

# *Approximation Algorithms*

Presentation: Valia Mitsou

# Outline

---

1. Introduction
2. Vertex Cover
3. Knapsack
4. TSP

# 1. Introduction

---

# Optimization Problems

---

- **Optimization Problem:** Every **instance** of the problem corresponds to some **feasible solutions** each of them having a value via an **Objective Function**.
- We seek for an **Optimal Solution** i.e. a feasible solution that has an optimal value.
- Optimization problems can be either **Maximization** or **Minimization**
- **Example:** The Vertex Cover Problem
  - **Min or Max:** Minimization
  - **Instance:** A graph
  - **Feasible Solutions:** Every Vertex Cover
  - **Objective Function:** The cardinality  $| \star |$  function
  - **Optimal Solution:** A Vertex Cover of minimum cardinality

## The PO-class (i)

---

Consider a minimization problem:

Given an instance of size  $n$  try to find the minimum possible feasible solution.

Then the corresponding decision problem would be:

Given an instance of size  $n$  and a fixed  $k$  (in binary) is there any feasible solution of value less or equal to  $k$ ?

~> If the decision version is polynomially solvable on  $n$  and  $\log k$  then we can construct a polynomial time algorithm for the optimization version

## The PO-class (ii)

---

- Determine  $l$ ,  $k = 2^l$  such that there is a feasible solution of value less or equal to  $2^l$  but there is not a feasible solution of value less or equal to  $2^{l-1}$ , by running  $\log k$  times the polynomial time algorithm for the decision version.
- Then do binary search to find the exact value of  $k$  ( $\log k$  runs of the decision version algorithm).

This implies a polynomial time algorithm on the size of the input. We call the class of problems that have a polynomial time solvable decision version **PO class** (PO stands for P-Optimization).

## The NPO-class

---

Problems in PO are polynomial time solvable. Thus we turn our attention to **NP-Optimization Problems** (i.e. the corresponding decision problem is in NP) and especially in **NP-hard problems**. Unless  $P=NP$  we cannot have a polynomial time algorithm to compute the optimal value for general instance of an NP-hard problem.

- Solve the problem exactly on limited instances.
- Find polynomial time **approximation algorithms**

## Notation

---

- $\Pi$ : Problem
- $I$ : Instance
- $\text{SOL}_A(\Pi, I)$ : The solution we obtain for the instance  $I$  of the problem  $\Pi$  using algorithm  $A$ .
- $\text{OPT}(\Pi, I)$ : The optimal solution for the instance  $I$  of the problem  $\Pi$ .

**Note:** We usually omit  $\Pi$ ,  $I$  and  $A$  from the above notation.

# Approximability

- An algorithm  $A$  for a **minimization** problem  $\Pi$  achieves a  $\rho_A$  **approximation factor**,  $(\rho_A : \mathbb{N} \rightarrow \mathbb{Q}^+)$  if for every instance  $I$  of size  $|I| = n$ :

$$\frac{\text{SOL}_A(I)}{\text{OPT}(I)} \leq \rho_A(n)$$

- An algorithm  $A$  for a **maximization** problem  $\Pi$  achieves a  $\rho_A$  **approximation factor**,  $(\rho_A : \mathbb{N} \rightarrow \mathbb{Q}^+)$  if for every instance  $I$  of size  $|I| = n$ :

$$\frac{\text{SOL}_A(I)}{\text{OPT}(I)} \geq \rho_A(n)$$

$\rightsquigarrow$  An approximation algorithm of factor  $\rho$  guarantees that the solution that the algorithm computes cannot be worse than  $\rho$  times the optimal solution.

# Approximation Schemes

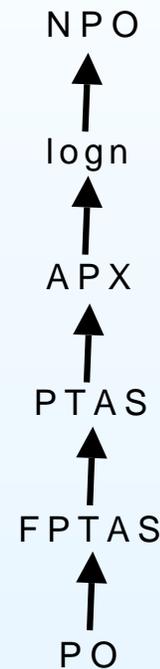
**Informally:** We can have as good approximation factor as we want trading off time.

