# Approximation Algorithms

- Approximation Algorithms for Optimization Problems: Types
- Absolute Approximation Algorithms
- Inapproximability by Absolute Approximate Algorithms
- Relative Approximation Algorithm
- InApproximability by Relative Approximate Algorithms
- Polynomial Time Approximation Schemes
- Fully Polynomial Time Approximation Schemes

Imdad ullah Khan

# NP-Hardness

When you prove a problem $X$ to be NP-Hard, then as per the almost consensus opinion of $\mathrm{P} \neq \mathrm{NP}$, it essentially means

1. There is no polynomial time
2. deterministic algorithm
3. to exactly/optimally solve the problem $X$
4. for all possible input instances

What are the option? Things to consider when your problem is NP-Hard

# Coping with NP-Hardness

- Do I need to solve the problem for all valid input instances?
  - Sometimes just need to solve a restricted version of the problem -
    ▷ (special cases) that include realistic instances
- Is exponential-time OK for my instances?
  - Exponential-time algorithms are "not slow" ▷ they don't scale well
  - If relevant instances are small, then they may be acceptable
  - Can bring exponent/base of runtime down ▷ $2^n \to 2^{\sqrt{n}}$ or $2^n \to 1.5^n$
- Is non-optimality OK?
  - What if our algorithm is better than others ▷ faster than bruteforce

# Coping with NP-Hardness

**Approaches to tackle hard problems**

1. **Special Cases:** Relevant structure on which the problem is easy
   - Exact results in poly-time only for special cases or a range of parameters
2. **Intelligent Exhaustive Search:** Exponential time in worst case
   - The base and/or exponent are usually smaller
   - Could be efficient on typical more realistic instances
   - Backtracking, Brand-and-Bound
3. **Nearly exact solutions:** Output is '*close*' to exact (optimal) solution
   - Approximation Algorithms: Solutions of guaranteed quality in poly-time
   - Heuristic: Solutions hopefully good in poly-time
4. **Randomized Algorithms:** Use coin flips for making decisions
   - Typically used for approximation, also used for problems in $\mathrm{P}$

# Coping with NP-Hardness

## To cope with NP-Hardness, sacrifice one of these features

| Poly-time | Deterministic | Exact/Opt Solution | All cases/ Parameters | Algorithmic Paradigm |
|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✗ | Special Cases Algorithms<br>Fixed Parameter Tractability |
| ✓ | ✓ | ✗ | ✓ | Approximation Algorithms<br>Heuristic Algorithms |
| ✗ | ✓ | ✓ | ✓ | Intelligent<br>Exhaustive Search |
| ✓ | ✗ | $\mathbb{E}$(✓) | ✓ | Mote Carlo<br>Randomized Algorithm |
| $\mathbb{E}$(✓) | ✗ | ✓ | ✓ | Las Vegas<br>Randomized Algorithm |

- Special cases of input instances (based on structure of a range of parameter(s))
- Approximation algorithms guarantee a bound on suboptimality
- Heuristics algorithms do not have any guarantee
- Randomized algorithms are generally used for problems in class $\mathrm{P}$

# Optimization Problems

An optimization problem P is characterized by three things

- $\mathcal{I}$: set of (valid) input instances
- $S(I)$: solution space, set of feasible solutions for an instance $I \in \mathcal{I}$
- $f : S(I) \to \mathbb{R}$: function giving value to each feasible solution

# Optimization Problems

An optimization problem P is characterized by three things

- $\mathcal{I}$: set of (valid) input instances
- $S(I)$: solution space, set of feasible solutions for an instance $I \in \mathcal{I}$
- $f : S(I) \to \mathbb{R}$: function giving value to each feasible solution

## Optimization Problem can be

- A maximization problem: Given $I \in \mathcal{I}$, the objective is to find a solution $s^* \in S(I)$ such that $f(s^*)$ is maximum, i.e.

$$\forall s \in S(I),\ f(s^*) \geq f(s)$$

- A minimization problem is defined analogously

# Optimization Problems

An optimization problem P is characterized by three things

- $\mathcal{I}$: set of (valid) input instances
- $S(I)$: solution space, set of feasible solutions for an instance $I \in \mathcal{I}$
- $f : S(I) \to \mathbb{R}$: function giving value to each feasible solution

Optimization Problem can be

- A maximization problem: Given $I \in \mathcal{I}$, the objective is to find a solution $s^* \in S(I)$ such that $f(s^*)$ is maximum, i.e.

$$\forall s \in S(I),\ f(s^*) \geq f(s)$$

- A minimization problem is defined analogously

**Note that optimal solution ($s^*$) need not be unique**

# Approximation Algorithms

Relax the requirement that algorithm always outputs optimal solution

Instead look for a feasible solution $s'$, whose value $f(s')$ is **close** to the value of optimal solution $s^*$

An approximation algorithm $A$ for an optimization problem $P$, is a **polynomial time** algorithm that on input instance $I \in \mathcal{I}$ outputs a solution $s \in S(I)$ **such that** $f(s)$ **is close to** $f(s^*)$

- $A(I)$: the solution output by $A$
- $\textsc{opt}(I)$: an optimal solution

We seek worst case closeness guarantees on values of outputs of $A$

i.e. we try to bound $\max_{I \in \mathcal{I}} \left| f(A(I)) - f(\textsc{opt}(I)) \right|$

