a deterministic algorithm, we are just looking for the worst instance. For randomized algorithms, we look for the worst possible expected size.

Ski rental

We go to a ski station x days where x is unknown.

Each day, one can either :

- Buy a pair of skis for B euros (forever) or,

Ski rental

We go to a ski station x days where x is unknown.

Each day, one can either :

- Buy a pair of skis for B euros (forever) or,
- Rent a pair of skis for 1 euro per day.

Ski rental

We go to a ski station x days where x is unknown.

Each day, one can either :

- Buy a pair of skis for B euros (forever) or,
- Rent a pair of skis for 1 euro per day.
- Each day, if we haven't yet bought a pair of ski, we can buy a pair.

Ski rental

We go to a ski station x days where x is unknown.
Each day, one can either :

- Buy a pair of skis for B euros (forever) or,
- Rent a pair of skis for 1 euro per day.
- Each day, if we haven't yet bought a pair of ski, we can buy a pair.

Buy a pair of ski or not.

Decide when the ski trip is over.

Ski rental

We go to a ski station x days where x is unknown.
Each day, one can either :

• Buy a pair of skis for B euros (forever) or,
• Rent a pair of skis for 1 euro per day.
• Each day, if we haven't yet bought a pair of ski, we can buy a pair.

Buy a pair of ski or not.

Decide when the ski trip is over.

Algorithm 1 : Buy a pair immediately.

Ski rental

We go to a ski station x days where x is unknown.
Each day, one can either :

- Buy a pair of skis for B euros (forever) or,
- Rent a pair of skis for 1 euro per day.
- Each day, if we haven't yet bought a pair of ski, we can buy a pair.

Buy a pair of ski or not.

Decide when the ski trip is over.

Algorithm 1 : Buy a pair immediately.
Opponent strategy : Stop immediately after day 1.

Ski rental

We go to a ski station x days where x is unknown.
Each day, one can either :

- Buy a pair of skis for B euros (forever) or,
- Rent a pair of skis for 1 euro per day.
- Each day, if we haven't yet bought a pair of ski, we can buy a pair.

Buy a pair of ski or not.

Decide when the ski trip is over.

Algorithm 1 : Buy a pair immediately.
Opponent strategy : Stop immediately after day 1.
Competitive ratio : $\frac{B}{1}$. $\rightarrow$ Bad when B is large...

Ski rental

We go to a ski station x days where x is unknown.
Each day, one can either :

- Buy a pair of skis for B euros (forever) or,
- Rent a pair of skis for 1 euro per day.
- Each day, if we haven't yet bought a pair of ski, we can buy a pair.

 Buy a pair of ski or not.

 Decide when the ski trip is over.

Algorithm 1 : Buy a pair immediately.
Opponent strategy : Stop immediately after day 1.
Competitive ratio : $\frac{B}{1}$. $\rightarrow$ Bad when B is large...

Algorithm 2 : Always rent a pair of ski.

Ski rental

We go to a ski station x days where x is unknown.
Each day, one can either :

- Buy a pair of skis for B euros (forever) or,
- Rent a pair of skis for 1 euro per day.
- Each day, if we haven't yet bought a pair of ski, we can buy a pair.

 Buy a pair of ski or not.

 Decide when the ski trip is over.

Algorithm 1 : Buy a pair immediately.
Opponent strategy : Stop immediately after day 1.
Competitive ratio : $\frac{B}{1}$. $\rightarrow$ Bad when B is large...

Algorithm 2 : Always rent a pair of ski.
Opponent strategy : Decide to stay at the ski station forever.

Ski rental

We go to a ski station x days where x is unknown.
Each day, one can either :

- Buy a pair of skis for B euros (forever) or,
- Rent a pair of skis for 1 euro per day.
- Each day, if we haven't yet bought a pair of ski, we can buy a pair.

 Buy a pair of ski or not.

 Decide when the ski trip is over.

Algorithm 1 : Buy a pair immediately.
Opponent strategy : Stop immediately after day 1.
Competitive ratio : $\frac{B}{1}$. $\rightarrow$ Bad when B is large...

Algorithm 2 : Always rent a pair of ski.
Opponent strategy : Decide to stay at the ski station forever.
Competitive ratio : $\frac{n}{B} \rightarrow +\infty$ when n tends to infinity.

Compromise - Break-even algorithm

- The first $B - 1$ days, we rent skis.
- The B-th day, we buy the skis.

Compromise - Break-even algorithm

- The first $B - 1$ days, we rent skis.
- The B-th day, we buy the skis.

Theorem

The break-even algorithm is $(2 - \frac{1}{B})$-competitive

Compromise - Break-even algorithm

- The first $B-1$ days, we rent skis.
- The B-th day, we buy the skis.

