Online algorithms for dummies

Nicolas Bousquet

Journées CALAMAR
(Journées Combinatoires des Alpes, des Littoraux Atlantique et Méditerranéen, 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.
**Greedy Algorithm**: Give to each vertex the smallest possible color.

Illustration: Graph coloring on trees
Illustration : Graph coloring on trees

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

→ This algorithm may output a $(\Delta + 1)$ coloring.

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

([Diagram of a tree with vertices colored])
Greedy Algorithm: Give to each vertex the smallest possible color.

Illustration: Graph coloring on trees

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

→ This algorithm may output a $(\Delta + 1)$ coloring. (while there exists a 2-coloring)
Greedy Algorithm: Give to each vertex the smallest possible color.

→ 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

**Performance of an online algorithm** Given a maximization problem, $I$ an instance, an algorithm is:

- **$\alpha$-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.

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.
Performance of online algorithms

**Performance of an online algorithm** Given a maximization problem, $I$ an instance, an algorithm is:

- **$\alpha$-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.

- **$\alpha$-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

Performance of an online algorithm Given a maximization problem, \( I \) an instance, an algorithm is:

- **\( \alpha \)-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.

- **\( \alpha \)-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.

Algorithm 1: Buy a pair immediately.
Opponent strategy: Stop immediately after day 1.
Competitive ratio: $B \rightarrow \infty$. 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: $nB \rightarrow +\infty$ when $n$ tends to infinity.
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.

![Simba](28x230) **Buy a pair of ski or not.**

![Simba](28x230) **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** : $\frac{B}{1}$. → 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.

![Ski figures](image)

**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.

The break-even algorithm is $2^{B-1}$-competitive.

**Theorem**

**Proof:** Let $k$ be 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 $2B - 1$. 
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 ratio 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)

There exists a \((1 - \frac{1}{e})^{-1}\)-competitive randomized online algorithm for ski rental.
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 $p_i$.
$\iff$ A randomized algorithm is a superposition of (a possibly infinite number of) deterministic algorithm ($A_i = \text{buy ski at time } i$).

(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 ($A_i =$ buy ski at time $i$).

(mixed strategy)

**Dominated strategy:**
A deterministic strategy $S_1$ is dominated by $S_2$ if for every possible choice of $t$ by the adversary, the cost($S_1$)$\geq$cost($S_2$).
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 ($A_i = \text{buy ski at time } i$).

(mixed strategy)

**Dominated strategy:**
A deterministic strategy $S_1$ is dominated by $S_2$ if for every possible choice of $t$ by the adversary, the cost($S_1$) ≥ cost($S_2$).

**Theorem**
No dominated strategy has a positive probability in an optimal mixed strategy.
Game theory perspective

- For every $i > B$, $A_i$ has probability 0 in an opt. strategy.
Game theory perspective

• For every $i > B$, $A_i$ has probability 0 in an opt. strategy.

Take $B = 4$

Cost of the strategies depending on the ending time

<table>
<thead>
<tr>
<th>Str. / Stop</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>$A_2$</td>
<td>1</td>
<td>B+1</td>
<td>B+1</td>
<td>B+1</td>
</tr>
<tr>
<td>$A_3$</td>
<td>1</td>
<td>2</td>
<td>B+2</td>
<td>B+2</td>
</tr>
<tr>
<td>$A_4$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>B+3</td>
</tr>
</tbody>
</table>
Game theory perspective

- For every $i > B$, $A_i$ has probability 0 in an opt. strategy.  

Take $B = 4$

Cost of the strategies depending on the ending time

<table>
<thead>
<tr>
<th>Str. / Stop</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>$A_2$</td>
<td>1</td>
<td>B+1</td>
<td>B+1</td>
<td>B+1</td>
</tr>
<tr>
<td>$A_3$</td>
<td>1</td>
<td>2</td>
<td>B+2</td>
<td>B+2</td>
</tr>
<tr>
<td>$A_4$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>B+3</td>
</tr>
</tbody>
</table>

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 \( i > B \), \( A_i \) has probability 0 in an opt. strategy.

Take \( B = 4 \)

<table>
<thead>
<tr>
<th>Str. / Stop</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_1 )</td>
<td>( B )</td>
<td>( B )</td>
<td>( B )</td>
<td>( B )</td>
</tr>
<tr>
<td>( A_2 )</td>
<td>1</td>
<td>( B+1 )</td>
<td>( B+1 )</td>
<td>( B+1 )</td>
</tr>
<tr>
<td>( A_3 )</td>
<td>1</td>
<td>2</td>
<td>( B+2 )</td>
<td>( B+2 )</td>
</tr>
<tr>
<td>( A_4 )</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>( B+3 )</td>
</tr>
</tbody>
</table>

Imagine that the opponent decides 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

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

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 $\Pi$ such that every deterministic online algorithm has expected competitive ratio at least $\mu$. Then, the competitive ratio of every randomized online algorithm for $\Pi$ is at least $\mu$. 
What about adaptive adversaries?

