Complexity Classes for Optimization Problems

Stefan Kugele

Technical University of Munich

Joint Bavarian Swiss International School 2004, Binntal
"I can’t find an efficient algorithm. I guess I’m just to dumb"
"I can’t find an efficient algorithm, because no such algorithm is possible!"

A famous cartoon by Garey & Johnson, 1979
"I can’t find an efficient algorithm, but neither can all these famous people."
Why using approximation?

Question

Why using approximation?

Answer

- We are not able to solve \( \mathcal{NP} \)-complete problems efficiently, that is, there is no known way to solve them in polynomial time unless \( \mathcal{P} = \mathcal{NP} \).
- Why not looking for an approximate solution?
Why using approximation?

Question
Why using approximation?

Answer
- We are not able to solve \( \mathcal{NP} \)-complete problems efficiently, that is, there is no known way to solve them in polynomial time unless \( \mathcal{P} = \mathcal{NP} \).
- Why not looking for an approximate solution?
Why using approximation?

Question

Why using approximation?

Answer

- We are not able to solve \( \mathcal{NP} \)-complete problems efficiently, that is, there is no known way to solve them in polynomial time unless \( \mathcal{P} = \mathcal{NP} \).
- Why not looking for an approximate solution?
Two basic principles

- Algorithm design
  - Sequential algorithms
  - Greedy approach
  - Local search
  - Linear programming (LP)
  - Dynamic programming (DP)
  - Randomized algorithms
- Complexity classes
  - That's what we are dealing with today
Two basic principles

- **Algorithm design**
  - Sequential algorithms
  - Greedy approach
  - Local search
  - Linear programming (LP)
  - Dynamic programming (DP)
  - Randomized algorithms

- **Complexity classes**
  - That’s what we are dealing with today
Two basic principles

- Algorithm design
  - Sequential algorithms
  - Greedy approach
  - Local search
  - Linear programming (LP)
  - Dynamic programming (DP)
  - Randomized algorithms

- Complexity classes

That's what we are dealing with today
Two basic principles

- Algorithm design
  - Sequential algorithms
    - Greedy approach
    - Local search
    - Linear programming (LP)
    - Dynamic programming (DP)
    - Randomized algorithms

- Complexity classes
  - That's what we are dealing with today
Two basic principles

- **Algorithm design**
  - Sequential algorithms
  - Greedy approach
  - Local search
  - Linear programming (LP)
  - Dynamic programming (DP)
  - Randomized algorithms

- **Complexity classes**

That's what we are dealing with today
Two basic principles

- Algorithm design
  - Sequential algorithms
  - Greedy approach
  - Local search
    - Linear programming (LP)
    - Dynamic programming (DP)
    - Randomized algorithms

- Complexity classes

That's what we are dealing with today
Two basic principles

- Algorithm design
  - Sequential algorithms
  - Greedy approach
  - Local search
  - Linear programming (LP)
    - Dynamic programming (DP)
    - Randomized algorithms

- Complexity classes

  That’s what we are dealing with today
Two basic principles

- Algorithm design
  - Sequential algorithms
  - Greedy approach
  - Local search
  - Linear programming (LP)
  - Dynamic programming (DP)
  - Randomized algorithms

- Complexity classes

That’s what we are dealing with today
Two basic principles

- Algorithm design
  - Sequential algorithms
  - Greedy approach
  - Local search
  - Linear programming (LP)
  - Dynamic programming (DP)
  - Randomized algorithms

- Complexity classes

That’s what we are dealing with today
Two basic principles

- Algorithm design
  - Sequential algorithms
  - Greedy approach
  - Local search
  - Linear programming (LP)
  - Dynamic programming (DP)
  - Randomized algorithms

- Complexity classes
  - That's what we are dealing with today
**Definition**

Optimization Problem

\[ O = (I, SOL, m, type) \]

- \( I \) the instance set
- \( SOL(i) \) the set of feasible solutions for instance \( i \) (\( SOL(i) \) for \( i \in I \))
- \( m(i, s) \) the measure of solution \( s \) with respect to instance \( i \) (positive integer for \( i \in I \) and \( s \in SOL(i) \))
- \( type \in \{\text{min, max}\} \)

\[ \text{opt}(i) = \text{type} \cdot m(i, s) \]

\[ s \in SOL(i) \]
An tight example

Example

Given is a knapsack with capacity $C$ and a set of items $S = \{1, 2, \ldots, n\}$, where item $i$ has weight $w_i$ and value $v_i$.

Problem

The problem is to find a subset $T \subseteq S$ that maximizes the value of $\sum_{i \in T} v_i$ given that $\sum_{i \in T} w_i \leq C$; that is all the items fit in the knapsack with capacity $C$.

- All set $T \subseteq S : \sum_{i \in T} w(i) \leq C$ are feasible solutions.
- $\sum_{i \in T} v_i$ is the quality of the solution $T$ with respect to instance $i$. 
An tight example

Example
Given is a knapsack with capacity $C$ and a set of items $S = \{1, 2, \ldots, n\}$, where item $i$ has weight $w_i$ and value $v_i$.

Problem
The problem is to find a subset $T \subseteq S$ that maximizes the value of $\sum_{i \in T} v_i$ given that $\sum_{i \in T} w_i \leq C$; that is all the items fit in the knapsack with capacity $C$.

- All set $T \subseteq S : \sum_{i \in T} w(i) \leq C$ are feasible solutions.
- $\sum_{i \in T} v_i$ is the quality of the solution $T$ with respect to instance $i$. 
An tight example

**Example**

Given is a knapsack with capacity $C$ and a set of items $S = \{1, 2, \ldots, n\}$, where item $i$ has weight $w_i$ and value $v_i$.

**Problem**

The problem is to find a subset $T \subseteq S$ that maximizes the value of $\sum_{i \in T} v_i$ given that $\sum_{i \in T} w_i \leq C$; that is all the items fit in the knapsack with capacity $C$.

- All set $T \subseteq S : \sum_{i \in T} w(i) \leq C$ are feasible solutions.
- $\sum_{i \in T} v_i$ is the quality of the solution $T$ with respect to instance $i$. 
An tight example (cont.)

**Instance**

Knapsack = (I, SOL, m, max)

\[ I = \{(S, w, C, v) \mid S = \{1, \ldots, n\}, \ w, v : S \to \mathbb{N}\} \]

\[ SOL(i) = \left\{ T \subseteq S : \sum_{i \in T} w(i) \leq C \right\} \]

\[ m(i, s) = \sum_{i \in T} v(i) \]
Outline

1. Approximation algorithms and errors

2. Classes
   - NPO
   - APX
   - PTAS and FPTAS
   - F – APX
   - Negative Results

3. Outlook
   - AP-Reductions
   - MaxSNP
Outline

1. Approximation algorithms and errors
   - Classes
     - NPO
     - APX
     - PTAS and FPTAS
     - F − APX
     - Negative Results

2. Outlook
   - AP-Reductions
   - MaxSNP
Approximation Algorithm

**Definition**

Given an optimization problem \( O = (I, SOL, m, type) \), an algorithm \( A \) is an approximation algorithm for \( O \) if, for any given instance \( i \in I \), it returns an approximate solution, that is a feasible solution \( A(i) \in SOL(i) \) with certain properties.

**Question**

But what is an approximate solution?

**Answer**

A solution whose value is "not too far" from the optimum.

What's the absolute error we make by approximating the solution?
Approximation Algorithm

Definition

Given an optimization problem \( O = (I, SOL, m, type) \), an algorithm \( A \) is an approximation algorithm for \( O \) if, for any given instance \( i \in I \), it returns an approximate solution, that is a feasible solution \( A(i) \in SOL(i) \) with certain properties.