Theorem

The break-even algorithm is $(2-\frac{1}{B})$-competitive

Proof : Let k the integer where the opponent decide to stop.

- If $k \leq B-1$, the optimal strategy consists in renting and that's what we do.
- If $k \geq B$, the optimal strategy (of cost B) consists in buying skis at day 1. The break-even strategy has cost $2B-1$.

Optimality of the algorithm

Theorem

No deterministic online algorithm has a competitive ration better than $(2 - \frac{1}{B})$.

Optimality of the algorithm

Theorem

No deterministic online algorithm has a competitive ration better than $(2 - \frac{1}{B})$.

Proof :

- Determinist strategy : choose an integer t.
- Opponent strategy : either choose $t' < t$ or $t' = t$.
- Make calculations...

Randomization helps

Model :

Choose a randomized algorithm.

Opponent chooses a date (fixed forever) knowing the random choices we will make but not their output (oblivious adversary)

Randomization helps

Model :


Choose a randomized algorithm.


Opponent chooses a date (fixed forever) knowing the random choices we will make but not their output (oblivious adversary)

Theorem

There exists a $(1-\frac{1}{e})^{-1}$-competitive randomized online algorithm for ski rental.

Randomized algorithm

Randomized algorithm :
Choose a probability distribution p on $\mathbb{N}$ and stop at time i with probability p_i.

Randomized algorithm

Randomized algorithm :
Choose a probability distribution p on $\mathbb{N}$ and stop at time i with probability p_i.
$\Leftrightarrow$ A randomized algorithm is a superposition of (a possibly infinite number of) deterministic algorithm ($\mathcal{A}_i =$ buy ski at time i).
(mixed strategy)

Randomized algorithm

Randomized algorithm :
Choose a probability distribution p on $\mathbb{N}$ and stop at time i with probability p_i.
$\Leftrightarrow$ A randomized algorithm is a superposition of (a possibly infinite number of) deterministic algorithm ($\mathcal{A}_i =$ buy ski at time i).
(mixed strategy)

Dominated strategy :
A deterministic strategy $\mathcal{S}_1$ is dominated by $\mathcal{S}_2$ if for **every** possible choice of t by the adversary, the cost($\mathcal{S}_1$)$\geq$cost($\mathcal{S}_2$).

Randomized algorithm

Randomized algorithm :
Choose a probability distribution p on $\mathbb{N}$ and stop at time i with probability p_i.
$\Leftrightarrow$ A randomized algorithm is a superposition of (a possibly infinite number of) deterministic algorithm ($\mathcal{A}_i =$ buy ski at time i).
(mixed strategy)

Dominated strategy :
A deterministic strategy $\mathcal{S}_1$ is dominated by $\mathcal{S}_2$ if for **every** possible choice of t by the adversary, the cost($\mathcal{S}_1$)$\geq$cost($\mathcal{S}_2$).

Theorem

No dominated strategy has a positive probability in an optimal mixed strategy.

Game theory perspective

- For every $i > B$, $\mathcal{A}_i$ has probability 0 in an opt. strategy.

Game theory perspective

- For every $i > B$, $\mathcal{A}_i$ has probability 0 in an opt. strategy.

Take $B = 4$

Cost of the strategies depending on the ending time

Str. / Stop	1	2	3	4
$\mathcal{A}_1$	B	B	B	B
$\mathcal{A}_2$	1	B+1	B+1	B+1
$\mathcal{A}_3$	1	2	B+2	B+2
$\mathcal{A}_4$	1	2	3	B+3

Game theory perspective

- For every $i > B$, $\mathcal{A}_i$ has probability 0 in an opt. strategy.

Take $B = 4$

Cost of the strategies depending on the ending time

Str. / Stop	1	2	3	4
$\mathcal{A}_1$	B	B	B	B
$\mathcal{A}_2$	1	B+1	B+1	B+1
$\mathcal{A}_3$	1	2	B+2	B+2
$\mathcal{A}_4$	1	2	3	B+3

Imagine that the opponent decide to stop at step 1. Then the optimal cost is 1 and the expected cost of the strategy is $Bp_1 + p_2 + p_3 + p_4$.

Game theory perspective

- For every $i > B$, $\mathcal{A}_i$ has probability 0 in an opt. strategy.

Take $B = 4$

Cost of the strategies depending on the ending time

Str. / Stop	1	2	3	4
$\mathcal{A}_1$	B	B	B	B
$\mathcal{A}_2$	1	B+1	B+1	B+1
$\mathcal{A}_3$	1	2	B+2	B+2
$\mathcal{A}_4$	1	2	3	B+3

Imagine that the opponent decide to stop at step 1. Then the optimal cost is 1 and the expected cost of the strategy is $Bp_1 + p_2 + p_3 + p_4$.

