Approximation Algorithms
NP-complete problems: problems A in NP such that $B \leq_p A$ for any B in NP

Examples: Circuit-Sat, SATISFIABILITY, 3-SAT, CLIQUE, VertexCover, Subset Sum, Hamiltonian Cycle, ...

NP-complete problems cannot be solved in polynomial time, unless $P = NP$

What to do when you want to solve an NP-complete (optimization) problem?

- still use **exact algorithm**: algorithm guaranteed to compute optimal solution
  - will not be polynomial-time, but perhaps fast enough for some instances

- **heuristic**: algorithm without guarantees on the quality of the solution

- **approximation algorithm**: algorithm that computes solution whose quality is guaranteed to be within a certain factor from OPT
Approximation algorithms: terminology

Consider minimization problem

OPT(I) = value of optimal solution for problem instance I
ALG(I) = value of solution computed by algorithm ALG for problem instance I

ALG is $\rho$-approximation algorithm if $\text{ALG}(I) \leq \rho \cdot \text{OPT}(I)$ for all inputs $I$

Note: $\rho$ can be a function of input size $n$ (e.g.: $O(\log n)$-approximation)

Example: Vertex Cover
if ALG computes, for any graph $G$, a cover with at most twice the minimum number of nodes in any vertex cover for $G$, then ALG is 2-approximation
Approximation algorithms: terminology (cont’d)

Consider minimization problem

OPT(I) = value of optimal solution for problem instance I
ALG(I) = value of solution computed by algorithm ALG for problem instance I

ALG is $\rho$-approximation algorithm if $ALG(I) \leq \rho \cdot OPT(I)$ for all inputs I

ALG(I) ≥ $\frac{1}{\rho} \cdot OPT(I)$

Example: CLIQUE
if ALG computes, for any graph G, a clique with at least half the maximum number of nodes in any clique in G, then ALG is 2-approximation.

NB In the literature, one often talks about $(1/\rho)$-approximation instead of $\rho$-approximation for maximization problems. In example, ALG would be $(1/2)$-approximation algorithm.
For $\rho$-approximation algorithm ALG (for minimization problem) we must

- describe algorithm …
- … and analyze running time …
- … and prove that $\text{ALG}(I) \leq \rho \cdot \text{OPT}(I)$ for all inputs $I$

How can we prove that $\text{ALG}(I) \leq \rho \cdot \text{OPT}(I)$ if we don’t know OPT ?!

we need to prove a lower bound on $\text{OPT}(I)$…

for maximization problems we need upper bound

Example: $\text{OPT}(I) \geq 100$

$\implies \text{ALG}(I) \leq 2 \cdot \text{OPT}(I)$

$\text{ALG}(I) \leq 200$
Example 1: Vertex Cover
G = (V,E) is undirected graph

vertex cover in G: subset \( C \subseteq V \) such that for each edge \((u,v)\) in \( E \) we have \( u \) in \( C \) or \( v \) in \( C \) (or both)

**Vertex Cover** (optimization version)

**Input:** undirected graph \( G = (V,E) \)

**Problem:** compute vertex cover for \( G \) with minimum number of vertices

Vertex Cover is NP-hard.
An approximation algorithm for Vertex Cover: first attempt

Approx(?)-Vertex-Cover \((G)\)
1. \(C \leftarrow \text{empty set}; \quad E^* \leftarrow E\)  // \(E^* = \text{set of edges not covered by } C\)
2. \textbf{while} \(E^*\) not empty
3. \textbf{do} take edge \((u,v)\) in \(E^*\)
4. \(C \leftarrow C \cup \{u\}\)
5. remove from \(E^*\) all edges that have \(u\) as an endpoint
6. \textbf{return} \(C\)
Does Approx(?)_Vertex-Cover have good approximation ratio?