**Formally:**

- $A$  is an **Approximation Scheme** (AS) for problem  $\Pi$  if on input  $(I, \varepsilon)$ , where  $I$  an instance and  $\varepsilon > 0$  an error parameter:
  - $\text{SOL}_A(I, \varepsilon) \leq (1 + \varepsilon) \cdot \text{OPT}(I)$ , for minimization problem
  - $\text{SOL}_A(I, \varepsilon) \geq (1 - \varepsilon) \cdot \text{OPT}(I)$ , for maximization problem
- $A$  is a **PTAS** (Polynomial Time AS) if for every fixed  $\varepsilon > 0$  it runs in polynomial time in the size of  $I$ .
- $A$  is an **FPTAS** (Fully PTAS) if for every fixed  $\varepsilon > 0$  it runs in polynomial time in the size of  $I$  and in  $1/\varepsilon$ .

# Approximation World

Depending on the approximation factor we have several classes of approximation:



- **logn**:  $\rho(n) = O(\log n)$
- **APX**:  $\rho(n) = \rho$  (constant factor approximation)

# Representatives

---

- **Non-approximable**: Traveling Salesman Problem
- **logn**: Set Cover
- **APX**: Ferry Cover 
- **PTAS**: Makespan Scheduling
- **FPTAS**: Knapsack

## 2. Vertex Cover

## The (Cardinality) Vertex Cover Problem

---

**Definition:** Given a graph  $G(V, E)$  find a minimum cardinality Vertex Cover, i.e. a set  $V' \subseteq V$  such that every edge has at least one endpoint in  $V'$ .

- A trivial feasible solution would be the set  $V$
- Finding a minimum cardinality Vertex Cover is NP-hard (reduction from 3-SAT)
- An approximation algorithm of factor 2 will be presented

## Lower Bounding

A general strategy for obtaining a  $\rho$ -approximation algorithm (for a minimization problem) is the following:

- Find a lower bound  $l$  of the optimal solution ( $l \leq \text{OPT}$ )
- Find a factor  $\rho$  such that  $\text{SOL} = \rho \cdot l$

$\rightsquigarrow$  The previous scheme implies  $\text{SOL} \leq \rho \cdot \text{OPT}$

# Matchings

- **Definition:** Given a graph  $G(V, E)$  a matching is a subset of the edges  $M \subseteq E$  such that no two edges in  $M$  share an endpoint.
- **Maximal Matching:** A matching that no more edges can be added.
- **Maximum Matching:** A maximum cardinality matching.

~> Maximal Matching is solved in polynomial time with the greedy algorithm

~> Maximum Matching is also solved in polynomial time via a reduction to max-flow

## A 2-Approximation Algorithm for Vertex Cover

---

- **The Algorithm:** Find a maximal matching  $M$  of the graph and output the set  $V'$  of matched vertices
  - **Correctness:**
    - Edges belonging in  $M$  are all covered by  $V'$
    - Since  $M$  is a maximal matching, any other edge  $e \in E \setminus M$  will share at least one endpoint  $v$  with some  $e' \in M$ . So  $v$  is in  $V'$  and guards  $e$ .
  - **Analysis:**
    - Any vertex cover should pick at least one endpoint of each matched edge  $\rightarrow |M| \leq \text{OPT}$
    - $|V'| = 2|M|$
- Thus  $\text{SOL} = |V'| = 2|M| \leq 2\text{OPT} \Rightarrow \text{SOL} \leq 2\text{OPT}$

$\rightsquigarrow$  Vertex Cover is in APX

# Can we do better?

---

## Questions

- Can the approximation guarantee be improved by a better analysis?
- Can an approximation algorithm with a better guarantee be designed using the same lower bounding scheme?
- Is there some other lower bounding methods that can lead to an improved approximation algorithm?

## Answers

- Tight Examples
- Other kind of examples
- This is not so immediate...