Similarly, if he decides to stop at step 2. The optimal cost is 2 and the expected cost of the strategy is $Bp_1 + (B+1)p_2 + 2p_3 + 2p_4$.

LP formulation

$$
\begin{aligned}
\min x & & \\
Bp_1 + p_2 + p_3 + p_4 & \leq & x \\
\frac{1}{2}(Bp_1 + (B+1)p_2 + 2p_3 + 2p_4) & \leq & x \\
\frac{1}{3}(Bp_1 + (B+1)p_2 + (B+2)p_3 + 3p_4) & \leq & x \\
\frac{1}{4}(Bp_1 + (B+1)p_2 + (B+2)p_3 + (B+3)p_4) & \leq & x \\
p_1 + p_2 + p_3 + p_4 & = & 1
\end{aligned}
$$

Best solution : $1/(1-\frac{1}{4})^4 \rightarrow (1-\frac{1}{e})^{-1}$.

Randomized lower bounds - Yao's lemma

Why is it complicated ?

Hard to find lower bounds : we have to find a strategy for opponent for every mixed strategy (and there are infinitely many...).

Randomized lower bounds - Yao's lemma

Why is it complicated ?

Hard to find lower bounds : we have to find a strategy for opponent for every mixed strategy (and there are infinitely many...).

Idea : Reverse the problem (via LP duality)

Randomized lower bounds - Yao's lemma

Why is it complicated ?

Hard to find lower bounds : we have to find a strategy for opponent for every mixed strategy (and there are infinitely many...).

Idea : Reverse the problem (via LP duality)

Yao's Lemma

Assume that there is a distribution $\mathcal{D}$ over instances of Π such that every deterministic online algorithm has expected competitive ratio at least μ. Then, the competitive ratio of every randomized online algorithm for Π is at least μ.

What about adaptive adversaries ?

You'll continue skiing until you decide to buy your skis !

$\rightarrow$ We cannot improve the 2-competitive factor.

Online matching

Model : Vertices arrive one by one (with their edges to already appeared vertices).

Matching : Subset of edges pairwise endpoint disjoint.

Online matching

Model : Vertices arrive one by one (with their edges to already appeared vertices).

Matching : Subset of edges pairwise endpoint disjoint.

Theorem

The Greedy Algorithm is $\frac{1}{2}$-competitive.

(Take an edge whenever it is possible)

Proof :

- The endpoints of the returned matching M is a vertex cover.
- By weak duality, $2|M| = VC \geq \min VC \geq OPT(\mathcal{M})$.

Online matching

Model : Vertices arrive one by one (with their edges to already appeared vertices).

Matching : Subset of edges pairwise endpoint disjoint.

Theorem

The Greedy Algorithm is $\frac{1}{2}$-competitive.
(Take an edge whenever it is possible)

Proof :

- The endpoints of the returned matching M is a vertex cover.
- By weak duality, $2|M| = VC \geq \min VC \geq OPT(\mathcal{M})$.

Theorem : No deterministic algorithm is α-competitive for $\alpha > \frac{1}{2}$.

Online matching

Model : Vertices arrive one by one (with their edges to already appeared vertices).

Matching : Subset of edges pairwise endpoint disjoint.

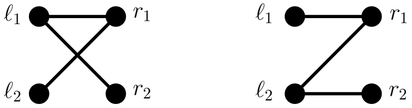
Theorem

The Greedy Algorithm is $\frac{1}{2}$-competitive.
(Take an edge whenever it is possible)

Proof :

- The endpoints of the returned matching M is a vertex cover.
- By weak duality, $2|M| = VC \geq \min VC \geq OPT(\mathcal{M})$.

Theorem : No deterministic algorithm is α-competitive for $\alpha > \frac{1}{2}$.



Online Fractional Bipartite Matching

Model :

Vertices of L are there from the beginning (offline vertices).
Vertices of R arrive one after another (online vertices).

Online Fractional Bipartite Matching

Model :

Vertices of L are there from the beginning (offline vertices).
Vertices of R arrive one after another (online vertices).

- Give weight to edges.
- Constraint : for every node, the sum of the weights of the edges incident to it is at most 1. (If weights are $\{0, 1\} \Rightarrow$ Matching)

Online Fractional Bipartite Matching

Model :
Vertices of L are there from the beginning (offline vertices).
Vertices of R arrive one after another (online vertices).