Question

But what is an approximate solution?

Answer

A solution whose value is "not too far" from the optimum.
Definition

Given an optimization problem \( O = (I, SOL, m, type) \), an algorithm \( A \) is an approximation algorithm for \( O \) if, for any given instance \( i \in I \), it returns an approximate solution, that is a feasible solution \( A(i) \in SOL(i) \) with certain properties.

Question

But what is an approximate solution?

Answer

A solution whose value is "not too far" from the optimum.
Definition

Given an optimization problem \( O = (I, SOL, m, \text{type}) \), an algorithm \( A \) is an approximation algorithm for \( O \) if, for any given instance \( i \in I \), it returns an approximate solution, that is a feasible solution \( A(i) \in SOL(i) \) with certain properties.

Question

But what is an approximate solution?

Answer

A solution whose value is "not too far" from the optimum.

What’s the absolute error we make by approximating the solution?
Absolute error

**Definition**

Given an optimization problem $O$, for any instance $i \in I$ and for any feasible solution $s$ of $i$, the **absolute error** for $s$ with respect to $i$ is defined as:

$$D(i, s) = |m^*(i) - m(i, s)|$$

where $m^*(i)$ denotes the measure of the optimal solution of instance $i$ and $m(i, s)$ denotes the measure of solution $s$. 
Absolute approximation algorithm

**Definition**

Given an optimization problem $O$ and an approximation algorithm $A$ for $O$, we say that $A$ is an **absolute approximation algorithm** if there exists a constant $k$ such that, for every instance $i$ of $O$,

$$D(i, A(i)) \leq k$$

To express the quality of an approximate solution, commonly used notations are:

- the relative error
- the performance ratio
Absolute approximation algorithm

Definition

Given an optimization problem $O$ and an approximation algorithm $A$ for $O$, we say that $A$ is an **absolute approximation algorithm** if there exists a constant $k$ such that, for every instance $i$ of $O$,

$$D(i, A(i)) \leq k$$

To express the quality of an approximate solution, commonly used notations are:

- the relative error
- the performance ratio
Relative error

Definition
Given an optimization problem $O$, for any instance $i$ of $O$ and for any feasible solution $s$ of $i$, the relative error with respect to $i$ is defined as

$$E(i,s) = \frac{|m^*(i) - m(i,s)|}{\max \{m^*(i), m(i,s)\}}$$

For both, maximization and minimization problems, the relative error is equal to 0 when the solution obtained is optimal, and becomes close to 1 when the approximate solution is very poor.
Definition

Given an optimization problem $O$ and an approximation algorithm $A$ for $O$, we say that $A$ is an $\epsilon$–approximate algorithm for $O$ if, given any input instance $i$ of $O$, the relative error of the approximate solution $A(i)$ provided by algorithm $A$ is bounded by $\epsilon$, that is

$$E(i, A(i)) \leq \epsilon$$

Different measure

Alternatively, the quality can be expressed by means of a different, but related, measure.
Definition

Given an optimization problem $O$ and an approximation algorithm $A$ for $O$, we say that $A$ is an $\epsilon$–approximate algorithm for $O$ if, given any input instance $i$ of $O$, the relative error of the approximate solution $A(i)$ provided by algorithm $A$ is bounded by $\epsilon$, that is

$$E(i, A(i)) \leq \epsilon$$

Different measure

Alternatively, the quality can be expressed by means of a different, but related, measure.
Performance ratio

Definition

Given an optimization problem $O$, for any instance $i$ of $O$ and for any feasible solution $s$ of $i$, the performance ratio of $s$ with respect to $i$ is defined as

$$R(i, s) = \max \left\{ \frac{m(i, s)}{m^*(i)}, \frac{m^*(i)}{m(i, s)} \right\}$$

For both, minimization and maximization, the value of the performance ratio is equal to 1 in the case of an optimal solution, and can assume arbitrarily large values in the case of an poor approximate solution.
Definition

Given an optimization problem \( O \) and an approximation algorithm \( A \) for \( O \), we say that \( A \) is an \( r \)-approximate algorithm for \( O \), given any input instance \( i \) of \( O \), the performance ratio of the approximate solution \( A(i) \) is bounded by \( r \), that is

\[ R(i, A(i)) \leq r \]

Relationship

\[
E(i, s) = 1 - \frac{1}{R(i, s)}
\]

\[
R(i, s) = -\frac{1}{E(i, s) - 1}
\]
**r–approximate algorithm**

**Definition**

Given an optimization problem $O$ and an approximation algorithm $A$ for $O$, we say that $A$ is an $r$–approximate algorithm for $O$, given any input instance $i$ of $O$, the performance ratio of the approximate solution $A(i)$ is bounded by $r$, that is

$$R(i, A(i)) \leq r$$

**Relationship**

$$E(i, s) = 1 - \frac{1}{R(i, s)}$$

$$R(i, s) = -\frac{1}{E(i, s) - 1}$$
Example: $E(i, s), R(i, s)$ for Minimum Vertex Cover

approx. solution

optimal solution

→ Flipchart
Outline

1. Approximation algorithms and errors

2. Classes
   - NPO
   - APX
   - PTAS and FPTAS
   - F − APX
   - Negative Results

3. Outlook
   - AP-Reductions
   - MaxSNP
The class $\mathcal{NPO}$

**Definition**

$\mathcal{NPO}$ is the class of optimization problems whose decision versions are in $\mathcal{NP}$.

$O = (I, SOL, m, type) \in \mathcal{NPO}$ iff

- $\exists$ polynomial $p : \forall i \in I, s \in SOL(i) : |s| \leq p(|i|)$
- deciding $s \in SOL(i)$ is in $\mathcal{P}$
- computing $m(s, i)$ is in $\mathcal{FP}$
The class $\mathcal{NPO}$

**Definition**

$\mathcal{NPO}$ is the class of optimization problems whose decision versions are in $\mathcal{NP}$.

$O = (I, SOL, m, \text{type}) \in \mathcal{NPO}$ iff

- \exists \text{ polynomial } p : \forall i \in I, s \in SOL(i) : |s| \leq p(|i|)$
- deciding $s \in SOL(i)$ is in $\mathcal{P}$
- computing $m(s, i)$ is in $\mathcal{FP}$
The class \( \mathcal{NPO} \)

**Definition**

\( \mathcal{NPO} \) is the class of optimization problems whose decision versions are in \( \mathcal{NP} \).

\[ O = (I, \text{SOL}, m, \text{type}) \in \mathcal{NPO} \iff \exists \text{ polynomial } p : \forall i \in I, s \in \text{SOL}(i) : |s| \leq p(|i|) \]

- deciding \( s \in \text{SOL}(i) \) is in \( \mathcal{P} \)
- computing \( m(s, i) \) is in \( \mathcal{FP} \)
The class \( \mathcal{NPO} \)

**Definition**

\( \mathcal{NPO} \) is the class of optimization problems whose decision versions are in \( \mathcal{NP} \).

\[ O = (I, SOL, m, \text{type}) \in \mathcal{NPO} \text{ iff} \]

- \( \exists \) polynomial \( p \) : \( \forall i \in I, \ s \in SOL(i) : |s| \leq p(|i|) \)
- deciding \( s \in SOL(i) \) is in \( \mathcal{P} \)
- computing \( m(s, i) \) is in \( \mathcal{FP} \)
The class $\text{APX}$

**Definition**

$\text{APX}$ is the class of all $\text{NPO}$ problems such that, for some $r \geq 1$, there exists a polynomial-time $r$-approximate algorithm for $O$.