## Tight Examples

- A better analysis might imply an  $l'$  s.t.  $l < l' \leq \text{OPT}$ . Then there would be a  $\rho' < \rho$  s.t.  $\rho \cdot l = \rho' \cdot l'$ , so

$$\text{SOL} = \rho \cdot l = \rho' \cdot l' \leq \rho' \text{OPT}$$

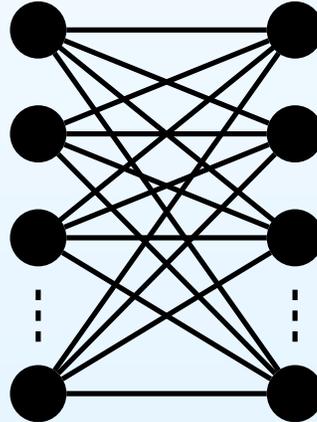
Thus we could obtain a better approximation factor  $\rho' < \rho$ .

- **Definition:** An infinite family of instances in which  $l = \text{OPT}$  is called **Tight Example** for the  $\rho$ -approximation algorithm.
- If  $l = \text{OPT}$  then there is no  $l' > l$  s.t.  $l' \leq \text{OPT}$ .  
~> So we can't find a better factor by better analysis

## Tight Example for the matching algorithm

---

- The infinite family  $K_{n,n}$  of the complete balanced bipartite graphs is a tight example.
- $|M| = n = \text{OPT}$ . So the solution returned is 2 times the optimal solution.



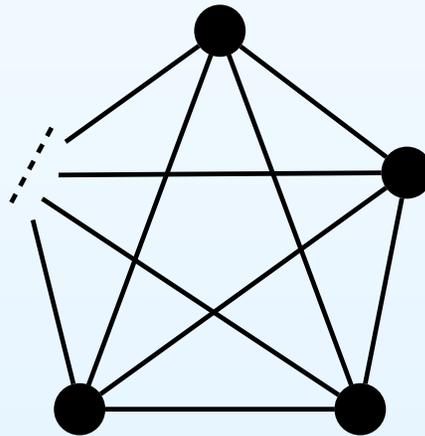
## Other kind of examples

---

- Using the same lower bound  $l \leq \text{OPT}$  we might find a better algorithm with  $\rho' < \rho$  that computes  $\text{SOL} = \rho' \cdot l$ . This would imply a better  $\rho'$  approximation algorithm.
- An infinite family where  $l = \frac{1}{\rho} \text{OPT}$  implies that  $\text{SOL} = l \cdot \rho' = \frac{1}{\rho} \rho' \text{OPT} < \text{OPT}$  (contradiction).  
~> Thus it is impossible to find another algorithm with better approximation factor using the lower bound  $l \leq \text{OPT}$

## Using the matching lower bound

- The infinite family  $K_{2n+1}$  of the complete bipartite graphs with odd number of vertices have an optimal vertex of cardinality  $2n$
- A maximal matching could be  $|M| = n = \frac{1}{2}\text{OPT}$ . So the solution returned is the optimal solution.



## Other lower bounds for Vertex Cover

---

- This is still an open research area.
- Best known result for the approximation factor (until 2004) is  $2 - \Theta\left(\frac{1}{\sqrt{\log n}}\right)$  (due to **George Karakostas**)
- Uses Linear Programming.

# 3. Knapsack

---

# Pseudo-polynomial time algorithms

---

- An instance  $I$  of any problem  $\Pi$  consists of **objects** (sets, graphs, ...) and **numbers**.
- The size of  $I$  ( $|I|$ ) is the number of bits needed to write the instance  $I$ .
- Numbers in  $I$  are written in **binary**
- Let  $I_u$  be the instance  $I$  where all numbers are written in **unary**
- **Definition:** A **pseudo-polynomial** time algorithm is an algorithm running in polynomial time in  $|I_u|$
- Pseudo-polynomial time algorithms can be obtained using **Dynamic Programming**

## Strong NP-hardness

- **Definition:** A problem is called **strongly NP-hard** if any problem in NP can be polynomially reduced to it and numbers in the reduced instance are written in unary
- **Informally:** A strongly NP-hard problem remains NP-hard even if the input numbers are less than some polynomial of the size of the objects.