- Give weight to edges.
- Constraint : for every node, the sum of the weights of the edges incident to it is at most 1. (If weights are $\{0,1\} \Rightarrow$ Matching)

Naive algorithm : Balance weight between all the edges incident to it (when possible)

(That is if r_i has degree d, give weight $\frac{1}{d}$ to every edge incident to it, when possible)

Online Fractional Bipartite Matching

Model :

Vertices of L are there from the beginning (offline vertices).
Vertices of R arrive one after another (online vertices).

- Give weight to edges.
- Constraint : for every node, the sum of the weights of the edges incident to it is at most 1. (If weights are $\{0,1\} \Rightarrow$ Matching)

Naive algorithm : Balance weight between all the edges incident to it (when possible)

(That is if r_i has degree d, give weight $\frac{1}{d}$ to every edge incident to it, when possible)

(Equivalently : Give weight 1 to r_i and $\frac{1}{d}$ to its neighbors)

Online Fractional Bipartite Matching

Model :

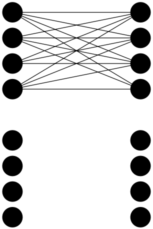
Vertices of L are there from the beginning (offline vertices).
Vertices of R arrive one after another (online vertices).

- Give weight to edges.
- Constraint : for every node, the sum of the weights of the edges incident to it is at most 1. (If weights are $\{0, 1\} \Rightarrow$ Matching)

Naive algorithm : Balance weight between all the edges incident to it (when possible)

(That is if r_i has degree d, give weight $\frac{1}{d}$ to every edge incident to it, when possible)

(Equivalently : Give weight 1 to r_i and $\frac{1}{d}$ to its neighbors)



Online Fractional Bipartite Matching

Model :

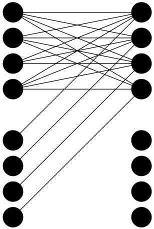
Vertices of L are there from the beginning (offline vertices).
Vertices of R arrive one after another (online vertices).

- Give weight to edges.
- Constraint : for every node, the sum of the weights of the edges incident to it is at most 1. (If weights are $\{0, 1\} \Rightarrow$ Matching)

Naive algorithm : Balance weight between all the edges incident to it (when possible)

(That is if r_i has degree d, give weight $\frac{1}{d}$ to every edge incident to it, when possible)

(Equivalently : Give weight 1 to r_i and $\frac{1}{d}$ to its neighbors)



Online Fractional Bipartite Matching

Model :

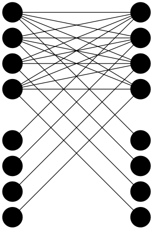
Vertices of L are there from the beginning (offline vertices).
Vertices of R arrive one after another (online vertices).

- Give weight to edges.
- Constraint : for every node, the sum of the weights of the edges incident to it is at most 1. (If weights are $\{0, 1\} \Rightarrow$ Matching)

Naive algorithm : Balance weight between all the edges incident to it (when possible)

(That is if r_i has degree d, give weight $\frac{1}{d}$ to every edge incident to it, when possible)

(Equivalently : Give weight 1 to r_i and $\frac{1}{d}$ to its neighbors)



Online Fractional Bipartite Matching

Model :

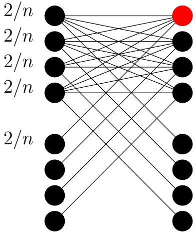
Vertices of L are there from the beginning (offline vertices).
Vertices of R arrive one after another (online vertices).

- Give weight to edges.
- Constraint : for every node, the sum of the weights of the edges incident to it is at most 1. (If weights are $\{0, 1\} \Rightarrow$ Matching)

Naive algorithm : Balance weight between all the edges incident to it (when possible)

(That is if r_i has degree d, give weight $\frac{1}{d}$ to every edge incident to it, when possible)

(Equivalently : Give weight 1 to r_i and $\frac{1}{d}$ to its neighbors)



Online Fractional Bipartite Matching

Model :

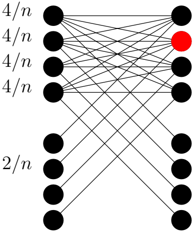
Vertices of L are there from the beginning (offline vertices).
Vertices of R arrive one after another (online vertices).

- Give weight to edges.
- Constraint : for every node, the sum of the weights of the edges incident to it is at most 1. (If weights are $\{0, 1\} \Rightarrow$ Matching)

Naive algorithm : Balance weight between all the edges incident to it (when possible)

(That is if r_i has degree d, give weight $\frac{1}{d}$ to every edge incident to it, when possible)

(Equivalently : Give weight 1 to r_i and $\frac{1}{d}$ to its neighbors)



What went wrong ?