**Inclusions**

$\text{APX} \subset \text{NPO} \iff P \neq \text{NP}$

**Example**

MinVertexCover, MaxSat, MaxKnapsack, MaxCut, MaxBinPacking, MaxPlanarGraphColoring
The class $\text{APX}$

**Definition**

$\text{APX}$ is the class of all $\text{NPO}$ problems such that, for some $r \geq 1$, there exists a polynomial-time $r$-approximate algorithm for $O$.

**Inclusions**

$\text{APX} \subset \text{NPO} \iff P \neq \text{NP}$

**Example**

MinVertexCover, MaxSat, MaxKnapsack, MaxCut, MaxBinPacking, MaxPlanarGraphColoring
The class $\text{APX}$

**Definition**

$\text{APX}$ is the class of all $\text{NPO}$ problems such that, for some $r \geq 1$, there exists a polynomial-time $r$-approximate algorithm for $O$.

**Inclusions**

$\text{APX} \subset \text{NPO} \iff \mathcal{P} \neq \text{NP}$

**Example**

MinVertexCover, MaxSat, MaxKnapsack, MaxCut, MaxBinPacking, MaxPlanarGraphColoring
The class $\mathcal{APX}$ (cont.)

Proof.

- Idee: TSP can not be $r$-approximated, no matter how large is the performance ratio $r$.
- Reduction from the $\mathcal{NP}$-complete HamiltonianCircuit decision problem.
- Let $G = (V, E)$ be an instance of HC with $|V| = n$.
- Construct for any $r \geq 1$ a MinTSP instance such that if we had a poly-time $r$-approximate algorithms for MinTSP, then we could decide whether the graph $G$ has a HC in polynomial time.
The class \( \text{APX} \) (cont.)

Proof.

- Idee: TSP can not be \( r \)-approximated, no matter how large is the performance ratio \( r \).
- Reduction from the \( \mathcal{NP} \)-complete HamiltonianCircuit decision problem.
  - Let \( G = (V, E) \) be an instance of HC with \( |V| = n \).
  - Construct for any \( r \geq 1 \) a MinTSP instance such that if we had a poly-time \( r \)-approximate algorithms for MinTSP, then we could decide whether the graph \( G \) has a HC in polynomial time.
The class $\text{APX}$ (cont.)

Proof.

- Idee: TSP can not be $r$-approximated, no matter how large is the performance ratio $r$.
- Reduction from the $\mathcal{NP}$-complete HamiltonianCircuit decision problem.
- Let $G = (V, E)$ be an instance of HC with $|V| = n$.
  - Construct for any $r \geq 1$ a MinTSP instance such that if we had a poly-time $r$-approximate algorithms for MinTSP, then we could decide whether the graph $G$ has a HC in polynomial time.
The class \( APX \) (cont.)

**Proof.**

- Idee: TSP can not be \( r \)-approximated, no matter how large is the performance ratio \( r \).
- Reduction from the \( NP \)-complete Hamiltonian Circuit decision problem.
- Let \( G = (V, E) \) be an instance of HC with \( |V| = n \).
- Construct for any \( r \geq 1 \) a MinTSP instance such that if we had a poly-time \( r \)-approximate algorithms for MinTSP, then we could decide whether the graph \( G \) has a HC in polynomial time.
The class $\mathcal{APX}$ (cont.)

Proof (cont.)

- The instance of MinTSP is defined on the same set of nodes $V$ and with distances:

  $$d(v_i, v_j) = \begin{cases} 
  1 & \text{if } (v_i, v_j) \in E \\
  1 + nr & \text{otherwise.}
  \end{cases}$$

- This instance of MinTSP has a solution of measure $n$ iff $G$ has a HC.

- The next smallest approximate solution has measure at least $n(1 + r)$ ($n - 1 + (1 + nr) = n + (nr) = n(1 + r)$) and the performance ratio is hence greater than $r$. 

Stefan Kugele  Complexity Classes for Optimization Problems
The class $\text{APX}$ (cont.)

Proof (cont.)

- The instance of MinTSP is defined on the same set of nodes $V$ and with distances:

$$d(v_i, v_j) = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 1 + nr & \text{otherwise.} \end{cases}$$

- This instance of MinTSP has a solution of measure $n$ iff $G$ has a HC.

- The next smallest approximate solution has measure at least $n(1 + r)$ ($(n - 1 + (1 + nr) = n + (nr) = n(1 + r)$) and the performance ratio is hence greater than $r$. 

Stefan Kugele
Complexity Classes for Optimization Problems
The class $\textbf{APX}$ (cont.)

Proof (cont.)

- The instance of MinTSP is defined on the same set of nodes $V$ and with distances:
  \[
  d(v_i, v_j) = \begin{cases} 
  1 & \text{if } (v_i, v_j) \in E \\
  1 + nr & \text{otherwise.}
  \end{cases}
  \]

- This instance of MinTSP has a solution of measure $n$ iff $G$ has a HC.

- The next smallest approximate solution has measure at least $n(1 + r) \left((n - 1 + (1 + nr) = n + (nr) = n(1 + r)\right)$ and the performance ratio is hence greater than $r$. 
If $G$ has no HC, then the optimal solution has measure at least $n(1 + r)$.

Therefore, if we had a polynomial $r$-approximate algorithm for MinTSP, we could use it to decide whether $G$ has a HC in the following way: apply the approximation algorithm to the instance of MinTSP and answer YES iff it returns a solution of measure $n$. 
The class $\text{APX}$ (cont.)

Proof (cont.)

- If $G$ has no HC, then the optimal solution has measure at least $n(1 + r)$.
- Therefore, if we had a polynomial $r$-approximate algorithm for MinTSP, we could use it to decide whether $G$ has a HC in the following way: apply the approximation algorithm to the instance of MinTSP and answer YES iff it returns a solution of measure $n$. 
Example: MinimumVertexCover

Example (MinimumVertexCover)

**Instance:** Graph $G = (V, E)$

**Query:** Smallest vertex cover

**Theorem:** MinimumVertexCover is 2-approximatable, that is $\text{MinimumVertexCover} \in \text{APX}$

**Proof:** The corresponding decision problem is $\text{NP}$-complete
Example: MinimumVertexCover

Example (MinimumVertexCover)

- **Instance:** Graph $G = (V, E)$
- **Query:** Smallest vertex cover
- **Theorem:** MinimumVertexCover is 2-approximatable, that is, $\text{MinimumVertexCover} \in \text{APX}$
- **Proof:** The corresponding decision problem is $\text{NP}$-complete.
Example: MinimumVertexCover

Example (MinimumVertexCover)

**Instance:** Graph $G = (V, E)$

**Query:** Smallest vertex cover

**Theorem:** MinimumVertexCover is 2-approximatable, that is $\text{MinimumVertexCover} \in \text{APX}$

**Proof:** The corresponding decision problem is $\mathcal{NP}$-complete.
Example: MinimumVertexCover

**Example (MinimumVertexCover)**

**Instance:** Graph $G = (V, E)$

**Query:** Smallest vertex cover

**Theorm:** MinimumVertexCover is 2-approximatable, that is $\text{MinimumVertexCover} \in \text{APX}$

**Proof:** The corresponding decision problem is $\mathcal{NP}$-complete
Example: MinimumVertexCover (cont.)

Algorithm MinimumVertexCover

procedure VertexCover-2-Approx($V, E$)
  while $E \neq \emptyset$ do
    pick an arbitrary edge $\{u, v\} \in E$
    add $u$ and $v$ to the vertex cover
    delete all edges covered by $u$ and $v$ from $E$
  end while
end procedure
Example: MinimumVertexCover (cont.)