No: if we pick the wrong vertices, approximation ratio can be $|V| - 1$

Second attempt: greedy

- always pick vertex that covers largest number of uncovered edges
- gives $O(\log |V|)$-approximation algorithm: better, but still not so good
An approximation algorithm for Vertex Cover: third attempt

Approx(2)-Vertex-Cover (G)
1. $C \leftarrow \text{empty set}; \quad E^* \leftarrow E$ // $E^*$ = set of edges not covered by $C$
2. while $E^*$ not empty
3. do take edge $(u,v)$ in $E^*$
4. $C \leftarrow C \cup \{u\} \{u,v\}$
5. remove from $E^*$ all edges that have $u$ as an endpoint
6. return $C$
Approx-Vertex-Cover \((G)\)

1. \(C \leftarrow \text{empty set}; \quad E^* \leftarrow E \quad // \quad E^* = \text{set of edges not covered by } C\)
2. \textbf{while} \(E^*\) not empty
3. \textbf{do} take edge \((u,v)\) in \(E^*\)
4. \(C \leftarrow C \cup \{u,v\}\)
5. remove from \(E^*\) all edges that have \(u\) and/or \(v\) as an endpoint
6. \textbf{return} \(C\)

Proving approximation ratio, step 1: try to get lower bound on OPT

![Graph](image)

two edges are \textbf{non-adjacent} if they do not have an endpoint in common

**Lemma.** Let \(N \subseteq E\) be any set of non-adjacent edges. Then \(OPT \geq |N|\).
Approx-Vertex-Cover ($G$)

1. $C \leftarrow \text{empty set}; \quad E^* \leftarrow E$ // $E^*$ = set of edges not covered by $C$
2. while $E^*$ not empty
3. do take edge $(u,v)$ in $E^*$
4. C $\leftarrow C \cup \{u,v\}$ // $N \leftarrow N \cup \{(u,v)\}$
5. remove from $E^*$ all edges that have $u$ and/or $v$ as an endpoint
6. return $C$

Theorem. Approx-Vertex-Cover is 2-approximation algorithm and can be implemented to run in $O( |V| + |E| )$ time.

Proof (of approximation ratio). Consider any input graph $G$.

Let $N$ be set of edges selected in step 3 during the algorithm.

- $N$ is set of non-adjacent edges, so $\text{OPT}(G) \geq |N|$
- $|C| = 2|N|$

Hence, $\text{ALG}(G) = |C| = 2|N| \leq 2\ \text{OPT}(G)$. 

\[\square\]
Theorem. Approx-Vertex-Cover is 2-approximation algorithm and can be implemented to run in $O(|V| + |E|)$ time.

Can we do better? (1.5)-approximation, or (1.1)-approximation, or ... ?)

Slightly better algorithm is known: approximation ratio $2 - \frac{1}{\Theta(\log |V|)}$

Theorem. No polynomial-time (1.3606)-approximation algorithm exists for Vertex Cover unless P = NP.

but some other problems can be approximated much better ...
Example 2: TSP
MST versus TSP

Min Spanning Tree
Input: weighted graph
Output: minimum-weight tree connecting all nodes

greedy: $O(|E| + |V| \log |V|)$

Traveling Salesman (TSP)
Input: complete weighted graph (non-negative edge weights)
Output: minimum-weight tour connecting all nodes

NP-complete
what about approximation algorithm?
Lemma. MST is lower bound: for any graph G, we have $\text{TSP}(G) \geq \text{MST}(G)$.

Triangle inequality: $\text{weight}(u, w) \leq \text{weight}(u, v) + \text{weight}(v, w)$ for any three vertices $u, v, w$.

A simple 2-approximation for TSP with triangle inequality:

Approx-TSP
1. Compute MST on G.
2. Double each edge of MST to get a cycle.
3. Take shortcuts to get a tour visiting every vertex exactly once.
Hardness of approximation of TSP without triangle inequality

**Theorem.** Let \( \rho \geq 1 \) be any constant. Then there cannot be a polynomial-time \( \rho \)-approximation algorithm for general TSP, unless P = NP.

\[ G: \quad \text{old edges weight 0} \]
\[ G^*: \quad \text{new edges: weight 1} \]

\( G \) has Hamiltonian cycle \iff \( G^* \) has tour of length \( |V| \)

\( \Rightarrow \) \( \rho \)-approx algorithm computes tour \( T \) with \( \text{length}(T) \leq \rho \cdot \text{OPT} = \rho \cdot |V| \)

and such a tour can only use edges of length 1
Polynomial-time approximation schemes

Consider minimization problem.

A polynomial-time approximation scheme (PTAS) is an algorithm ALG with two parameters, an input instance I and a number \( \varepsilon > 0 \), such that:

i. \( ALG(I, \varepsilon) \leq (1+\varepsilon) \cdot OPT(I) \) (so it’s \((1+\varepsilon)\)-approximation algorithm)

ii. running time of ALG is polynomial in \( n \) for any constant \( \varepsilon > 0 \).