⇒ Strongly NP-hard problems cannot admit a pseudo-polynomial time algorithm, assuming  $P \neq NP$   
(else we could solve the reduced instance in polynomial time, thus we could solve every problem in NP in polynomial time. That would imply  $P = NP$ )

## The existence of FPTAS

---

**Theorem:** For a minimization problem  $\Pi$  if  $\forall$  instance  $I$ ,

- OPT is strictly bounded by a polynomial of  $|I_u|$  and
- the objective function is integer valued

then  $\Pi$  admits an FPTAS  $\Rightarrow$   $\Pi$  admits a pseudo-polynomial time algorithm

$\rightsquigarrow$  A strongly NP-hard problem (under the previous assumptions) cannot admit an FPTAS unless  $P = NP$

## The Knapsack Problem (i)

---

- **Definition:** The discrete version is given a set of  $n$  items  $X = \{x_1, \dots, x_n\}$  where a *profit*  $: X \rightarrow \mathbb{N}$  and a *weight*  $: X \rightarrow \mathbb{N}$  function are provided and a “knapsack” of total capacity  $B \in \mathbb{N}$ , find a subset  $Y \subseteq X$  whose total size is bounded by  $B$  and maximizes the total profit.
- **Definition:** The continuous version is given a set of  $n$  continuous items  $X = \{x_1, \dots, x_n\}$  where *profit* and *weight* function are provided and a “knapsack” of total capacity  $B \in \mathbb{N}$ , find a sequence  $\{w_1, \dots, w_n\}$  of portions where  $\sum_{i=1}^n w_i = B$  that maximizes the total profit.

## The Knapsack Problem (ii)

---

- The **greedy algorithm** (sort the objects by decreasing ratio of profit to weight) solves in **polynomial time the continuous version**
- The greedy algorithm can be made to perform arbitrarily bad for the discrete version.
- **Discrete Knapsack is NP-hard**
- **Pseudo-polynomial time** and **FPTAS** algorithms will be presented for the discrete version.
- For now on we focus on **discrete knapsack** and call it “**knapsack**”

## A pseudo-polynomial time algorithm for knapsack (i)

---

- Let  $P$  be the profit of the most profitable object
- $nP$  is a trivial upper bound on the total profit
- For  $i \in \{1, \dots, n\}$  and  $p \in \{1, \dots, nP\}$  let  $S(i, p)$  denote a subset of  $\{x_1, \dots, x_i\}$  whose total profit is exactly  $p$  and its total weight is minimized
- Let  $W(i, p)$  denote the weight of  $S(i, p)$  ( $\infty$  if no such a set exists)

## A pseudo-polynomial time algorithm for knapsack (ii)

---

The following inductive relation computes all values  $W(i, p)$  in  $O(n^2 P)$

- $W(1, p)$  is  $weight(x_1)$  if  $p = profit(x_1)$ ,  $\infty$  else
- $W(i + 1, p) = \begin{cases} W(i, p), & profit(x_{i+1}) > p \\ \min\{W(i, p), weight(x_{i+1}) + W(i, p - profit(x_{i+1}))\}, & \text{else} \end{cases}$

The optimal solution of the problem is  $\max\{p | W(n, p) \leq B\}$

$\rightsquigarrow$  The optimal solution can be computed in polynomial time on  $n$  and  $P$

## An FPTAS for Knapsack

- **Idea:** The previous algorithm could be a polynomial time algorithm if  $P$  was bounded by a polynomial of  $n$
- Ignore a number of least significant bits of the profits of the objects
- Modified profits  $profit'$  should now be numbers **bounded by a polynomial of  $n$  and  $\frac{1}{\varepsilon}$**  ( $\varepsilon$  is the error parameter)
- The algorithm:
  1. Given  $\varepsilon > 0$  define  $K = \frac{\varepsilon P}{n}$
  2. Set new profit function  $profit'$ ,  $profit'(x_i) = \lfloor \frac{profit(x_i)}{K} \rfloor$
  3. Run the pseudo-polynomial time algorithm described previously and output the result

# Analysis

**Theorem:** The previous algorithm is an FPTAS

1.  $SOL \geq (1 - \varepsilon)OPT$
2. Runs in polynomial time in  $n$  and  $\frac{1}{\varepsilon}$