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

→ 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.

The Greedy Algorithm is $1/2$-competitive. (Take an edge whenever it is possible)

**Theorem**

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

**Theorem**: No deterministic algorithm is $\alpha$-competitive for $\alpha > 1/2$. 

$\ell_1 \leq \ell_2 \leq r_1 \leq r_2$ 

$\ell_1 \ell_2 r_1 r_2 \ell_1 \ell_2 r_1 r_2$ 

$\ell_1 \ell_2 r_1 r_2$ 

$17/27$
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, $2|M| = VC \geq \min VC \geq OPT(M)$. 
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, $2|M| = VC \geq \min VC \geq OPT(M)$.

**Theorem** : No deterministic algorithm is $\alpha$-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, $2|M| = VC \geq \min VC \geq OPT(M)$.

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

\[
\begin{align*}
\ell_1 \quad & \quad \bullet \quad & \quad r_1 \\
\ell_2 \quad & \quad \bullet \quad & \quad r_2 \\
\ell_1 \quad & \quad \bullet \quad & \quad r_1 \\
\ell_2 \quad & \quad \bullet \quad & \quad r_2
\end{align*}
\]
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 $1/d$ to every edge incident to it, when possible)
(Equivalently: Give weight 1 to $r_i$ and $1/d$ 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)
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)
Waterfilling algorithm

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

Mathematically:

\[ d(i) = \sum_{(i,j) \in E} x_{ij}. \]

Initial level of water on \( \ell_i \)

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.).
Waterfilling algorithm

What went wrong?
We assign weight without distinction between neighbors.

Waterfilling algorithm:
Balance weight: Maximize the minimum of the weights

Mathematically:
• $d(i) = \sum_{j \in E} x_{ij}$ (Initial level of water on $\ell_i$)
• 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.)
Waterfilling algorithm

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 $\ell_i$)
• 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.).
Waterfilling algorithm

What went wrong?
We assign weight without distinction between neighbors.

Waterfilling algorithm:
Balance weight: Maximize the minimum of the weights.

\[ d(i) = \sum_{(i,j) \in E} x_{ij}. \] (Initial level of water on \( \ell_i \))

\[ \text{Find } \ell_j = \min_{i \in N(j)} d(i) + r_i \text{ such that } \sum r_i = 1 \text{ (with } \ell_j \leq 1). \] (Final level of water)

\[ \text{Update } x_{ij}: \text{ increase it by } \ell_j - d(i) = r_i \text{ (or 0 if neg.).} \]
Waterfilling algorithm

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 \( \ell_i \))
- 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.)
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

Mathematically:
- \( d(i) = \sum_{(i,j) \in E} X_{ij} \). (Initial level of water on \( \ell_i \))
- 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

Fractional matching:

\[
\max \sum_{(i,j) \in E} x_{ij} \text{ subject to } \sum_{j \in (i,j) \in E} x_{ij} \leq 1 \forall i \in L
\]

\[
\sum_{i \in (i,j) \in E} x_{ij} \leq 1 \forall j \in R
\]

\[
x_{ij} \leq 1 \forall (i,j) \in E
\]

Fractional Vertex Cover:

\[
\min \sum \alpha_i + \beta_j \text{ subject to } \alpha_i + \beta_j \geq 1 \forall (i,j) \in E
\]

\[
\alpha_i, \beta_j \geq 0 \forall i, j
\]

Idea:

- Start with a solution where \(x_{ij} = 0\) (with no constraint since \(G = \emptyset\)).
- Update solution by increasing \(x_{ij}\) and increasing \(\alpha_i\) creating \(\beta_j\). Each time a vertex is added, we update:

\[
\alpha_i = g(d(i))
\]

\[
\beta_j = 1 - g(\ell(j))
\]

where \(g(y) = e^y - 1/e - 1/2\)
Primal-dual analysis

Fractional matching :

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

soumis à

\[
\sum_{j/(i,j) \in E} x_{ij} \leq 1 \quad \forall i \in L
\]

\[
\sum_{i/(i,j) \in E} x_{ij} \leq 1 \quad \forall j \in R
\]

\[
x_{ij} \leq 1 \quad \forall (i,j) \in E
\]
Primal-dual analysis

Fractional matching:

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

soumis à

\[
\sum_{j \neq (i,j) \in E} x_{ij} \leq 1 \quad \forall i \in L
\]

\[
\sum_{i \neq (i,j) \in E} x_{ij} \leq 1 \quad \forall j \in R
\]

\[
x_{ij} \leq 1 \quad \forall (i,j) \in E
\]