Algorithm MinimumVertexCover

while $E \neq \emptyset$ do
    pick an arbitrary edge $\{u, v\} \in E$
    add $u$ and $v$ to the vertex cover
    delete edges covered by $u$ and $v$ from $E$
end while

Trace

$VC = \{\}$
$E = \{(A, G), (A, E), (A, D), (A, C), (B, G), (B, F), (B, D), (B, C), (D, G), (D, F)\}$
Example: MinimumVertexCover (cont.)

Algorithm MinimumVertexCover

\[
\text{while } E \neq \emptyset \text{ do }
\begin{align*}
&\text{pick an arbitrary edge } \{u, v\} \in E \\
&\text{add } u \text{ and } v \text{ to the vertex cover} \\
&\text{delete edges covered by } u \text{ and } v \text{ from } E
\end{align*}
\]

end while

Trace

\[
VC = \{\}
\]

\[
E = \{(A, G), (A, E), (A, D), (A, C), (B, G), (B, F), (B, D), (B, C), (D, G), (D, F)\}
\]
Example: MinimumVertexCover (cont.)

Algorithm MinimumVertexCover

while $E \neq \emptyset$ do
  pick an arbitrary edge $\{u, v\} \in E$
  add $u$ and $v$ to the vertex cover
  delete edges covered by $u$ and $v$ from $E$
end while

Trace

$VC = \{A, D\}$
$E = \{(A, G), (A, E), (A, D), (A, C), (B, G), (B, F), (B, D), (B, C), (D, G), (D, F)\}$
Example: MinimumVertexCover (cont.)

Algorithm MinimumVertexCover

while $E \neq \emptyset$ do
    pick an arbitrary edge $\{u, v\} \in E$
    add $u$ and $v$ to the vertex cover
    delete edges covered by $u$ and $v$ from $E$
end while

Trace

$VC = \{A, D\}$

$E = \{(A, G), (A, E), (A, D), (A, C),
(B, G), (B, F), (B, D), (B, C),
(D, G), (D, F)\}$
Example: MinimumVertexCover (cont.)

Algorithm MinimumVertexCover

while $E \neq \emptyset$ do
    pick an arbitrary edge $\{u, v\} \in E$
    add $u$ and $v$ to the vertex cover
    delete edges covered by $u$ and $v$ from $E$
end while

Trace

$VC = \{A, D\}$
$E = \{(B, G), (B, F), (B, C)\}$
Example: MinimumVertexCover (cont.)

Algorithm MinimumVertexCover

while $E \neq \emptyset$ do
    pick an arbitrary edge $\{u, v\} \in E$
    add $u$ and $v$ to the vertex cover
    delete edges covered by $u$ and $v$ from $E$
end while

Trace

$VC = \{A, D, B, G\}$
$E = \{(B, G), (B, F), (B, C)\}$
Example: MinimumVertexCover (cont.)

Algorithm MinimumVertexCover

while $E \neq \emptyset$ do
  pick an arbitrary edge $\{u, v\} \in E$
  add $u$ and $v$ to the vertex cover
  delete edges covered by $u$ and $v$ from $E$
end while

Trace

$VC = \{A, D, B, G\}$

$E = \{(B, G), (B, F), (B, C)\}$
Example: MinimumVertexCover (cont.)

Algorithm MinimumVertexCover

\[\text{while } E \neq \emptyset \text{ do} \]
\[\quad \text{pick an arbitrary edge } \{u, v\} \in E \]
\[\quad \text{add } u \text{ and } v \text{ to the vertex cover} \]
\[\quad \text{delete edges covered by } u \text{ and } v \text{ from } E \]
\[\text{end while}\]

Trace

\[\text{VC} = \{A, D, B, G\} \quad \leftarrow 2 - \text{approx.result} \]
\[E = \{\} \]
Example: Minimum Vertex Cover (cont.)

approx. solution

optimal solution
Example: MinimumVertexCover (cont.)

Question

The result is a vertex cover but is its size maximum twice the optimum?

Answer

Yes. No two edges chosen by the algorithm have shared nodes. Hence, a vertex cover of only those edges has to contain at least either of them, i.e. be at least half of the size of the found vertex cover.
Example: MinimumVertexCover (cont.)

**Question**

The result is a vertex cover but is its size maximum twice the optimum?

**Answer**

Yes. No two edges chosen by the algorithm have shared nodes. Hence, a vertex cover of only those edges has to contain at least either of them, i.e. be at least half of the size of the found vertex cover.
Inclusions so far ($\mathcal{P} \neq \mathcal{NP}$)

Stefan Kugele
Complexity Classes for Optimization Problems
Limits to approximability: The gap technique

Theorem

Let \( O' \) be an NP-complete decision problem and let \( O \) be an \( NP \) minimization problem. Let us suppose that there exist two polynomial-time computable functions

\[
f : I_{O'} \rightarrow I_O \\
c : I_{O'} \rightarrow \mathbb{N}
\]

a constant gap > 0, such that for any instance \( i \) of \( O' \),

\[
m^*(f(i)) = \begin{cases} 
  c(i) & \text{if } i \text{ is a positive instance} \\
  c(i)(1 + \text{gap}) & \text{otherwise}.
\end{cases}
\]

Then no polynomial-time \( r \)-approximate algorithm for \( O \) with \( r < 1 + \text{gap} \) can exist, unless \( P = NP \).
Limits to approximability: *The gap technique* (cont.)

Proof.

→ Flipchart
Limits to approximability: The gap technique (cont.)

Example (1)

Given a planar graph decide whether this graph is 3-colorable. This problem is \( \mathcal{NP} \)-complete. But any planar graph can be colored with 4 colors.

Define \( f \) as the identity function: \( f(G) = G \) is a planar graph

- If \( G \) is 3-colorable, then \( m^*(f(G)) = 3 \)
- If \( G \) is not 3-colorable, then \( m^*(f(G)) = 4 = 3(1 + \frac{1}{3}) \)
- \( \text{gap} = \frac{1}{3} \)

Theorem

MinimumGraphColoring has no \( r \)-approximate algorithm with \( r < \frac{4}{3} \) unless \( \mathcal{P} = \mathcal{NP} \).
Limits to approximability: *The gap technique* (cont.)

**Example (1)**

Given a planar graph decide whether this graph is 3-colorable. This problem is \( \mathcal{NP} \)-complete. But any planar graph can be colored with 4 colors.

Define \( f \) as the identity function: \( f(G) = G \) is a planar graph

- If \( G \) is 3-colorable, then \( m^*(f(G)) = 3 \)
- If \( G \) is not 3-colorable, then \( m^*(f(G)) = 4 = 3(1 + \frac{1}{3}) \)
- \( \text{gap} = \frac{1}{3} \)

**Theorem**

MinimumGraphColoring has no \( r \)-approximate algorithm with \( r < \frac{4}{3} \) unless \( \mathcal{P} = \mathcal{NP} \).
Limits to approximability: *The gap technique* (cont.)

**Example (1)**

Given a planar graph decide whether this graph is 3-colorable. This problem is \( \mathcal{NP} \)-complete. But any planar graph can be colored with 4 colors.

Define \( f \) as the identity function: \( f(G) = G \) is a planar graph

- If \( G \) is 3-colorable, then \( m^*(f(G)) = 3 \)
- If \( G \) is not 3-colorable, then \( m^*(f(G)) = 4 = 3(1 + \frac{1}{3}) \)

\( \text{gap} = \frac{1}{3} \)

**Theorem**

MinimumGraphColoring has no \( r \)-approximate algorithm with \( r < \frac{4}{3} \) unless \( \mathcal{P} = \mathcal{NP} \).
Limits to approximability: *The gap technique (cont.)*

**Example (1)**

Given a planar graph decide whether this graph is 3-colorable. This problem is \( NP \)-complete. But any planar graph can be colored with 4 colors.

