GEMS OF TCS

P VS NP

Sasha Golovnev
October 20, 2021
Search Problems
A search problem is defined by an algorithm $\mathcal{C}$ that takes an instance $I$ and a candidate solution $S$, and runs in time polynomial in the length of $I$. We say that $S$ is a solution to $I$ iff $\mathcal{C}(S, I) = \text{true}$. 
Example

For SAT, $I$ is a Boolean formula, $S$ is an assignment of Boolean constants to its variables. The corresponding algorithm $C$ checks whether $S$ satisfies all clauses of $I$. 
### Definition

A **search problem** is defined by an algorithm $\mathcal{C}$ that takes an instance $I$ and a candidate solution $S$, and runs in time polynomial in the length of $I$. We say that $S$ is a solution to $I$ iff $\mathcal{C}(S, I) = \text{true}$.

**Class NP**
**Class NP**

**Definition**

A *search problem* is defined by an algorithm $C$ that takes an instance $I$ and a candidate solution $S$, and runs in time polynomial in the length of $I$. We say that $S$ is a solution to $I$ iff $C(S, I) = \text{true}$.

**Definition**

NP is the class of all search problems.
• **NP** stands for “non-deterministic polynomial time”: one can guess a solution, and then verify its correctness in polynomial time.
• **NP** stands for “non-deterministic polynomial time”: one can guess a solution, and then verify its correctness in polynomial time.

• In other words, the class **NP** contains all problems whose solutions can be efficiently verified.
**Definition**

\( \mathsf{P} \) is the class of all search problems that can be solved in polynomial time.
Given a complete weighted graph, find a path of minimum total weight (length) visiting each node exactly once.
Given a complete weighted graph and a budget $b$, find a path of total weight (length) $\leq b$ visiting each node exactly once.
Minimum Spanning Tree

Given a complete weighted graph and a budget $b$, connect all vertices by $n - 1$ edges of minimum total weight (length)

![Diagram of a minimum spanning tree with edge weights and total length 6]
TSP AND MST

MST

Given $n$ cities, connect them by $(n - 1)$ roads of minimal total length

Can be solved efficiently

TSP

Given $n$ cities, connect them in a path of minimal total length

No polynomial algorithm known!
TSP AND MST

MST

Given $n$ cities, connect them by $(n - 1)$ roads of minimal total length

Can be solved efficiently
## TSP AND MST

<table>
<thead>
<tr>
<th></th>
<th>MST</th>
<th>TSP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MST</strong></td>
<td>Given $n$ cities, connect them by $(n - 1)$ roads of minimal total length</td>
<td><strong>TSP</strong></td>
</tr>
<tr>
<td><strong>Can be solved efficiently</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## TSP AND MST

<table>
<thead>
<tr>
<th>MST</th>
<th>TSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given $n$ cities, connect them by $(n - 1)$ roads of minimal total length</td>
<td>Given $n$ cities, connect them in a path of minimal total length</td>
</tr>
<tr>
<td>Can be solved efficiently</td>
<td>No polynomial algorithm known!</td>
</tr>
</tbody>
</table>
### Longest Path

**Input:** A weighted graph, two vertices \( s, t \), and a budget \( b \).

**Output:** A simple path (containing no repeated vertices) of total length at least \( b \).
Example
Example
Shortest path

Find a simple path from $s$ to $t$ of total length at most $b$
Shortest path

Find a simple path from \( s \) to \( t \) of total length \( \text{at most} \ b \)

Can be solved efficiently

Longest path

Find a simple path from \( s \) to \( t \) of total length \( \text{at least} \ b \)

No polynomial algorithm known!
<table>
<thead>
<tr>
<th>Shortest path</th>
<th>Longest path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find a simple path from s to t of total length <strong>at most</strong> b</td>
<td>Find a simple path from s to t of total length <strong>at least</strong> b</td>
</tr>
<tr>
<td>Can be solved efficiently</td>
<td></td>
</tr>
<tr>
<td>Shortest path</td>
<td>Longest path</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>Find a simple path from s to t of total length at most $b$</td>
<td>Find a simple path from s to t of total length at least $b$</td>
</tr>
<tr>
<td>Can be solved efficiently</td>
<td>No polynomial algorithm known!</td>
</tr>
</tbody>
</table>
Integer Linear Programming Problem