**Proof:**

1. Let  $S$  and  $O$  denote the output set and the optimal set

- $profit'(x_i) = \lfloor \frac{profit(x_i)}{K} \rfloor \Rightarrow profit(x_i) \geq K \cdot profit'(x_i) \geq profit(x_i) - K$
- $K = \frac{\varepsilon P}{n}$
- $profit'(S) \geq profit'(O)$
- $OPT \geq P$

Thus,  $SOL = profit(S) \geq K \cdot profit'(S) \geq K \cdot profit'(O) \geq profit(O) - nK = OPT - \varepsilon P \geq (1 - \varepsilon) \cdot OPT$

2. The algorithm's running time is  $O(n^2 \lfloor \frac{P}{K} \rfloor) = O(n^2 \lfloor \frac{n}{\varepsilon} \rfloor)$

# 4. TSP

---

## Hardness of Approximation

---

To show that an optimization problem  $\Pi$  is hard to approximate we can use

- A **Gap-introducing reduction**: Reduces an NP-complete decision problem  $\Pi'$  to  $\Pi$
- A **Gap-preserving reduction**: Reduces a hard to approximate optimization problem  $\Pi'$  to  $\Pi$

## Gap-introducing reductions (i)

---

Suppose that  $\Pi'$  is a decision problem and  $\Pi$  a minimization problem (similar for maximization).

A reduction  $h$  from  $\Pi'$  to  $\Pi$  is called **gap-introducing** if:

1. Transforms (in polynomial time) any instance  $I'$  of  $\Pi'$  to an instance  $I = h(I')$  of  $\Pi$
2. There are functions  $f$  and  $\alpha$  s.t.
  - If  $I'$  is a **'yes instance'** of  $\Pi'$  then  $\text{OPT}(\Pi, I) \leq f(I)$
  - If  $I'$  is a **'no instance'** of  $\Pi'$  then  $\text{OPT}(\Pi, I) > \alpha(|I|) \cdot f(I)$

## Gap-introducing reductions (ii)

**Theorem:** If  $\Pi'$  is NP-complete then  $\Pi$  cannot be approximated with a factor  $\alpha$

**Proof:** If  $\Pi$  had an approximation algorithm of factor  $\alpha$  then  $\text{SOL} \leq \alpha \cdot \text{OPT}$ . So,

- $I'$  is a 'yes instance' of  $\Pi'$   $\Rightarrow \text{SOL} \leq \alpha \cdot \text{OPT}(\Pi, I) \leq \alpha \cdot f(I)$
- $I'$  is a 'no instance' of  $\Pi'$   $\Rightarrow \text{SOL} > \text{OPT}(\Pi, I) > \alpha(|I|) \cdot f(I)$

Then by using the approximation algorithm for  $P_i$  we could be able to determine in polynomial time whether the instance  $I'$  is 'yes' or 'no'.

Since  $P_i$  is NP-complete, this would imply  $P = NP$

## Gap-preserving reductions (i)

Suppose that  $\Pi'$  is a minimization problem and  $\Pi$  a minimization (similar for other cases).

A reduction  $h$  from  $\Pi'$  to  $\Pi$  is called **gap-preserving** if:

1. Transforms (in polynomial time) any instance  $I'$  of  $\Pi'$  to an instance  $I = h(I')$  of  $\Pi$
2. There are functions  $f, f', \alpha, \beta$  s.t.
  - $\text{OPT}(\Pi', I') \leq f'(I') \Rightarrow \text{OPT}(\Pi, I) \leq f(I)$
  - $\text{OPT}(\Pi', I') > \beta(|I'|) \cdot f'(I') \Rightarrow \text{OPT}(\Pi, I) > \alpha(|I|) \cdot f(I)$

## Gap-preserving reductions (ii)

**Theorem:** If  $\Pi'$  is non-approximable with a factor  $\beta$  then  $\Pi$  cannot be approximated with a factor  $\alpha$  unless  $P = NP$