Example of possible running times: \( O(n^2 / \varepsilon^5) \), \( O(2^{1/\varepsilon} n \log n) \), \( O(n^{3/\varepsilon}) \), etc.

FPTAS = fully polynomial-time approximation scheme

= PTAS with running time not only polynomial in \( n \), but also in \( 1/\varepsilon \)
Polynomial-time approximation schemes

Consider a minimization problem.

A polynomial-time approximation scheme (PTAS) is an algorithm ALG with two parameters, an input instance I and a number $\varepsilon > 0$, such that:

i. $\text{ALG}(I, \varepsilon) \leq (1 + \varepsilon) \cdot \text{OPT}(I)$ (so it’s $(1 + \varepsilon)$-approximation algorithm)

ii. running time of ALG is polynomial in $n$ for any constant $\varepsilon > 0$.

Example of possible running times: $O(n^2 / \varepsilon^5)$, $O(2^{1/\varepsilon} n \log n)$, $O(n^{3/\varepsilon})$, etc.

FPTAS = fully polynomial-time approximation scheme

= PTAS with running time not only polynomial in $n$, but also in $1/\varepsilon$
**Subset Sum (optimization version)**

“Find maximum load for a truck, under the condition that the load does not exceed a given weight limit for the truck.”

**Input**: multi-set $X$ of non-negative integers, non-negative integer $t$

**Problem**: compute subset $S$ that maximizes $\sum_{x \in S} x$

under the condition that $\sum_{x \in S} x \leq t$ ?

**Example**: $X = \{3, 5, 7, 10, 16, 23, 41\}$  
$t = 50$

$S = \{3, 5, 41\}$ gives sum 49, which is optimal

For simplicity we will only compute value of optimal solution, not solution itself.
An exponential-time exact algorithm for Subset Sum

Use backtracking: generate all subset sums that are at most $t$

```
Exact-Subset-Sum ($X$, $t$)
1. $L \leftarrow$ Generate-Subset-Sums ($X$, $t$)
2. return the maximum sum in $L$
```
Generating subset sums

- \( X = \{ x_1, x_2, \ldots, x_n \} \)
- choices: do we take \( x_1 \) into subset? do we take \( x_2 \) into subset? ETC

Define \( L_i = \{ \text{all possible sums you can make with } x_1, \ldots, x_i \} \)

Example: \( X = 2, 7, 5 \)

\( L_0 = \{ 0 \} \)

\( L_1 = \{ 0 \} \cup \{ 0+2 \} = \{ 0, 2 \} \quad \text{// for each sum in } L_0, \text{ we can add } x_1 \text{ to it or not} \)

\( L_2 = \{ 0, 2 \} \cup \{ 0+7, 2+7 \} = \{ 0, 2, 7, 9 \} \)

\( L_3 = \{ 0, 2, 7, 9 \} \cup \{ 0+5, 2+5, 7+5, 9+5 \} = \{ 0, 2, 5, 7, 9, 12, 14 \} \)
Generating subset sums

- \( X = \{ x_1, x_2, \ldots, x_n \} \)
- **choices:** do we take \( x_1 \) into subset? do we take \( x_2 \) into subset? ETC

```
Generate-Subsets (X, t)
1. \( L_0 \leftarrow \{ 0 \} \)
2. for \( i \leftarrow 1 \) to \( n \)
3.   do Copy \( L_{i-1} \) to list \( M_{i-1} \)
4.   Add \( x_i \) to each number in \( M_{i-1} \) and remove numbers larger than \( t \)
5. \( L_i \leftarrow L_{i-1} \cup M_{i-1} \)
6. Remove duplicates from \( L_i \)
7. return \( L_n \)
```

**running time:** \( \Omega \) (size of list \( L_n \)) = \( \Omega(2^n) \)
Turning the exponential-time exact algorithm into an FPTAS

Speeding up the algorithm: don’t keep all numbers in $L_i$ \( ( \trim L_i ) \)

$L_i = 0, 104, 134, 135, 145, 149, 204, 207, 224, 300$

if we keep only one of them, we make only a small mistake

if numbers $y$ and $z$ in $L_i$ are almost the same, then throw away one of them

when exactly can we afford to do this? throw away $y$ when \( 1 \leq y / z \leq 1 + \delta \) for a suitable parameter $\delta$

which one? safer to keep the smaller number, let’s say $z$
Trim \(( L, \delta )\)