Define \( f \) as the identity function: \( f(G) = G \) is a planar graph

- If \( G \) is 3-colorable, then \( m^*(f(G)) = 3 \)
- If \( G \) is not 3-colorable, then \( m^*(f(G)) = 4 = 3(1 + \frac{1}{3}) \)
- \( gap = \frac{1}{3} \)

**Theorem**

 MinimumGraphColoring has no \( r \)-approximate algorithm with \( r < \frac{4}{3} \) unless \( \mathcal{P} = \mathcal{NP} \).
Limits to approximability: *The gap technique* (cont.)

**Example (1)**

Given a planar graph decide whether this graph is 3-colorable. This problem is \( \mathcal{NP} \)-complete. But any planar graph can be colored with 4 colors.

Define \( f \) as the identity function: \( f(G) = G \) is a planar graph

- If \( G \) is 3-colorable, then \( m^*(f(G)) = 3 \)
- If \( G \) is not 3-colorable, then \( m^*(f(G)) = 4 = 3(1 + \frac{1}{3}) \)
- \( \text{gap} = \frac{1}{3} \)

**Theorem**

MinimumGraphColoring has no \( r \)-approximate algorithm with \( r < \frac{4}{3} \) unless \( \mathcal{P} = \mathcal{NP} \).
Limits to approximability: *The gap technique* (cont.)

Example (2)

MinimumBinPacking → Flipchart
Definition

Let $O$ be an $\mathcal{NP}$ problem. An algorithm $A$ is said to be a polynomial time approximation scheme ($\mathcal{PTAS}$) for $O$ if, for any instance $i$ of $O$ and any rational value $r > 1$, $A$ when applied to input $(i, r)$ returns an $r$-approximate solution of $i$ in time polynomial in $|i|$. 

- The running time of a $\mathcal{PTAS}$ may also depend exponentially on $\frac{1}{r-1}$.
- The better the approximation, the larger may be the running time.
Polynomial-time approximation schemes (PTAS)

Definition

Let $O$ be an NPO problem. An algorithm $A$ is said to be a polynomial time approximation scheme (PTAS) for $O$ if, for any instance $i$ of $O$ and any rational value $r > 1$, $A$ when applied to input $(i, r)$ returns an $r$-approximate solution of $i$ in time polynomial in $|i|$.

- The running time of a PTAS may also depend exponentially on $\frac{1}{r-1}$
- The better the approximation, the larger may be the running time
Definition

Let $O$ be an $\mathcal{NP}$ problem. An algorithm $A$ is said to be a polynomial time approximation scheme (PTAS) for $O$ if, for any instance $i$ of $O$ and any rational value $r > 1$, $A$ when applied to input $(i, r)$ returns an $r$-approximate solution of $i$ in time polynomial in $|i|$.

- The running time of a PTAS may also depend exponentially on $\frac{1}{r-1}$.
- The better the approximation, the larger may be the running time.
The class $\textbf{PTAS}$

**Definition**

$\textbf{PTAS}$ is the class of $\textbf{NPO}$ problems that admit a polynomial-time approximation scheme.

**Example**

MaxIntegerKnapsack, MaxIndependentSet (for planar graphs)

**Inclusions**

$\textbf{PTAS} \subset \textbf{APX} \iff P \neq NP$

In some cases, the increase in the running time of the approximation scheme with the degree of approximation may prevent any practical use of the scheme.
The class \( \text{PTAS} \)

**Definition**

\( \text{PTAS} \) is the class of \( \text{NP} \)O problems that admit a polynomial-time approximation scheme.

**Example**

MaxIntegerKnapsack, MaxIndependentSet (for planar graphs)

**Inclusions**

\[ \text{PTAS} \subset \text{APX} \iff \mathbb{P} \neq \mathbb{NP} \]

In some cases, the increase in the running time of the approximation scheme with the degree of approximation may prevent any practical use of the scheme.
The class $\text{PTAS}$

**Definition**

$\text{PTAS}$ is the class of $\text{NPO}$ problems that admit a polynomial-time approximation scheme.

**Example**

MaxIntegerKnapsack, MaxIndependentSet (for planar graphs)

**Inclusions**

$\text{PTAS} \subset \text{APX} \iff \mathcal{P} \neq \mathcal{NP}$

In some cases, the increase in the running time of the approximation scheme with the degree of approximation may prevent any practical use of the scheme.
The class \( \text{PTAS} \)

**Definition**

\( \text{PTAS} \) is the class of \( \text{NPO} \) problems that admit a polynomial-time approximation scheme.

**Example**

MaxIntegerKnapsack, MaxIndependentSet (for planar graphs)

**Inclusions**

\[ \text{PTAS} \subset \text{APX} \iff \mathcal{P} \neq \mathcal{NP} \]

In some cases, the increase in the running time of the approximation scheme with the degree of approximation may prevent any practical use of the scheme.
The class \( \mathcal{PTAS} \) (cont.)

Proof.

- Already done. MinimumBinPacking-example
- MinimumBinPacking has no \( r \)-approximate algorithm with \( r < \frac{3}{2} \) unless \( \mathcal{P} = \mathcal{NP} \).
- Therefore, unless \( \mathcal{P} = \mathcal{NP} \), MinimumBinPacking does not admit a \( \mathcal{PTAS} \).
Inclusions so far ($P \neq NP$)
A much better situation would arise when the running time is polynomial both in the size of the input and in the inverse of the performance ratio.

**Definition**

Let $O$ be an $\mathcal{NP}$ problem. An algorithm is said to be a fully polynomial time approximation scheme ($\text{FPTAS}$) for $O$ if, for any instance $i$ of $O$ and for any rational value $r > 1$, $A$ when applied to input $(i, r)$ returns an $r$-approximate solution of $i$ in time polynomial both in $|i|$ and $\frac{1}{(r-1)}$. 
Fully polynomial-time approximation scheme (FPTAS)

A much better situation would arise when the running time is polynomial both in the size of the input and in the inverse of the performance ratio.

**Definition**

Let $O$ be an $NP\bar{O}$ problem. An algorithm is said to be a fully polynomial time approximation scheme (FPTAS) for $O$ if, for any instance $i$ of $O$ and for any rational value $r > 1$, $A$ when applied to input $(i, r)$ returns an $r$-approximate solution of $i$ in time polynomial both in $|i|$ and $\frac{1}{(r-1)}$. 

Stefan Kugele
Complexity Classes for Optimization Problems
The class $\textsf{FPTAS}$

**Definition**

$\textsf{FPTAS}$ is the class of $\textsf{NPO}$ problems that admit a fully polynomial-time approximation scheme.

**Example**

MaximumKnapsack

**Inclusions**

$\textsf{FPTAS} \subset \textsf{PTAS} \iff \textsf{P} \neq \textsf{NP}$
The class **FPTAS**

**Definition**

**FPTAS** is the class of \( \mathcal{NPO} \) problems that admit a fully polynomial-time approximation scheme.

**Example**

MaximumKnapsack

**Inclusions**

\[ \text{FPTAS} \subset \text{PTAS} \iff \mathcal{P} \neq \mathcal{NP} \]
The class \textit{FPTAS} 

\textbf{Definition}

\textit{FPTAS} is the class of \textit{NPO} problems that admit a fully polynomial-time approximation scheme.