# Quality of Approximation: Types

## Absolute Approximation Algorithms

Given an optimization problem $P$ with value function $f$ on solution space

An algorithm $A$ is called **absolute approximation** algorithm if there is a constant $k$ such that for any instance $I$

$$\left|f(A(I)) - f(\textsc{opt}(I))\right| \leq k$$

- For a minimization problem this means $f(A(I)) \leq f(\textsc{opt}(I)) + k$
- For a maximization problem this means $f(A(I)) \geq f(\textsc{opt}(I)) - k$

# Quality of Approximation: Types

## Approximation Factor/Ratio

Given an optimization problem $P$ with value function $f$ on solution space

The approximation ratio or approximation factor of an algorithm $A$ is defined as the ratio *'between'* value of output of $A$ and value of $\text{OPT}$

- For minimization problem it is $f\big(A(I)\big)/f\big(\text{OPT}(I)\big)$
- For maximization problem it is $f\big(\text{OPT}(I)\big)/f\big(A(I)\big)$

▷ Note: approximation factor is always bigger than 1

Generally, approximation factor is defined as $max\left\{\frac{f\left(A(I)\right)}{f\left(\text{OPT}(I)\right)}, \frac{f\left(\text{OPT}(I)\right)}{f\left(A(I)\right)}\right\}$

# Quality of Approximation: Types

## Relative Approximation Algorithm

Given an optimization problem $P$ with value function $f$ on solution space

An algorithm $A$ is called a $\alpha(n)$-**approximate** algorithm, if for any instance $I$ of size $n$, $A$ achieves an approximation ratio $\alpha(n)$

- For a minimization problem this means $f\left(A(I)\right) \;\leq\; \alpha(n)\cdot f\left(\textsc{opt}(I)\right)$
- For a maximization problem this means $f\left(A(I)\right) \;\geq\; {}^{1}/_{\alpha(n)}\cdot f\left(\textsc{opt}(I)\right)$

# Quality of Approximation: Types

## Constant Factor (relative) Approximation Algorithm

Given an optimization problem $P$ with value function $f$ on solution space

An algorithm $A$ is called an $\alpha$-**approximate** algorithm, if for any instance $I$, $A$ achieves an approximation ratio $\alpha$

- For a minimization problem this means $f\left(A(I)\right) \leq \alpha \cdot f\left(\text{OPT}(I)\right)$
- For a maximization problem this means $f\left(A(I)\right) \geq {}^{1}/_{\alpha} \cdot f\left(\text{OPT}(I)\right)$

# Quality of Approximation: Types

## Approximation Error

Given an optimization problem $P$ with value function $f$ on solution space

The approximation error of $A$ is its approximation factor minus 1

- For a minimization problem it is

  $$f\big(A(I)\big)/f\big(\text{OPT}(I)\big) - 1 \quad = \quad f\big(A(I)\big) - f\big(\text{OPT}(I)\big)/f\big(\text{OPT}(I)\big)$$

- For a maximization problem it is

  $$f\big(\text{OPT}(I)\big)/f\big(A(I)\big) - 1 \quad = \quad f\big(\text{OPT}(I)\big) - f\big(A(I)\big)/f\big(A(I)\big)$$

▷ Useful when approximation ratio is close to 1

Also called relative approximation error

# Quality of Approximation: Types

## Polynomial Time Approximation Scheme (PTAS)

Given an optimization problem $P$ with value function $f$ on solution space

A family of algorithms $A(\epsilon)$ is called a **polynomial time approximation scheme** if for a given parameter $\epsilon$, on any instance $I$, $A(\epsilon)$ achieves an approximation error $\epsilon$ and runtime of $A$ is polynomial in $|I| = n$

- For a minimization problem this means $f\left(A(I)\right) \leq (1+\epsilon) \cdot f\left(\text{OPT}(I)\right)$
- For a maximization problem this means $f\left(A(I)\right) \geq (1-\epsilon) \cdot f\left(\text{OPT}(I)\right)$

Runtime of $A$ could be exponential in $1/\epsilon$ ▷ e.g. $O(n^{1/\epsilon})$

# Quality of Approximation: Types

## Fully Polynomial Time Approximation Scheme (FPTAS)

Given an optimization problem $P$ with value function $f$ on solution space

A family of algorithms $A(\epsilon)$ is called a **fully polynomial time approximation scheme** if for a given $\epsilon$, on any instance $I$, $A(\epsilon)$ achieves an approximation error $\epsilon$ and runtime of $A$ is polynomial in $|I| = n$ and $1/\epsilon$

- For a minimization problem this means $f(A(I)) \leq (1+\epsilon) \cdot f(\textsc{opt}(I))$
- For a maximization problem this means $f(A(I)) \geq (1-\epsilon) \cdot f(\textsc{opt}(I))$

Runtime of $A$ cannot be exponential in $1/\epsilon$ ▷ e.g. $O(1/\epsilon^2 n^3)$

**Constant factor decrease in $\epsilon$ increases runtime by a constant factor**

# Quality of Approximation: Types

Some simple exercises to clarify the definitions

- What does an $k$-absolute approximate algorithm mean for $k = 0$?
- What does an $\alpha$-approximate algorithm mean for $\alpha = 1$?
- What is the error of 2-approximate algorithm?
- What is the approx. factor of an algor with 1% approx. error?
- Is $\alpha$-approximate algorithm the same $\alpha = (1 + \epsilon)$-PTAS ?

- An absolute approximate algorithm is the most desirable, why?
- Absolute approximate algorithms are rare, FPTAS is the next desirable

  ▷ Not known for many problem, but when available they are almost as good as an optimal algorithm