// \( L \) is a sorted list with elements \( z_0, z_1, z_2, \ldots, z_m \) (in order) \( \text{NB} \ z_0 = 0 \)

\begin{itemize}
  \item last \( \leftarrow z_1 \)
  \item for \( j \leftarrow 2 \) to \( m \)
    \begin{itemize}
      \item do if \( z_j / \text{last} \leq 1 + \delta \)
        \begin{itemize}
          \item then remove \( z_j \) from \( L \) // trim
          \item else last \( \leftarrow z_j \) // don’t trim
        \end{itemize}
    \end{itemize}
\end{itemize}

Generate-Trimmed-Subsets \(( X, t )\)

1. \( L_0 \leftarrow \text{empty set} \)
2. for \( i \leftarrow 1 \) to \( n \)
3. \( \text{do} \) Copy \( L_{i-1} \) to list \( M_{i-1} \)
4. Add \( x_i \) to each number in \( M_{i-1} \) and remove numbers larger than \( t \)
5. \( L_i \leftarrow L_{i-1} \cup M_{i-1} \)
6. Remove duplicates from \( L_i \) \( \text{Trim} \ (L_i, \delta) \)
7. return \( L_n \)
Lemma. The size of any list $L_i$ is $O\left(\frac{1}{\delta} \cdot \log t\right)$

Proof. Let $L = z_0, z_1, z_2, \ldots, z_m$ (in order). Then $\frac{z_j}{z_{j-1}} \geq 1+\delta$,

so $z_m \geq (1+\delta) \cdot z_{m-1}$

$\geq (1+\delta)^2 \cdot z_{m-2}$

$\geq \ldots$

$\geq (1+\delta)^{m-1} \cdot z_1$

$\geq (1+\delta)^{m-1}$

Hence, $(1+\delta)^{m-1} \leq z_m \leq t$, so $(m - 1) \cdot \log (1+\delta) \leq \log t$

Conclusion: $m = O\left(\frac{1}{\delta} \cdot \log t\right)$ since $\log(1 + \delta) \approx \delta$ for small $\delta$
Lemma. If we choose $\delta = \epsilon/(2n)$, then approximation ratio is $1+\epsilon$. 

Proof. $L_i = \text{list we maintain}$  
$U_i = \text{list we would maintain if we would never do any trimming}$

Claim: For any $y$ in $U_i$ there is $z$ in $L_i$ such that $z \leq y \leq (1+\delta)^i \cdot z$

Proof of Claim. By induction. ☐

OPT = value of optimal solution.

Then OPT is max number in $U_n$, so there is $z$ in $L_i$ with

$$z \geq \left( \frac{1}{(1+\delta)^n} \right) \cdot \text{OPT}$$

$$= \left( \frac{1}{(1+\epsilon/(2n))^n} \right) \cdot \text{OPT}$$

$$\geq \left( \frac{1}{(1+\epsilon)} \right) \cdot \text{OPT}$$
Lemma. If we choose $\delta = \varepsilon/(2n)$, then approximation ratio is $1+\varepsilon$.

Lemma. The size of any list $L_i$ is $O\left(\frac{1}{\delta} \cdot \log t\right)$.

Plug in $\delta = \varepsilon/(2n)$: size of $L_i$ is $O\left(\frac{n}{\varepsilon} \cdot \log t\right)$

Need $\log t$ bits to represent $t$

$\Rightarrow$ size of $L_i$ polynomial in input size and $1/\varepsilon$

Running time is $O\left(\sum_i |L_i|\right)$

$\Rightarrow$ running time polynomial in input size and $1/\varepsilon$

Theorem. There is an FPTAS for the Subset Sum optimization problem.
Summary

- **ρ-approximation algorithm:**
  algorithm for which computed solution is within factor ρ from OPT

- to prove approximation ratio we usually need lower bound on OPT
  (or, for maximization, upper bound)

- **PTAS** = polynomial-time approximation scheme
  = algorithm with two parameters, input instance and ε > 0, such that
  - approximation ratio is $1 + \varepsilon$
  - running time is polynomial in $n$ for constant $\varepsilon$

- **FPTAS** = PTAS whose running time is polynomial in $1/\varepsilon$

- some problems are even hard to approximate