Input: A set of linear inequalities $Ax \leq b$.
Output: Integer solution.
Example

\[ x_1 \geq 0.5 \]
\[ -x_1 + 8x_2 \geq 0 \]
\[ -x_1 - 8x_2 \geq -8 \]
Example

\[ x_1 \geq 0.5 \]
\[ -x_1 + 8x_2 \geq 0 \]
\[ -x_1 - 8x_2 \geq -8 \]
Example

\[ x_1 \geq 0.5 \]
\[ -x_1 + 8x_2 \geq 0 \]
\[ -x_1 - 8x_2 \geq -8 \]
Example

\[ x_1 \geq 0.5 \]
\[ -x_1 + 8x_2 \geq 0 \]
\[ -x_1 - 8x_2 \geq -8 \]
Example

\[ x_1 \geq 0.5 \]

\[ -x_1 + 8x_2 \geq 0 \]

\[ -x_1 - 8x_2 \geq -8 \]
# Integer Linear Programming

<table>
<thead>
<tr>
<th>LP</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Find a <strong>real</strong> solution of a system of linear inequalities</td>
<td></td>
</tr>
</tbody>
</table>
# Integer Linear Programming

<table>
<thead>
<tr>
<th>LP</th>
<th>Find a <strong>real</strong> solution of a system of linear inequalities</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Can be solved efficiently</td>
</tr>
</tbody>
</table>
# Integer Linear Programming

<table>
<thead>
<tr>
<th><strong>LP</strong></th>
<th><strong>ILP</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Find a <em>real</em> solution of a system of linear inequalities</td>
<td>Find an <em>integer</em> solution of a system of linear inequalities</td>
</tr>
<tr>
<td>Can be solved efficiently</td>
<td></td>
</tr>
</tbody>
</table>
## Integer Linear Programming

<table>
<thead>
<tr>
<th>LP</th>
<th>ILP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Find a</strong> real solution of a system of linear inequalities</td>
<td><strong>Find an</strong> integer solution of a system of linear inequalities</td>
</tr>
<tr>
<td>Can be solved efficiently</td>
<td>No polynomial algorithm known!</td>
</tr>
</tbody>
</table>
Independent set

**Input:** A graph and a budget $b$.

**Output:** A subset of vertices of size at least $b$ such that no two of them are adjacent.
Example
A maximum independent set in a tree can be found by a simple greedy algorithm: it is safe to take into a solution all the leaves.
### Independent set in a tree

Find an independent set of size at least $b$ in a given tree
Independent set in a tree

Find an independent set of size at least $b$ in a given tree

Can be solved efficiently
<table>
<thead>
<tr>
<th><strong>Independent set in a tree</strong></th>
<th><strong>Independent set in a graph</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Find an independent set of size at least $b$ in a given <em>tree</em></td>
<td>Find an independent set of size at least $b$ in a given <em>graph</em></td>
</tr>
<tr>
<td>Can be solved efficiently</td>
<td></td>
</tr>
<tr>
<td>Independent set in a tree</td>
<td>Independent set in a graph</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Find an independent set of size at least $b$ in a given tree</td>
<td>Find an independent set of size at least $b$ in a given graph</td>
</tr>
<tr>
<td>Can be solved efficiently</td>
<td>No polynomial algorithm known!</td>
</tr>
</tbody>
</table>
It turns out that all these hard problems are in a sense a single hard problem: a polynomial time algorithm for any of these problems can be used to solve all of them in polynomial time!
<table>
<thead>
<tr>
<th>Class P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problems whose solution can be found efficiently</td>
</tr>
<tr>
<td>Class P</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>Problems whose solution can be <strong>found</strong> efficiently</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>• MST</td>
</tr>
<tr>
<td>• Shortest path</td>
</tr>
<tr>
<td>• LP</td>
</tr>
<tr>
<td>• IS on trees</td>
</tr>
<tr>
<td>Class P</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>Problems whose solution can be found efficiently</td>
</tr>
</tbody>
</table>

- MST
- Shortest path
- LP
- IS on trees
<table>
<thead>
<tr>
<th>Class P</th>
<th>Class NP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problems whose solution can be <strong>found</strong> efficiently</td>
<td>Problems whose solution can be <strong>verified</strong> efficiently</td>
</tr>
<tr>
<td>• MST</td>
<td>• TSP</td>
</tr>
<tr>
<td>• Shortest path</td>
<td>• Longest path</td>
</tr>
<tr>
<td>• LP</td>
<td>• ILP</td>
</tr>
<tr>
<td>• IS on trees</td>
<td>• IS on graphs</td>
</tr>
</tbody>
</table>
The main open problem in Computer Science

Is \( P \) equal to \( NP \)?
The main open problem in Computer Science

Is \( \mathbf{P} \) equal to \( \mathbf{NP} \)?

Millenium Prize Problem

Clay Mathematics Institute: $1M prize for solving the problem
• If $P=NP$, then all search problems can be solved in polynomial time.
• If P=NP, then all search problems can be solved in polynomial time.

