GEMS OF TCS

EXPONENTIAL-TIME ALGORITHMS

Sasha Golovnev

February 11, 2021
Exact Algorithms

- We need to solve problem exactly
EXACT ALGORITHMS

- We need to solve problem exactly
- Problem takes exponential time solve exactly
EXACT ALGORITHMS

• We need to solve problem exactly

• Problem takes exponential time solve exactly

• Intelligent exhaustive search: finding optimal solution without going through all candidate solutions
### Running Time

<table>
<thead>
<tr>
<th>Running Time</th>
<th>Streaming alg.</th>
<th>Poly-time</th>
<th>Exp-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>running time:</td>
<td>$n$</td>
<td>$n^2$</td>
<td>$n^3$</td>
</tr>
<tr>
<td>less than $10^9$:</td>
<td>$10^9$</td>
<td>$10^{4.5}$</td>
<td>$10^3$</td>
</tr>
</tbody>
</table>

$30k$
# Running Time

<table>
<thead>
<tr>
<th>running time:</th>
<th>$n$</th>
<th>$n^2$</th>
<th>$n^3$</th>
<th>$n!$</th>
</tr>
</thead>
<tbody>
<tr>
<td>less than $10^9$:</td>
<td>$10^9$</td>
<td>$10^{4.5}$</td>
<td>$10^3$</td>
<td>12</td>
</tr>
</tbody>
</table>

$n! \approx 2^{n \log_2 n}$

$\text{Exp-time alg's}$

<table>
<thead>
<tr>
<th>running time:</th>
<th>$n!$</th>
<th>$4^n$</th>
<th>$2^n$</th>
<th>$1.308^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>less than $10^9$:</td>
<td>12</td>
<td>14</td>
<td>29</td>
<td>77</td>
</tr>
</tbody>
</table>
Traveling Salesman Problem (TSP)
**TRAVELING SALESMAN PROBLEM**

Given a complete weighted graph, find a cycle (or a path) of minimum total weight (length) visiting each node exactly once.

[Graph showing a weighted graph with labeled edges and a cycle highlighted.]

\[\text{length: 9}\]
TRAVELING SALESMAN PROBLEM

Given a complete weighted graph, find a cycle (or a path) of minimum total weight (length) visiting each node exactly once.
ALGORITHMS

• Classical optimization problem with countless number of real life applications (see Lecture 1)
ALGORITHMS

• Classical optimization problem with countless number of real life applications (see Lecture 1)
• No polynomial time algorithms known
ALGORITHMS

• Classical optimization problem with countless number of real life applications (see Lecture 1)
• No polynomial time algorithms known
• We’ll see exact exponential-time algorithms
A naive algorithm just checks all possible $\sim n!$ cycles.
**Brute Force Solution**

A naive algorithm just checks all possible $\sim n!$ cycles.

$$n! \approx 2^{n \log_2 n} = e^{n \ln n}$$

<table>
<thead>
<tr>
<th>We’ll see</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n! \approx n^n$</td>
</tr>
</tbody>
</table>

- Use dynamic programming to solve TSP in $O(n^2 \cdot 2^n) \approx 2^n$
Brute Force Solution

A naive algorithm just checks all possible $\sim n!$ cycles.

We’ll see

- Use dynamic programming to solve TSP in $O(n^2 \cdot 2^n)$
- The running time is exponential, but is much better than $n!$
Dynamic Programming

- Dynamic programming is one of the most powerful algorithmic techniques

1962 still remaining best known for TSP

was invented for TSP
Dynamic Programming

- Dynamic programming is one of the most powerful algorithmic techniques
- Rough idea: express a solution for a problem through solutions for smaller subproblems
Dynamic Programming

- Dynamic programming is one of the most powerful algorithmic techniques
- Rough idea: express a solution for a problem through solutions for smaller subproblems
- Solve subproblems one by one. Store solutions to subproblems in a table to avoid recomputing the same thing again
**Subproblems**

- For a subset of vertices \( \{S\} \subseteq \{1, \ldots, n\} \) containing the vertex 1 and a vertex \( i \in S \), let \( C(S, i) \) be the length of the shortest path that starts at 1, ends at \( i \) and visits all vertices from \( S \) exactly once.
For any set $S \subseteq \{1, \ldots, n\}$
For any vertex $i \in \{1, \ldots, n\}$
\[
C(S, i) = \text{length of shortest path that:}
\]
1. Starts at 1
2. Ends at $i$
3. Visits every vertex from $S$ exactly once
$C(\{1\}, 1) = 0$