\textbf{Example}

MaximumKnapsack

\textbf{Inclusions}

\[ \text{FPTAS} \subset \text{PTAS} \iff \mathbb{P} \neq \mathbb{NP} \]
The class \textit{FPTAS} (cont.)

\begin{itemize}
\item Some hints:
  \begin{itemize}
  \item MaximumIndependentSet
  \item polynomially bounded
  \end{itemize}
\item Later, if you want to ;-)\end{itemize}
The class $FPTAS$ (cont.)

Proof.

- Some hints:
  - MaximumIndependentSet
    - polynomially bounded
  - Later, if you want to ;-)
The class $\mathcal{FPTAS}$ (cont.)

Proof.

- Some hints:
  - MaximumIndependentSet
  - polynomially bounded

- Later, if you want to ;-)
The class \textit{FPTAS} (cont.)

Proof.

- Some hints:
  - \texttt{MaximumIndependentSet}
  - polynomially bounded
- Later, if you want to ;-)
Inclusions so far ($P \neq NP$)
Definition
Let $O$ be an $\mathcal{NP}$ problem. $O$ is said to be in $\mathcal{F} - \mathcal{APX}$ if and only if there exists an $f$-approximation algorithm $A$ for $O$ which runs in polynomial-time for some function $f \in \mathcal{F}$.

Inclusions
\[ \text{FPTAS} \subset \text{PTAS} \subset \text{APX} \subset \text{log} - \text{APX} \subset \text{poly} - \text{APX} \subset \exp - \text{APX} \subset \mathcal{NP} \ L \equiv \mathcal{P} \neq \mathcal{NP} \]
**F – APX**

**Definition**
Let $O$ be an $\mathcal{NP}O$ problem. $O$ is said to be in $\mathcal{F} – \mathcal{APX}$ if and only if there exists an $f$-approximation algorithm $A$ for $O$ which runs in polynomial-time for some function $f \in \mathcal{F}$.

**Inclusions**
\[
\mathcal{FPTAS} \subset \mathcal{PTAS} \subset \mathcal{APX} \subset \log – \mathcal{APX} \subset \text{poly} – \mathcal{APX} \subset \text{exp} – \mathcal{APX} \subset \mathcal{NP}O \iff \mathcal{P} \neq \mathcal{NP}
\]
$\mathcal{F} - \mathcal{APX}$ (cont.)

Example

$\mathcal{APX}$ Max3Sat

$\log - \mathcal{APX}$ SetCover

$\text{poly} - \mathcal{APX}$ Coloring

$\exp - \mathcal{APX}$ TSP
$F - APX$ (cont.)

Example

$APX$ Max3Sat
$log-APX$ SetCover
$poly-APX$ Coloring
$exp-APX$ TSP
Example

- $\mathcal{APX}$: Max3Sat
- $\log - \mathcal{APX}$: SetCover
- $\text{poly} - \mathcal{APX}$: Coloring
- $\exp - \mathcal{APX}$: TSP
### Example

<table>
<thead>
<tr>
<th>Class</th>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{APX} )</td>
<td>Max3Sat</td>
</tr>
<tr>
<td>( \log \text{−APX} )</td>
<td>SetCover</td>
</tr>
<tr>
<td>( \text{poly−APX} )</td>
<td>Coloring</td>
</tr>
<tr>
<td>( \exp \text{−APX} )</td>
<td>TSP</td>
</tr>
</tbody>
</table>
Inclusions so far ($\mathcal{P} \neq \mathcal{NP}$)
Polynomially bounded optimization problems

**Definition**

An optimization problem is \textit{polynomially bounded} if there exists a polynomial $p$ such that, for any instance $i$ and for any $s \in SOL(i)$, $m(i, s) \leq p(|i|)$.

**Theorem**

No $NP$-hard polynomially bounded optimization problem belongs to the class $FPTAS$ unless $P = NP$.

**Example**

MaximumIndependentSet
Polynomially bounded optimization problems

**Definition**

An optimization problem is **polynomially bounded** if there exists a polynomial \( p \) such that, for any instance \( i \) and for any \( s \in SOL(i) \),
\[
m(i, s) \leq p(|i|).
\]

**Theorem**

No \( \mathcal{NP} \)-hard polynomially bounded optimization problem belongs to the class \( \mathcal{FPTAS} \) unless \( \mathcal{P} = \mathcal{NP} \).

**Example**

MaximumIndependentSet
Polynomially bounded optimization problems

**Definition**

An optimization problem is **polynomially bounded** if there exists a polynomial $p$ such that, for any instance $i$ and for any $s \in SOL(i)$, $m(i, s) \leq p(|i|)$.

**Theorem**

No $\mathcal{NP}$-hard polynomially bounded optimization problem belongs to the class $\mathcal{FPTAS}$ unless $\mathcal{P} = \mathcal{NP}$.

**Example**

MaximumIndependentSet
Polynomially bounded optimization problems (cont.)

Proof.

→ Flipchart
Pseudo-polynomial problem

**Definition**

An \( NPO \) problem \( O \) is **pseudo-polynomial** if it can be solved by an algorithm that, on any instance \( i \), runs in time bounded by a polynomial in \( |i| \) and in \( \max(i) \), where \( \max(i) \) denotes the value of the largest number occurring in \( i \).

**Theorem**

Let \( O \) be an \( NPO \) problem in \( FPTAS \). If a polynomial \( p \) exists such that, for every input \( i \), \( m^*(x) \leq p(|i|, \max(i)) \), then \( O \) is a pseudo-polynomial problem.

**Example**

MaximumKnapsack: \( \max(i) = \max\{a_1, \ldots, a_n, p_1, \ldots, p_n\} \)
Pseudo-polynomial problem

**Definition**

An $\mathcal{NP}$ problem $O$ is **pseudo-polynomial** if it can be solved by an algorithm that, on any instance $i$, runs in time bounded by a polynomial in $|i|$ and in $\max(i)$, where $\max(i)$ denotes the value of the largest number occurring in $i$.

**Theorem**

Let $O$ be an $\mathcal{NP}$ problem in $\mathcal{FPTAS}$. If a polynomial $p$ exists such that, for every input $i$, $m^*(x) \leq p(|i|, \max(i))$, then $O$ is a pseudo-polynomial problem.

**Example**

Maximum Knapsack: $\max(i) = \max\{a_1, \ldots, a_n, p_1, \ldots, p_n\}$
Pseudo-polynomial problem

**Definition**
An $\mathcal{NP}$ problem $O$ is **pseudo-polynomial** if it can be solved by an algorithm that, on any instance $i$, runs in time bounded by a polynomial in $|i|$ and in $\max(i)$, where $\max(i)$ denotes the value of the largest number occurring in $i$.

**Theorem**
Let $O$ be an $\mathcal{NP}$ problem in $\mathcal{FPTAS}$. If a polynomial $p$ exists such that, for every input $i$, $m^*(x) \leq p(|i|, \max(i))$, then $O$ is a pseudo-polynomial problem.

**Example**
**MaximumKnapsack**: \[ \max(i) = \max\{a_1, \ldots, a_n, p_1, \ldots, p_n\} \]
Let $O$ be an $\mathcal{NP\Omega}$ problem and let $p$ be a polynomial. We denote by $O^{\text{max},p}$ the problem obtained by restricting $O$ to only those instances $i$ which $\max(i) \leq p(|i|)$.

**Definition**

An $\mathcal{NP\Omega}$ problem $O$ is said to be strongly $\mathcal{NP}$-hard if a polynomial $p$ exists such that $O^{\text{max},p}$ is $\mathcal{NP}$-hard.
Let $O$ be an $\mathcal{NP}$ problem and let $p$ be a polynomial. We denote by $O^{\text{max},p}$ the problem obtained by restricting $O$ to only those instances $i$ which $\max(i) \leq p(|i|)$.