• If P≠NP, then there exist search problems that cannot be solved in polynomial time.
Reductions
INFORMALLY

We say that a search problem $A$ is reduced to a search problem $B$ and write $A \rightarrow B$, if a polynomial time algorithm for $B$ can be used (as a black box) to solve $A$ in polynomial time.
Reduction: $A \rightarrow B$

instance $I$ of $A$
REDUCTION: $A \rightarrow B$

instance $I$ of $A$

Algorithm for $A$

Algorithm for $B$
Reduction: $A \rightarrow B$

Instance $I$ of $A$

Algorithm for $A$

$f$

Algorithm for $B$
REDUCTION: $A \rightarrow B$

Algorithm for $A$

$\rightarrow$

instance $I$ of $A$

$f$

instance $f(I)$ of $B$

Algorithm for $B$
Reduction: \( A \rightarrow B \)

Instance \( I \) of \( A \)

Algorithm for \( A \)

\( f \)

Instance \( f(I) \) of \( B \)

Algorithm for \( B \)

No solution to \( f(I) \)
Reduction: \( A \rightarrow B \)

- Instance \( I \) of \( A \)
- Algorithm for \( A \) \( \rightarrow \) \( f \)
- Instance \( f(I) \) of \( B \)
- Algorithm for \( B \)
- No solution to \( f(I) \)
- Algorithm for \( B \)
- No solution to \( I \)
**REDUCTION: A → B**

- **Algorithm for A**
  - instance $I$ of $A$
  - $f(I)$
  - instance $f(I)$ of $B$
  - **Algorithm for B**
    - no solution to $f(I)$
    - solution $S$ to $f(I)$
  - no solution to $I$
**Reduction: $A \rightarrow B$**

- Instance $I$ of $A$
- Algorithm for $A$
  - $f$
  - Instance $f(I)$ of $B$
  - Algorithm for $B$
    - No solution to $f(I)$
    - Solution $S$ to $f(I)$
    - $h$
- No solution to $I$
REDUCTION: $A \rightarrow B$

- Instance $I$ of $A$
- Algorithm for $A$
- $f(I)$
  - Instance $f(I)$ of $B$
  - Algorithm for $B$
  - Solution $S$ to $f(I)$
  - $h(S)$
- No solution to $f(I)$
- No solution to $I$
- Solution $h(S)$ to $I$
**Definition**

We say that a search problem $A$ is reduced to a search problem $B$ and write $A \rightarrow B$, if there exists a polynomial time algorithm $f$ that converts any instance $I$ of $A$ into an instance $f(I)$ of $B$, together with a polynomial time algorithm $h$ that converts any solution $S$ to $f(I)$ back to a solution $h(S)$ to $A$. If there is no solution to $f(I)$, then there is no solution to $I$. 

---

<table>
<thead>
<tr>
<th>Formally</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Definition</strong></td>
</tr>
<tr>
<td>We say that a search problem $A$ is reduced to a search problem $B$ and write $A \rightarrow B$, if there exists a polynomial time algorithm $f$ that converts any instance $I$ of $A$ into an instance $f(I)$ of $B$, together with a polynomial time algorithm $h$ that converts any solution $S$ to $f(I)$ back to a solution $h(S)$ to $A$. If there is no solution to $f(I)$, then there is no solution to $I$.</td>
</tr>
</tbody>
</table>
GRAPH OF SEARCH PROBLEMS

NP
<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>A search problem is called <strong>NP-complete</strong> if all other search problems reduce to it.</td>
</tr>
</tbody>
</table>
A search problem is called **NP-complete** if all other search problems reduce to it.
Do they exist?

It’s not at all immediate that $\textbf{NP}$-complete problems even exist. We’ll see later that all hard problems that we’ve seen in the previous part are in fact $\textbf{NP}$-complete!
Two ways of using $A \rightarrow B$:

- if $B$ is easy (can be solved in polynomial time), then so is $A$

- if $A$ is hard (cannot be solved in polynomial time), then so is $B$
<table>
<thead>
<tr>
<th>Lemma</th>
</tr>
</thead>
<tbody>
<tr>
<td>If $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$.</td>
</tr>
</tbody>
</table>
Pictorially
PICTORIALLY

NP
Pictorially

NP
### Corollary

If $A \rightarrow B$ and $A$ is **NP**-complete, then so is $B$. 
Corollary

If $A \rightarrow B$ and $A$ is NP-complete, then so is $B$. 
**Corollary**

If $A \rightarrow B$ and $A$ is NP-complete, then so is $B$. 
Corollary

If $A \rightarrow B$ and $A$ is \textbf{NP}-complete, then so is $B$. 
Corollary

If $A \rightarrow B$ and $A$ is $\textbf{NP}$-complete, then so is $B$. 
Corollary

If $A \rightarrow B$ and $A$ is $\text{NP}$-complete, then so is $B$. 

**Diagram:**

- Node $A$ points to node $B$.
- Node $B$ is labeled as $\text{NP}$.