We assign weight without distinction between neighbors.

Waterfilling algorithm

What went wrong ?

We assign weight without distinction between neighbors.

Waterfilling algorithm :

Balance weight : Maximize the minimum of the weights

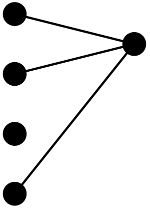
Waterfilling algorithm

What went wrong ?

We assign weight without distinction between neighbors.

Waterfilling algorithm :

Balance weight : Maximize the minimum of the weights



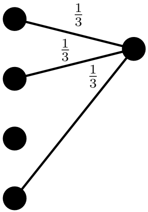
Waterfilling algorithm

What went wrong ?

We assign weight without distinction between neighbors.

Waterfilling algorithm :

Balance weight : Maximize the minimum of the weights



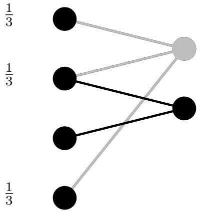
Waterfilling algorithm

What went wrong ?

We assign weight without distinction between neighbors.

Waterfilling algorithm :

Balance weight : Maximize the minimum of the weights



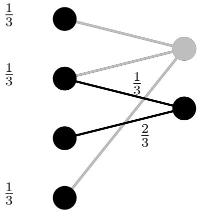
Waterfilling algorithm

What went wrong ?

We assign weight without distinction between neighbors.

Waterfilling algorithm :

Balance weight : Maximize the minimum of the weights



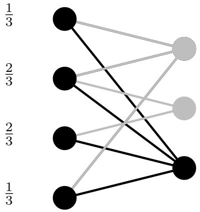
Waterfilling algorithm

What went wrong ?

We assign weight without distinction between neighbors.

Waterfilling algorithm :

Balance weight : Maximize the minimum of the weights



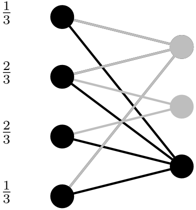
Waterfilling algorithm

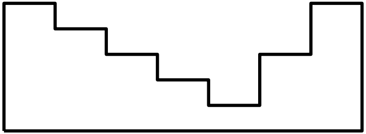
What went wrong ?

We assign weight without distinction between neighbors.

Waterfilling algorithm :

Balance weight : Maximize the minimum of the weights



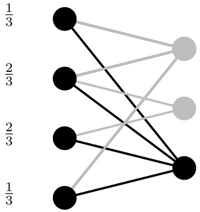


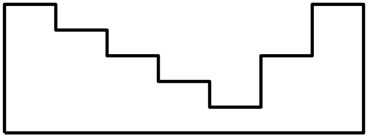
What went wrong ?

We assign weight without distinction between neighbors.

Waterfilling algorithm :

Balance weight : Maximize the minimum of the weights





Mathematically :

- $d(i) = \sum_{(i,j)\in E} x_{ij}$. (Initial level of water on ℓ_i)
- Find $\ell_j = \min_{i\in N(j)} d(i) + r_i$ such that $\sum r_i = 1$ (with $\ell_j \leq 1$).
  (Final level of water)
- Update x_{ij} : increase it by $\ell_j - d(i) = r_i$ (or 0 if neg.).

Primal-dual analysis

Primal-dual analysis

Fractional matching :

$$\max \sum_{(i,j)\in E} x_{ij}$$

$$\text{soumis à}$$

$$\sum_{j/(i,j)\in E} x_{ij} \leq 1 \qquad \forall i \in L$$

$$\sum_{i/(i,j)\in E} x_{ij} \leq 1 \qquad \forall j \in R$$

$$x_{ij} \leq 1 \qquad \forall (i,j) \in E$$

Primal-dual analysis

Fractional matching :

$$\max \sum_{(i,j)\in E} x_{ij}$$

soumis à

$$\sum_{j/(i,j)\in E} x_{ij} \leq 1 \qquad \forall i \in L$$

$$\sum_{i/(i,j)\in E} x_{ij} \leq 1 \qquad \forall j \in R$$

$$x_{ij} \leq 1 \qquad \forall (i,j) \in E$$

Fractional Vertex Cover

$$\min \sum \alpha_i + \beta_j$$

soumis à

$$\alpha_i + \beta_j \geq 1 \qquad \forall (i,j) \in E$$

$$\alpha_i, \beta_j \geq 0 \qquad \forall i, j$$

Primal-dual analysis

Fractional matching :

$$\max \sum_{(i,j)\in E} x_{ij}$$

soumis à

$$\sum_{j/(i,j)\in E} x_{ij} \leq 1 \qquad \forall i \in L$$

$$\sum_{i/(i,j)\in E} x_{ij} \leq 1 \qquad \forall j \in R$$

$$x_{ij} \leq 1 \qquad \forall (i,j) \in E$$

Fractional Vertex Cover

$$\min \sum \alpha_i + \beta_j$$

soumis à

$$\alpha_i + \beta_j \geq 1 \qquad \forall (i,j) \in E$$

$$\alpha_i, \beta_j \geq 0 \qquad \forall i, j$$

Idea :

- Start with a solution where $x_{ij} = 0$ (with no constraint since $G = \emptyset$).
- Update sol. by increasing x_{ij} and increasing α_i / creating β_j.