$C(\{1, 2\}, 1) = +\infty$

$C(\{1, 2, 3\}, 1) = +\infty$

$C(\{1, 2, 3\}, 3) = 1 + 7 = 8$

$C(\{1, 2, 3\}, 2) = 10 + 7 = 17$

$C(\{1, 2, 3, 4\}) = \min (\quad C(\{1, 2, 3\}, 3) + 3, \quad C(\{1, 2, 3\}, 2) + 15 \quad)$.
SUBPROBLEMS

• For a subset of vertices $\mathcal{S} \subseteq \{1, \ldots, n\}$ containing the vertex 1 and a vertex $i \in \mathcal{S}$, let $C(\mathcal{S}, i)$ be the length of the shortest path that starts at 1, ends at $i$ and visits all vertices from $\mathcal{S}$ exactly once.

• $C(\{1\}, 1) = 0$ and $C(\mathcal{S}, 1) = +\infty$ when $|\mathcal{S}| > 1$
Recurrence Relation

\[ C(S, i) = \min_{j \in S} C(S \setminus \{i\}, j) + d_{ji} \]

- Consider the second-to-last vertex \( j \) on the required shortest path from 1 to \( i \) visiting all vertices from \( S \).
Recurrence Relation

• Consider the second-to-last vertex $j$ on the required shortest path from 1 to $i$ visiting all vertices from $S$
• The subpath from 1 to $j$ is the shortest one visiting all vertices from $S - \{i\}$ exactly once
Recurrence Relation

• Consider the second-to-last vertex $j$ on the required shortest path from 1 to $i$ visiting all vertices from $S$
• The subpath from 1 to $j$ is the shortest one visiting all vertices from $S - \{i\}$ exactly once
• Hence
\[
C(S, i) = \min_j \{C(S - \{i\}, j) + d_{ji}\},
\]
where the minimum is over all $j \in S$ such that $j \neq i$
ORDER OF SUBPROBLEMS

\( C(S, i) \)

- Need to process all subsets \( S \subseteq \{1, \ldots, n\} \) in an order that guarantees that when computing the value of \( C(S, i) \), the values of \( C(S - \{i\}, j) \) have already been computed.
ORDER OF SUBPROBLEMS

- Need to process all subsets $S \subseteq \{1, \ldots, n\}$ in an order that guarantees that when computing the value of $C(S, i)$, the values of $C(S - \{i\}, j)$ have already been computed.
- For example, we can process subsets in order of increasing size.
ALGORITHM

\$S, i$

\[ C(*, *) \leftarrow +\infty \]

\[ C(\{1\}, 1) \leftarrow 0 \]
ALGORITHM

\[
C(\ast, \ast) \leftarrow +\infty \\
C(\{1\}, 1) \leftarrow 0 \\
\text{for } s \text{ from } 2 \text{ to } n: \\
\quad \text{for all } 1 \in S \subseteq \{1, \ldots, n\} \text{ of size } s:
\]

size of \(S\): \(s = 151\)
**ALGORITHM**

\[ C(\ast, \ast) \leftarrow +\infty \]

\[ C(\{1\}, 1) \leftarrow 0 \]

for s from 2 to n:

for all \( 1 \in S \subseteq \{1, \ldots, n\} \) of size s:

for all \( i \in S, i \neq 1 \):

for all \( j \in S, j \neq i \)

\[
C(S, \ast) - \text{always set } +1 \\
C(S, i) \leftarrow \min\{C(S, i), C(S - \{i\}, j) + d_{ji}\}
\]

the last vertex of path second-to-last

\[ C(S - \{i,j\}, j) \]

always set +1
length of cycle = C(\ell_1, \ldots, \ell_3, i) + d_{i,1}

length of shortest cycle =

= \min C(\ell_1, \ldots, \ell_3, i) + d_{i,1}
ALGORITHM

\[ C(*, *) \leftarrow +\infty \]
\[ C(\{1\}, 1) \leftarrow 0 \]

for s from 2 to n: 
    for all \(1 \in S \subseteq \{1, \ldots, n\}\) of size s: 
        \(2^n\)
        for all \(i \in S, i \neq 1\): 
            \(2^n\)
            for all \(j \in S, j \neq i\): 
                \(2^n\)
                \[ C(S, i) \leftarrow \min\{C(S, i), C(S - \{i\}, j) + d_{ji}\} \]

return \(\min_i\{C(\{1, \ldots, n\}, i) + d_{i, 1}\}\)