Fractional Vertex Cover

\[
\min \sum \alpha_i + \beta_j
\]

soumis à

\[
\alpha_i + \beta_j \geq 1 \quad \forall (i,j) \in E
\]

\[
\alpha_i, \beta_j \geq 0 \quad \forall i, j
\]
Primal-dual analysis

Fractional matching:

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

soumis à

\[ \sum_{j/(i,j) \in E} x_{ij} \leq 1 \quad \forall i \in L \]

\[ \sum_{i/(i,j) \in E} x_{ij} \leq 1 \quad \forall j \in R \]

\[ x_{ij} \leq 1 \quad \forall (i,j) \in E \]

Idea:

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

Fractional Vertex Cover

\[ \min \sum \alpha_i + \beta_j \]

soumis à

\[ \alpha_i + \beta_j \geq 1 \quad \forall (i,j) \in E \]

\[ \alpha_i, \beta_j \geq 0 \quad \forall i, j \]
Primal-dual analysis

**Fractional matching :**

\[
\text{max} \sum_{(i,j) \in E} x_{ij} \text{ soumis à } \sum_{j/(i,j) \in E} x_{ij} \leq 1 \quad \forall i \in L
\]
\[
\sum_{i/(i,j) \in E} x_{ij} \leq 1 \quad \forall j \in R
\]
\[
x_{ij} \leq 1 \quad \forall (i,j) \in E
\]

**Fractional Vertex Cover**

\[
\text{min} \sum \alpha_i + \beta_j \text{ soumis à } \alpha_i + \beta_j \geq 1 \quad \forall (i,j) \in E
\]
\[
\alpha_i, \beta_j \geq 0 \quad \forall i, j
\]

**Idea :**

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

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.)

\[
\begin{align*}
\alpha_i &= g(d(i)) \\
\beta_j &= 1 - g(\ell(j))
\end{align*}
\]

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
By Weak Duality theorem, it provides an \(e - 1\)-approximation algorithm.
Analysis (cont.)

\begin{align*}
\left\{ \begin{array}{l}
\alpha_i = g(d(i)) \\
\beta_j = 1 - g(\ell(j))
\end{array} \right.
\end{align*}

**Observation 1**: For every \(i, j \in E\), \(\alpha_i + \beta_j \geq 1\).
Analysis (cont.)

\[
\begin{align*}
\alpha_i &= g(d(i)) \\
\beta_j &= 1 - g(\ell(j))
\end{align*}
\]

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.)

\[
\begin{aligned}
\alpha_i &= g(d(i)) \\
\beta_j &= 1 - g(\ell(j))
\end{aligned}
\]

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

By Weak Duality theorem, it provides an \(\frac{e}{e-1}\)-approximation algorithm.
Analysis (cont.)

\[
\begin{align*}
\alpha_i &= g(d(i)) \\
\beta_j &= 1 - g(\ell(j))
\end{align*}
\]

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.
How can we prove such a thing?

\[
\frac{e}{e - 1} \sum_{i,j} x_{ij} \geq \sum_{i} \alpha_i + \sum_{j} \beta_j
\]
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 \( \alpha_i \) in \( N(j) \) by \( g(\ell(j)) - g(d(i)) \).
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 \( \alpha_i \) in \( N(j) \) 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 \(\alpha_i \) in \(N(j) \uparrow\) by \(g(\ell(j)) - g(d(i))\). Rel. to integral of \(g' (\times C)\).
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 \( \alpha_i \) in \( N(j) \) by \( g(\ell(j)) - g(d(i)) \). Rel. to integral of \( g' \times C \).

\( \Rightarrow g \) is the function satisfying \( 1 - g + g' = \frac{e}{e-1} \).
The Waterfilling Algorithm is a deterministic algorithm for fractional matching of competitive ratio $\frac{e}{e-1}$.
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.
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 $j \geq i$. 
**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 on-line bipartite matching.

**Algorithm** **Ranking**

Choose a random ordering \(\sigma\) of \(A\).
When a vertex of \(B\) arrives, match it with its largest (in \(\sigma\)) available neighbor in \(A\).
There exists a \((1 - \frac{1}{e})\)-competitive randomized algorithm for online bipartite matching.

**Algorithm Ranking**
Choose a random ordering \(\sigma\) of \(A\).
When a vertex of \(B\) arrives, match it with its largest (in \(\sigma\)) 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 \([0, 1]\).
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”.
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) $\alpha_i, \beta_j$ 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) $\alpha_i, \beta_j$ 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) $\alpha_i, \beta_j$ 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) \( \alpha_i, \beta_j \) 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) \( \alpha_i, \beta_j \) 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.
Conclusion

Thanks for your attention!