**Text:**

- Showing NP-completeness
- Corollary
  - If $A \rightarrow B$ and $A$ is $\text{NP}$-complete, then so is $B$. 

**Diagram:** (A simple diagram with a directed edge from $A$ to $B$, with $B$ labeled as $\text{NP}$.)
NP-Completeness of SAT
Goal

Show that every search problem reduces to SAT.
Goal

Show that every search problem reduces to SAT.

Instead, we show that any problem reduces to Circuit SAT problem, which, in turn, reduces to SAT.
Circuit

\[
\begin{align*}
x & \quad \lor \quad \lnot \quad y \\
& \quad \land \quad \lor \\
& \quad \lor \\
& \quad \lor \\
\text{output} &
\end{align*}
\]
A **circuit** is a directed acyclic graph of in-degree at most 2. Nodes of in-degree 0 are called **inputs** and are marked by Boolean variables and constants. Nodes of in-degree 1 and 2 are called **gates**: gates of in-degree 1 are labeled with NOT, gates of in-degree 2 are labeled with AND or OR. One of the sinks is marked as **output**.
Circuit-SAT

**Input:** A circuit.

**Output:** An assignment of Boolean values to the input variables of the circuit that makes the output true.
SAT is a special case of Circuit-SAT as a SAT formula can be represented as a circuit:

**Example:** \((x \lor y \lor z)(y \lor \overline{x})\)
To reduce Circuit-SAT to SAT, we need to design a polynomial time algorithm that for a given circuit outputs a SAT formula which is satisfiable, if and only if the circuit is satisfiable.
IDEA

• Introduce a Boolean variable for each gate

• For each gate, write down a few clauses that describe the relationship between this gate and its direct predecessors
$\neg\ g\ h\ (h\lor g)(\bar{h}\lor \bar{g})$
\[(h_1 \lor \bar{g})(h_2 \lor \bar{g})(\bar{h}_1 \lor \bar{h}_2 \lor g)\]
**OR GATES**

![OR Gate Diagram]

\[(\overline{h}_1 \lor g)(\overline{h}_2 \lor g)(h_1 \lor h_2 \lor \overline{g})\]
Output Gate

$g \bigcirc$ output $(g)$
• The resulting SAT formula is consistent with the initial circuit: in any satisfying assignment of the formula, the value of $g$ is equal to the value of the gate labeled with $g$ in the circuit.
• The resulting SAT formula is consistent with the initial circuit: in any satisfying assignment of the formula, the value of $g$ is equal to the value of the gate labeled with $g$ in the circuit

• Therefore, the SAT formula and the circuit are equisatisfiable
• The resulting SAT formula is consistent with the initial circuit: in any satisfying assignment of the formula, the value of $g$ is equal to the value of the gate labeled with $g$ in the circuit.
• Therefore, the SAT formula and the circuit are equisatisfiable.
• The reduction takes polynomial time.
Goal

Reduce every search problem to Circuit-SAT.
<table>
<thead>
<tr>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reduce every search problem to Circuit-SAT.</td>
</tr>
</tbody>
</table>

- Let $A$ be a search problem
Goal

Reduce every search problem to Circuit-SAT.

• Let $A$ be a search problem
• We know that there exists an algorithm $C$ that takes an instance $I$ of $A$ and a candidate solution $S$ and checks whether $S$ is a solution for $I$ in time polynomial in $|I|$
Goal

Reduce every search problem to Circuit-SAT.

- Let $A$ be a search problem
- We know that there exists an algorithm $C$ that takes an instance $I$ of $A$ and a candidate solution $S$ and checks whether $S$ is a solution for $I$ in time polynomial in $|I|$
- In particular, $|S|$ is polynomial in $|I|$
Note that a computer is in fact a circuit implemented on a chip.
• Note that a computer is in fact a circuit implemented on a chip
• Each step of the algorithm $C(I, S)$ is performed by this computer’s circuit
• Note that a computer is in fact a circuit implemented on a chip
• Each step of the algorithm $C(I, S)$ is performed by this computer’s circuit
• This gives a circuit of size polynomial in $|I|$ that has input bits for $I$ and $S$ and outputs whether $S$ is a solution for $I$ (a separate circuit for each input length)
To solve an instance $I$ of the problem $A$:

- take a circuit corresponding to $C(I, \cdot)$
To solve an instance $I$ of the problem $A$:

- take a circuit corresponding to $C(I, \cdot)$
- the inputs to this circuit encode candidate solutions
To solve an instance $I$ of the problem $A$:

- take a circuit corresponding to $C(I, \cdot)$
- the inputs to this circuit encode candidate solutions
- use a Circuit-SAT algorithm for this circuit to find a solution (if exists)
SUMMARY

Circuit-SAT → SAT