**Definition**

An $\mathcal{NP}$ problem $O$ is said to be strongly $\mathcal{NP}$-hard if a polynomial $p$ exists such that $O^{\text{max},p}$ is $\mathcal{NP}$-hard.
Theorem

If \( \mathcal{P} \neq \mathcal{NP} \), then no strongly \( \mathcal{NP} \)-hard problem can be pseudo-polynomial.

Proof.

→ Flipchart

From the last two theorems, the following result can be derived. Let \( O \) be a strongly \( \mathcal{NP} \)-hard problem that admits a polynomial \( p \) such that \( m^*(i) \leq p(|i|, \max(i)) \), for every input \( i \). If \( \mathcal{P} \neq \mathcal{NP} \), then \( O \) does not belong to the class \( \text{FPTAS} \).
If \( P \neq \mathcal{NP} \), then no strongly \( \mathcal{NP} \)-hard problem can be pseudo-polynomial.

Proof.

From the last two theorems, the following result can be derived. Let \( O \) be a strongly \( \mathcal{NP} \)-hard problem that admits a polynomial \( p \) such that \( m^*(i) \leq p(|i|, \max(i)) \), for every input \( i \). If \( P \neq \mathcal{NP} \), then \( O \) does not belong to the class \( \text{FPTAS} \).
Strongly \( \mathcal{NP} \)-hard problem (cont.)

**Theorem**

If \( \mathcal{P} \neq \mathcal{NP} \), then no strongly \( \mathcal{NP} \)-hard problem can be pseudo-polynomial.

**Proof.**

→ Flipchart

From the last two theorems, the following result can be derived. Let \( O \) be a strongly \( \mathcal{NP} \)-hard problem that admits a polynomial \( p \) such that \( m^*(i) \leq p(|i|, \max(i)) \), for every input \( i \). If \( \mathcal{P} \neq \mathcal{NP} \), then \( O \) does not belong to the class \( \mathcal{FPTAS} \).
Negative results for the class \( \text{FPTAS} \)

- The class of combinatorial problems in \( \text{PTAS} \) that admit a \( \text{FPTAS} \) is drastically reduced of those problems, whose value of the optimal measure is polynomially bounded with respect to the length of the instance.
- No \( \text{NP} \)-hard polynomially bounded optimization problem belongs to the class \( \text{FPTAS} \) unless \( \mathcal{P} = \mathcal{NP} \).
- No \( \text{NP} \)-hard problem that admits a polynomial \( p \) such that \( m^*(i) \leq p(|i|, \max(i)) \), for every input \( i \) belongs to the class \( \text{FPTAS} \) unless \( \mathcal{P} = \mathcal{NP} \).
Negative results for the class \( \text{FPTAS} \)

- The class of combinatorial problems in \( \text{PTAS} \) that admit a \( \text{FPTAS} \) is drastically reduced of those problems, whose value of the optimal measure is polynomially bounded with respect to the length of the instance.

- No \( \text{NP} \)-hard polynomially bounded optimization problem belongs to the class \( \text{FPTAS} \) unless \( \mathcal{P} = \mathcal{NP} \).

- No \( \text{NP} \)-hard problem that admits a polynomial \( p \) such that \( m^*(i) \leq p(|i|, \max(i)) \), for every input \( i \) belongs to the class \( \text{FPTAS} \) unless \( \mathcal{P} = \mathcal{NP} \).
Negative results for the class $\mathcal{FPTAS}$

- The class of combinatorial problems in $\mathcal{PTAS}$ that admit a $\mathcal{FPTAS}$ is drastically reduced of those problems, whose value of the optimal measure is polynomially bounded with respect to the length of the instance.

- No $\mathcal{NP}$-hard polynomially bounded optimization problem belongs to the class $\mathcal{FPTAS}$ unless $\mathcal{P} = \mathcal{NP}$.

- No $\mathcal{NP}$-hard problem that admits a polynomial $p$ such that $m^*(i) \leq p(|i|, \max(i))$, for every input $i$ belongs to the class $\mathcal{FPTAS}$ unless $\mathcal{P} = \mathcal{NP}$. 
Negative results for the class $\mathcal{FPTAS}$

- The class of combinatorial problems in $\mathcal{PTAS}$ that admit a $\mathcal{FPTAS}$ is drastically reduced of those problems, whose value of the optimal measure is polynomially bounded with respect to the length of the instance.

- No $\mathcal{NP}$-hard polynomially bounded optimization problem belongs to the class $\mathcal{FPTAS}$ unless $\mathcal{P} = \mathcal{NP}$.

- No $\mathcal{NP}$-hard problem that admits a polynomial $p$ such that $m^*(i) \leq p(|i|, \max(i))$, for every input $i$ belongs to the class $\mathcal{FPTAS}$ unless $\mathcal{P} = \mathcal{NP}$.
Outline

1. Approximation algorithms and errors
2. Classes
   - \( NPO \)
   - \( APX \)
   - \( PTAS \) and \( FPTAS \)
   - \( F - APX \)
   - Negative Results
3. Outlook
   - AP-Reductions
   - \( MaxSNP \)
Definition

Let $O_1$ and $O_2$ be two optimization problems in $\mathcal{NP}_O$. $O_1$ is said to be AP-reducible to $O_2$, in symbol $O_1 \leq_{AP} O_2$, if two functions $f$ and $g$ and a positive constant $\alpha \geq 1$ exist such that:

- For any instance $i \in I_{O_1}$ and for any rational $r > 1$, $f(i, r) \in I_{O_2}$.
- For any instance $i \in I_{O_1}$ and for any rational $r > 1$, if $SOL_{O_1}(i) \neq \emptyset$ then $SOL_{O_2}(f(i, r)) \neq \emptyset$.
- For any instance $i \in I_{O_1}$, for any rational $r > 1$, and for any $y \in SOL_{O_2}(f(i, r))$, $g(i, y, r) \in SOL_{O_1}(i)$.
Approximation Preserving Reductions

Definition

Let $O_1$ and $O_2$ be two optimization problems in $\mathcal{NP}O$. $O_1$ is said to be **AP-reducible** to $O_2$, in symbol $O_1 \leq_{AP} O_2$, if two functions $f$ and $g$ and a positive constant $\alpha \geq 1$ exist such that:

- For any instance $i \in I_{O_1}$ and for any rational $r > 1$, $f(i, r) \in I_{O_2}$.
- For any instance $i \in I_{O_1}$ and for any rational $r > 1$, if $SOL_{O_1}(i) \neq \emptyset$ then $SOL_{O_2}(f(i, r)) \neq \emptyset$.
- For any instance $i \in I_{O_1}$, for any rational $r > 1$, and for any $y \in SOL_{O_2}(f(i, r))$, $g(i, y, r) \in SOL_{O_1}(i)$. 

Stefan Kugele

Complexity Classes for Optimization Problems
Approximation Preserving Reductions

Definition

Let $O_1$ and $O_2$ be two optimization problems in $\mathcal{NP}$. $O_1$ is said to be \textbf{AP-reducible} to $O_2$, in symbol $O_1 \leq_{AP} O_2$, if two functions $f$ and $g$ and a positive constant $\alpha \geq 1$ exist such that:

- For any instance $i \in I_{O_1}$ and for any rational $r > 1$, $f(i, r) \in I_{O_2}$.
- For any instance $i \in I_{O_1}$ and for any rational $r > 1$, if $SOL_{O_1}(i) \neq \emptyset$ then $SOL_{O_2}(f(i, r)) \neq \emptyset$.
- For any instance $i \in I_{O_1}$, for any rational $r > 1$, and for any $y \in SOL_{O_2}(f(i, r))$, $g(i, y, r) \in SOL_{O_1}(i)$. 