**Proof:** If  $\Pi$  had an approximation algorithm of factor  $\alpha$  then  $\text{SOL} \geq \alpha \cdot \text{OPT}$ . So,

- $\text{OPT}(\Pi', I') \leq f'(I') \Rightarrow \text{SOL} \leq \alpha \cdot \text{OPT}(\Pi, I) \leq \alpha \cdot f(I)$
- $\text{OPT}(\Pi', I') > \beta(|I'|)f'(I') \Rightarrow \text{SOL} > \text{OPT}(\Pi, I) > \alpha(|I|) \cdot f(I)$

But  $P_{i'}$  cannot be approximated with a factor  $\beta$  means that there is an NP-complete decision problem  $P_{i''}$  and a gap-introducing reduction from  $P_{i''}$  to  $P_{i'}$  s.t.

- $I''$  is a 'yes instance' of  $\Pi'' \Rightarrow \text{OPT}(\Pi', I') \leq f''(I')$
- $I''$  is a 'no instance' of  $\Pi'' \Rightarrow \text{OPT}(\Pi', I') > \beta(|I''|) \cdot f''(I')$

Thus, by running the algorithm for  $\Pi$  we could decide  $\Pi''$ . This implies  $P = NP$

# The Traveling Salesman Problem

---

**Definition:** Given a complete graph  $K_n(V, E)$  and a weight function  $w : E \rightarrow \mathbb{Q}$  find a tour, i.e. a permutation of the vertices, that has minimum total weight.

- The TSP problem is NP-hard
- TSP is non-approximable with a factor  $\alpha(n)$  polynomial in  $n$ , via a gap-introducing reduction from **Hamilton Cycle**.

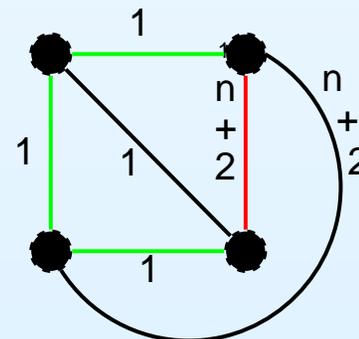
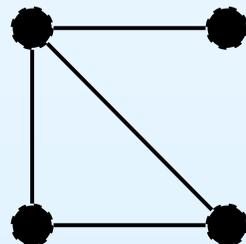
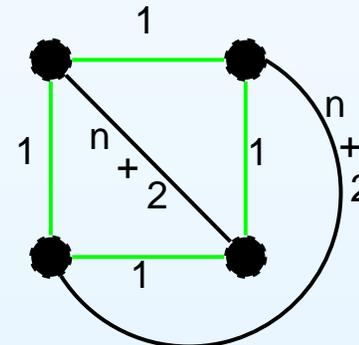
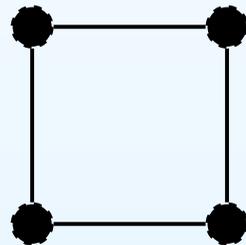
**Definition:** Given a graph  $G(V, E)$  a **Hamilton Cycle** is a cycle that uses every vertex only once.

- To determine whether  $G$  has a Hamilton Cycle or not is NP-complete.

## TSP is non-approximable (i)

**Reduction:**  $G(V, E), |V| = n$ , is an instance of Hamilton Cycle.  
The instance of TSP will be  $K_n$  with a weight function  $w$ ,  
 $w(e) = 1$  if  $e \in E$  else  $w(e) = n + 2$ . Then

- If  $G$  has a Hamilton Cycle then  $\text{OPT}(\text{TSP}) = n$
- If  $I'$  is a 'no instance' of  $\Pi'$  then  $\text{OPT}(\text{TSP}) > 2n$



## TSP is non-approximable (ii)

---

- ~> TSP is **APX-hard**, i.e. there exist a constant  $\alpha$  (in the example 2) that TSP cannot be approximated with factor  $\alpha$ , unless  $P = NP$
- ~> **Bonus!!!** In the reduction if we set  $w(e) = \alpha(n) \cdot n, e \notin E$  then we cannot have an  $\alpha(n)$  approximation factor for TSP. Thus TSP is non-approximable

THE END!!!