Primal-dual analysis

Fractional matching :

$$\max \sum_{(i,j)\in E} x_{ij}$$

soumis à

$$\sum_{j/(i,j)\in E} x_{ij} \leq 1 \qquad \forall i \in L$$

$$\sum_{i/(i,j)\in E} x_{ij} \leq 1 \qquad \forall j \in R$$

$$x_{ij} \leq 1 \qquad \forall (i,j) \in E$$

Fractional Vertex Cover

$$\min \sum \alpha_i + \beta_j$$

soumis à

$$\alpha_i + \beta_j \geq 1 \qquad \forall (i,j) \in E$$

$$\alpha_i, \beta_j \geq 0 \qquad \forall i, j$$

Idea :

- Start with a solution where $x_{ij} = 0$ (with no constraint since $G = \emptyset$).
- Update sol. by increasing x_{ij} and increasing α_i / creating β_j.

Each time a vertex is added, we update :

$$\begin{cases} \alpha_i = g(d(i)) \\ \beta_j = 1 - g(\ell(j)) \end{cases}$$

where $g(y) = \frac{e^y - 1}{e - 1}$

Analysis (cont.)

$$\begin{cases} \alpha_i = g(d(i)) \\ \beta_j = 1 - g(\ell(j)) \end{cases}$$

Analysis (cont.)

$$\begin{cases} \alpha_i = g(d(i)) \\ \beta_j = 1 - g(\ell(j)) \end{cases}$$

Observation 1 : For every $i, j \in E$, $\alpha_i + \beta_j \geq 1$.

Analysis (cont.)

$$\begin{cases} \alpha_i = g(d(i)) \\ \beta_j = 1 - g(\ell(j)) \end{cases}$$

Observation 1 : For every $i, j \in E$, $\alpha_i + \beta_j \geq 1$.
Proof :

- The level of water $d(i)$ increases with time and g is increasing.
- $\ell(j)$ is fixed forever and $\ell(j) \geq d(i)$ at step j.

Analysis (cont.)

$$\begin{cases} \alpha_i = g(d(i)) \\ \beta_j = 1 - g(\ell(j)) \end{cases}$$

Observation 1 : For every $i, j \in E$, $\alpha_i + \beta_j \geq 1$.
Proof :

- The level of water $d(i)$ increases with time and g is increasing.
- $\ell(j)$ is fixed forever and $\ell(j) \geq d(i)$ at step j.

Key lemma

$$\frac{e}{e-1} \sum_{i,j} x_{ij} \geq \sum_i \alpha_i + \sum_j \beta_j$$

Analysis (cont.)

$$\begin{cases} \alpha_i = g(d(i)) \\ \beta_j = 1 - g(\ell(j)) \end{cases}$$

Observation 1 : For every $i, j \in E$, $\alpha_i + \beta_j \geq 1$.
Proof :

- The level of water $d(i)$ increases with time and g is increasing.
- $\ell(j)$ is fixed forever and $\ell(j) \geq d(i)$ at step j.

Key lemma

$$\frac{e}{e-1} \sum_{i,j} x_{ij} \geq \sum_i \alpha_i + \sum_j \beta_j$$

By Weak Duality theorem, it provides a $\frac{e}{e-1}$-approximation algorithm.

Analysis (cont. 2)

How can we prove such a thing ?

$$\frac{e}{e-1}\sum_{i,j} x_{ij} \geq \sum_i \alpha_i + \sum_j \beta_j$$

Analysis (cont. 2)

How can we prove such a thing ?

$$\frac{e}{e-1}\sum_{i,j} x_{ij} \geq \sum_i \alpha_i + \sum_j \beta_j$$

Idea **(oversimplified)** **:**

What increases in the primal :

$$C = \sum_{i \in N(j)} r_i = \sum_{i \in N(j)} \ell_j - d(i)$$

Analysis (cont. 2)

How can we prove such a thing ?

$$\frac{e}{e-1}\sum_{i,j} x_{ij} \geq \sum_i \alpha_i + \sum_j \beta_j$$

Idea **(oversimplified)** **:**

What increases in the primal :

$$C = \sum_{i \in N(j)} r_i = \sum_{i \in N(j)} \ell_j - d(i)$$

What increases in the dual :

- $\beta_j = 1 - g(\ell(j))$.
- Each α_i in $N(j) \nearrow$ by $g(\ell(j)) - g(d(i))$.