Stefan Kugele 
Complexity Classes for Optimization Problems
Approximation Preserving Reductions

Definition

Let $O_1$ and $O_2$ be two optimization problems in $\mathcal{NP} \mathcal{O}$. $O_1$ is said to be **AP-reducible** to $O_2$, in symbol $O_1 \leq_{AP} O_2$, if two functions $f$ and $g$ and a positive constant $\alpha \geq 1$ exist such that:

- For any instance $i \in I_{O_1}$ and for any rational $r > 1$, $f(i, r) \in I_{O_2}$.
- For any instance $i \in I_{O_1}$ and for any rational $r > 1$, if $\text{SOL}_{O_1}(i) \neq \emptyset$ then $\text{SOL}_{O_2}(f(i, r)) \neq \emptyset$.
- For any instance $i \in I_{O_1}$, for any rational $r > 1$, and for any $y \in \text{SOL}_{O_2}(f(i, r))$, $g(i, y, r) \in \text{SOL}_{O_1}(i)$. 
**Definition (cont.)**

- $f$ and $g$ are computable by two algorithms $A_f$ and $A_g$, respectively, whose running time is polynomial for any fixed rational $r$.
- For any instance $i \in I_{O_1}$, for any rational $r > 1$, and for any $y \in SOL_{O_2}(f(i, r))$, 
  \[ R_{O_2}(f(i, r), y) \leq r \Rightarrow R_{O_1}(i, g(x, y, r)) \leq 1 + \alpha(r - 1). \]

  This is the **AP-condition**.
- The triple $(f, g, \alpha)$ is said to be an AP-reduction from $O_1$ to $O_2$. 

---

Stefan Kugele

Complexity Classes for Optimization Problems
Approximation Preserving Reductions (cont.)

Definition (cont.)

- $f$ and $g$ are computable by two algorithms $A_f$ and $A_g$, respectively, whose running time is polynomial for any fixed rational $r$.
- For any instance $i \in I_{O_1}$, for any rational $r > 1$, and for any $y \in SOL_{O_2}(f(i, r))$, $R_{O_2}(f(i, r), y) \leq r \Rightarrow R_{O_1}(i, g(x, y, r)) \leq 1 + \alpha(r - 1)$.

This is the **AP-condition**.

- The triple $(f, g, \alpha)$ is said to be an AP-reduction from $O_1$ to $O_2$. 
Approximation Preserving Reductions (cont.)

Definition (cont.)

- $f$ and $g$ are computable by two algorithms $A_f$ and $A_g$, respectively, whose running time is polynomial for any fixed rational $r$.
- For any instance $i \in I_{O_1}$, for any rational $r > 1$, and for any $y \in SOL_{O_2}(f(i, r))$,

$$R_{O_2}(f(i, r), y) \leq r \Rightarrow R_{O_1}(i, g(x, y, r)) \leq 1 + \alpha(r - 1).$$

This is the AP-condition.
- The triple $(f, g, \alpha)$ is said to be an AP-reduction from $O_1$ to $O_2$. 
AP-Reduction (cont.)

\[ \text{AP-Reduction} \]

\[ I_{O_1} \]
\[ f \]
\[ f(i, r) = i' \]
\[ \text{SOL}_{O_1}(i) \]
\[ g(i, y, r) \]
\[ g \]
\[ \text{SOL}_{O_2}(f(i)) \]
\[ A_{O_2} \]
**Lemma**

If $O_1 \leq_{AP} O_2$ and $O_2 \in APX$ (respectively, $O_2 \in PTAS$), then $O_1 \in APX$ (respectively, $O_1 \in PTAS$).

**Proof.**

→ Flipchart

**Example**

MaximumClique $\leq_{AP}$ MaximumIndependentSet
AP-Reduction (cont.)

Lemma

If $O_1 \leq_{AP} O_2$ and $O_2 \in APX$ (respectively, $O_2 \in PTAS$), then $O_1 \in APX$ (respectively, $O_1 \in PTAS$).

Proof.

→ Flipchart

Example

MaximumClique $\leq_{AP}$ MaximumIndependentSet
Lemma

If $O_1 \leq_{AP} O_2$ and $O_2 \in APX$ (respectively, $O_2 \in PTAS$), then $O_1 \in APX$ (respectively, $O_1 \in PTAS$).

Proof.

→ Flipchart

Example

MaximumClique $\leq_{AP}$ MaximumIndependentSet
Fagin’s Theorem (1974)

Theorem

A property is expressible in existential second-order logic ($\exists SO$) iff it is decidable in $\mathcal{NP}$. 

Fagin, Ron, IBM
The class $\mathsf{SNP}$ ($\mathsf{strictNP}$)

The class $\mathsf{SNP}$ consists of all properties expressible as

$$\exists S \forall x_1 \forall x_2 \ldots \forall x_k \varphi(S, G, x_1, \ldots, x_k)$$

- $\varphi$ is a quantifier-free First-Order expression involving the variables $x_i$ and the structures $G$ and $S$.
- $G$ is the input, $S$ is the demanded relation that satisfies $\varphi$.

Modifications

- $\varphi$ holds not for all $k$-tuples of nodes $(x_1, \ldots, x_k)$, instead we seek the relation $S$ such that $\varphi$ holds for as many $k$-tuples $(x_1, \ldots, x_k)$ as possible.
- $G$ now is a collection $G_1, \ldots, G_m$ of relations of arbitrary arity.
The class $\mathcal{SNP}$ (strict $\mathcal{NP}$)

The class $\mathcal{SNP}$ consists of all properties expressible as

$$\exists S \forall x_1 \forall x_2 \ldots \forall x_k \varphi(S, G, x_1, \ldots, x_k)$$

- $\varphi$ is a quantifier-free First-Order expression involving the variables $x_i$ and the structures $G$ and $S$.
- $G$ is the input, $S$ is the demanded relation that satisfies $\varphi$.

**Modifications**

- $\varphi$ holds not for all $k$-tuples of nodes $(x_1, \ldots, x_k)$, instead we seek the relation $S$ such that $\varphi$ holds for as many $k$-tuples $(x_1, \ldots, x_k)$ as possible.
- $G$ now is a collection $G_1, \ldots, G_m$ of relations of arbitrary arity.
The classes $\text{MaxSNP}_0$ and $\text{MaxSNP}$

**$\text{MaxSNP}_0$**

$$\max_S \left| \left\{ (x_1, \ldots, x_k) \in V^k : \varphi(G_1, \ldots, G_m, S, x_1, \ldots, x_k) \right\} \right|$$

**Definition**

$\text{MaxSNP}$ is the class of all optimization problems that are L-reducible to a problem in $\text{MaxSNP}_0$

- Introduced in 1989 by Papadimitriou and Yannakakis
- They showed, that all problems in $\text{MaxSNP}$ are in $\text{APX}$
- You can get an approximation algorithm canonical out of the problem definition using $\varphi$
The classes $\text{MaxSNP}_0$ and $\text{MaxSNP}$

$\text{MaxSNP}_0$

$$\max_S \left| \left\{ (x_1, \ldots, x_k) \in V^k : \varphi(G_1, \ldots, G_m, S, x_1, \ldots, x_k) \right\} \right|$$

Definition

$\text{MaxSNP}$ is the class of all optimization problems that are L-reducible to a problem in $\text{MaxSNP}_0$

- Introduced in 1989 by Papadimitriou and Yannakakis
- They showed, that all problems in $\text{MaxSNP}$ are in $\text{APX}$
- You can get an approximation algorithm canonical out of the problem definition using $\varphi$