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

What is the practical meaning of 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

How to cope with NP-HARDNESS? Things to consider

- 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 includes 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 down exponent or base of runtime ($2^n \rightarrow 2^{\sqrt{n}}$ or $2^n \rightarrow 1.5^n$)
- Is non-optimality OK?
  - Is it OK if our algorithm outrun other algorithms (the brute force one)?

"I can't find an efficient algorithm, but neither can all these famous people."

How to cope with NP-HARDNESS? Sacrifice some feature

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

Approaches to tackle NP-HARD problems

1. Special Cases: Relevant structure on which the problem is easier
   - 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 Algorithms: Solutions hopefully good in poly-time
4. Randomized Algorithms: Use coin flips for making decisions
   - Typically used for approximation, also used for problems in P

Optimization Problems

An optimization problem 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

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

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

- Minimization problems are 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 an optimal solution, $f(s^*)$
- We seek worst case closeness guarantees between $f(s')$ and $f(s^*)$

An approximation algorithm A for an optimization problem is a **polynomial time** algorithm that on input instance $I \in \mathcal{I}$ outputs a solution $s' \in S(I)$

- $A(I)$: the solution output by A $\triangleright$ s'
- $\text{OPT}(I)$: an optimal solution $\triangleright$ s^*

such that $\max\limits_{I \in \mathcal{I}} \left| f(A(I)) - f(\text{OPT}(I)) \right|$ or $\max\limits_{I \in \mathcal{I}} \dfrac{f(A(I))}{f(\text{OPT}(I))}$ is **small**

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

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

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

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 OPT

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

Note: approximation factor is always bigger than 1, generally

Approximation factor is defined as $max\left\{\frac{f\big(A(I)\big)}{f\big(\mathrm{OPT}(I)\big)}, \frac{f\big(\mathrm{OPT}(I)\big)}{f\big(A(I)\big)}\right\}$

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

Constant Factor (relative) Approximation Algorithm

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

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

- For a minimization problem1 this means $f\big(A(I)\big) \;\leq\; \alpha \cdot f\big(\text{OPT}(I)\big)$
- For a maximization problem this means $f\big(A(I)\big) \;\geq\; {}^{1}\!/\!{}_{\alpha} \cdot f\big(\text{OPT}(I)\big)$

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

  $f\big(A(I)\big)/f\big(\textsc{opt}(I)\big) - 1 \quad = \quad f\big(A(I)\big) - f\big(\textsc{opt}(I)\big)/f\big(\textsc{opt}(I)\big)$

- For a maximization problem it is

  $f\big(\textsc{opt}(I)\big)/f\big(A(I)\big) - 1 \quad = \quad f\big(\textsc{opt}(I)\big) - f\big(A(I)\big)/f\big(A(I)\big)$

It is very useful when approximation ratio is close to 1

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 ϵ, on any instance I, $A(\epsilon)$ achieves an approximation error ϵ and runtime of A is polynomial in $|I| = n$

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

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

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 ϵ, on any instance I, $A(\epsilon)$ achieves an approximation error ϵ and runtime of A is polynomial in $|I| = n$ and $1/\epsilon$

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

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

Constant factor decrease in ϵ increases runtime by a constant factor

Quality of Approximation: Types

- An absolute approximate algorithm is the most desirable, why?
- What does an α-approximate algorithm mean for $\alpha = 1$?
- What is the error of 2-approximate algorithm?
- What is the approximation factor of an algorithm with 1% approximation error?

Absolute approximate algorithms are rare, an FPTAS is the next desirable

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