Analysis (cont. 2)

How can we prove such a thing ?

$$\frac{e}{e-1}\sum_{i,j} x_{ij} \geq \sum_i \alpha_i + \sum_j \beta_j$$

Idea **(oversimplified)** **:**

What increases in the primal :

$$C = \sum_{i \in N(j)} r_i = \sum_{i \in N(j)} \ell_j - d(i)$$

What increases in the dual :

- $\beta_j = 1 - g(\ell(j))$.
- Each α_i in $N(j) \nearrow$ by $g(\ell(j)) - g(d(i))$. Rel. to integral of g' $(\times C)$.

Analysis (cont. 2)

How can we prove such a thing ?

$$\frac{e}{e-1}\sum_{i,j} x_{ij} \geq \sum_i \alpha_i + \sum_j \beta_j$$

Idea **(oversimplified)** **:**

What increases in the primal :

$$C = \sum_{i \in N(j)} r_i = \sum_{i \in N(j)} \ell_j - d(i)$$

What increases in the dual :

- $\beta_j = 1 - g(\ell(j))$. Related to the integral of $1 - g$ $(\times C)$.
- Each α_i in $N(j) \nearrow$ by $g(\ell(j)) - g(d(i))$. Rel. to integral of g' $(\times C)$.

Analysis (cont. 2)

How can we prove such a thing ?

$$\frac{e}{e-1}\sum_{i,j} x_{ij} \geq \sum_i \alpha_i + \sum_j \beta_j$$

Idea **(oversimplified)** **:**

What increases in the primal :

$$C = \sum_{i \in N(j)} r_i = \sum_{i \in N(j)} \ell_j - d(i)$$

What increases in the dual :

- $\beta_j = 1 - g(\ell(j))$. Related to the integral of $1 - g$ $(\times C)$.
- Each α_i in $N(j) \nearrow$ by $g(\ell(j)) - g(d(i))$. Rel. to integral of g' $(\times C)$.

$\Rightarrow g$ is the function satisfying $1 - g + g' = \frac{e}{e-1}$.

Summary

Theorem

The Waterfilling Algorithm is a deterministic algorithm for fractional matching of competitive ratio $\frac{e}{e-1}$.

Summary

Theorem

The Waterfilling Algorithm is a deterministic algorithm for fractional matching of competitive ratio $\frac{e}{e-1}$.

Remark : No deterministic algorithm can beat this ratio.

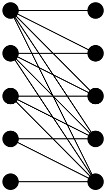
Summary

Theorem

The Waterfilling Algorithm is a deterministic algorithm for fractional matching of competitive ratio $\frac{e}{e-1}$.

Remark : No deterministic algorithm can beat this ratio.

Proof :
Half graph $=$ Edges l_i, r_j for every $j \geq i$.



Summary

Theorem

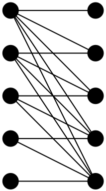
The Waterfilling Algorithm is a deterministic algorithm for fractional matching of competitive ratio $\frac{e}{e-1}$.

Remark : No deterministic algorithm can beat this ratio.

Proof :

Half graph = Edges l_i, r_j for every $j \geq i$.

No deterministic algorithm can behave well against all the permutations of the RHS of the half graph.



Randomized Online Bipartite Matching

Model : Vertices of L are there from the beginning (offline vertices).
Vertices of R arrive one after another (online vertices).

Reminder : No deterministic algorithm can beats competitive ratio $\frac{1}{2}$.

Randomized algorithm

Theorem (Karp, Vazirani, Vazirani '90, Goel, Mehta'08)

There exists a $(1 - \frac{1}{e})$-competitive randomized algorithm for online bipartite matching.

Randomized algorithm

Theorem (Karp, Vazirani, Vazirani '90, Goel, Mehta'08)

There exists a $(1 - \frac{1}{e})$-competitive randomized algorithm for online bipartite matching.

Algorithm RANKING

Choose a random ordering σ of A.

When a vertex of B arrives, match it with its largest (in σ) available neighbor in A.

Randomized algorithm

Theorem (Karp, Vazirani, Vazirani '90, Goel, Mehta'08)

There exists a $(1 - \frac{1}{e})$-competitive randomized algorithm for on-line bipartite matching.

Algorithm RANKING
Choose a random ordering σ of A.
When a vertex of B arrives, match it with its largest (in σ) available neighbor in A.

Two proofs :

- Primal dual approach (Devanur, Jain, Kleinberg '13)
- With a "typical" probabilistic argument KVV'90, GM'08

Primal dual approach

For the analysis : instead of a ranking, we associate to each vertex i of L a random real Y_i in $[0,1]$.

For the analysis : instead of a ranking, we associate to each vertex i of L a random real Y_i in $[0, 1]$.

Tim Roughgarden "*The rough idea is to set things up so that the probability that a given edge is included the matching plays the same role as its fractional value in the WF algorithm*".

Primal dual approach

For the analysis : instead of a ranking, we associate to each vertex i of L a random real Y_i in $[0, 1]$.

Tim Roughgarden "*The rough idea is to set things up so that the probability that a given edge is included the matching plays the same role as its fractional value in the WF algorithm*".

(Very sketchy) flavour of the proof :

- We will define some (randomized) α_i, β_j when (i, j) are matched.
- $\alpha_i = \frac{e}{e-1} h(Y_i)$ and $\beta_i = \frac{e}{e-1}(1 - h(Y_i))$.
- If (i, j) is added in M then the dual increases by $\frac{e}{e-1}$.

Primal dual approach

For the analysis : instead of a ranking, we associate to each vertex i of L a random real Y_i in $[0, 1]$.

Tim Roughgarden "*The rough idea is to set things up so that the probability that a given edge is included the matching plays the same role as its fractional value in the WF algorithm*".

(Very sketchy) flavour of the proof :

- We will define some (randomized) α_i, β_j when (i, j) are matched.
- $\alpha_i = \frac{e}{e-1} h(Y_i)$ and $\beta_i = \frac{e}{e-1}(1 - h(Y_i))$.
- If (i, j) is added in M then the dual increases by $\frac{e}{e-1}$.
- Key Lemma : For every $(i, j) \in E$, $\mathbb{E}(\alpha_i + \beta_j) \geq 1$.

Primal dual approach

For the analysis : instead of a ranking, we associate to each vertex i of L a random real Y_i in $[0,1]$.

Tim Roughgarden "*The rough idea is to set things up so that the probability that a given edge is included the matching plays the same role as its fractional value in the WF algorithm*".

(Very sketchy) flavour of the proof :

- We will define some (randomized) α_i, β_j when (i,j) are matched.
- $\alpha_i = \frac{e}{e-1} h(Y_i)$ and $\beta_i = \frac{e}{e-1}(1 - h(Y_i))$.
- If (i,j) is added in M then the dual increases by $\frac{e}{e-1}$.
- Key Lemma : For every $(i,j) \in E$, $\mathbb{E}(\alpha_i + \beta_j) \geq 1$.
- $\Rightarrow$ In expectation the constraints of the dual are satisfied.

Primal dual approach

For the analysis : instead of a ranking, we associate to each vertex i of L a random real Y_i in $[0,1]$.

Tim Roughgarden "*The rough idea is to set things up so that the probability that a given edge is included the matching plays the same role as its fractional value in the WF algorithm*".

(Very sketchy) flavour of the proof :

- We will define some (randomized) α_i, β_j when (i,j) are matched.
- $\alpha_i = \frac{e}{e-1} h(Y_i)$ and $\beta_i = \frac{e}{e-1}(1-h(Y_i))$.
- If (i,j) is added in M then the dual increases by $\frac{e}{e-1}$.
- Key Lemma : For every $(i,j) \in E$, $\mathbb{E}(\alpha_i + \beta_j) \geq 1$.
- $\Rightarrow$ In expectation the constraints of the dual are satisfied.

Primal dual approach

For the analysis : instead of a ranking, we associate to each vertex i of L a random real Y_i in $[0, 1]$.

Tim Roughgarden "*The rough idea is to set things up so that the probability that a given edge is included the matching plays the same role as its fractional value in the WF algorithm*".

(Very sketchy) flavour of the proof :

- We will define some (randomized) α_i, β_j when (i, j) are matched.
- $\alpha_i = \frac{e}{e-1} h(Y_i)$ and $\beta_i = \frac{e}{e-1}(1 - h(Y_i))$.
- If (i, j) is added in M then the dual increases by $\frac{e}{e-1}$.
- Key Lemma : For every $(i, j) \in E$, $\mathbb{E}(\alpha_i + \beta_j) \geq 1$.
- $\Rightarrow$ In expectation the constraints of the dual are satisfied.

Follows from properties of $h(y) = e^{y-1}$ close to the ones of the previous proof.

Thanks for your attention !