\(\approx 2^n\)

Run-time \(\leq 2^n \cdot n^3\)
Satisfiability Problem (SAT)
SAT

\[ n \text{ vars} \]
\[ x_i \in \{0, 1\} \]
\[ x_1 = x_2 = x_3 = 1 \]
\[ (x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2) \land (\neg x_1 \lor x_3) \land (x_2 \lor \neg x_3) \]
\[ 1 \quad 1 \quad 1 \quad 1 \]
\[ \text{SAT} \]

\[ (x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2) \land (\neg x_1 \lor x_3) \land (x_2 \lor \neg x_3) \]

\[ (x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2) \land (\neg x_1 \lor x_3) \land (x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \]

\[ \text{UNSAT} \]
$k$-SAT

\[
\phi(x_1, \ldots, x_n) = \left( x_1 \lor \neg x_2 \lor \ldots \lor x_k \right) \land \\
\ldots \land \\
\left( x_2 \lor \neg x_3 \lor \ldots \lor x_8 \right)
\]
$k$-SAT

$$
\phi(x_1, \ldots, x_n) = (x_1 \lor \neg x_2 \lor \ldots \lor x_k) \land \\
\ldots \land \\
(x_2 \lor \neg x_3 \lor \ldots \lor x_8)
$$

$\phi$ is **satisfiable** if

$$
\exists x \in \{0, 1\}^n : \phi(x) = 1.
$$

Otherwise, $\phi$ is **unsatisfiable**
$k$-SAT

\[ \phi(x_1, \ldots, x_n) = (x_1 \lor \neg x_2 \lor \ldots \lor x_k) \land \\
\ldots \\
(x_2 \lor \neg x_3 \lor \ldots \lor x_8) \]

\( \phi \) is satisfiable if

\[ \exists x \in \{0, 1\}^n : \phi(x) = 1. \]

Otherwise, \( \phi \) is unsatisfiable

\( n \) Boolean vars, \( m \) clauses
\( k\)-SAT

\[
\phi(x_1, \ldots, x_n) = (x_1 \lor \neg x_2 \lor \ldots \lor x_k) \land \\
\ldots \land \\
(x_2 \lor \neg x_3 \lor \ldots \lor x_8)
\]

\( \phi \) is satisfiable if

\[
\exists x \in \{0, 1\}^n : \phi(x) = 1.
\]

Otherwise, \( \phi \) is unsatisfiable

\( n \) Boolean vars, \( m \) clauses

\( k\)-SAT is SAT where clause length \( \leq k \)
\( k \)-SAT. EXAMPLES

\[ 3 \text{-SAT} \]

\[ (x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2) \land (\neg x_1 \lor x_3) \land (x_2 \lor \neg x_3) \]
$k$-SAT. EXAMPLES

$$(x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2) \land (\neg x_1 \lor x_3) \land (x_2 \lor \neg x_3)$$

$1$-SAT

$$(x_1) \land (\neg x_2) \land (x_3) \land (\neg x_1)$$
COMPLEXITY OF SAT
Complexity of SAT

SAT

$\kappa$-SAT
5-SAT
4-SAT
3-SAT

NP

2-SAT
1-SAT

P
But how hard is SAT?
SAT IN $2^n$

- $O^*(\cdot)$ suppresses polynomial factors in the input length:

$$2^n n^{10} m^2 = O^*(2^n)$$
SAT in $2^n$

- $O^*(\cdot)$ suppresses polynomial factors in the input length:

$$2^n n^{10} m^2 = O^*(2^n)$$

- SAT can be solved in time $O^*(2^n)$

$x, \ldots, x_n \in \{0, 1\}^n$ — $2^n$ such assignments

For each assignment, in linear time check assignment satisfies formula
SAT in $2^n$

- $O^*(\cdot)$ suppresses polynomial factors in the input length:

\[ 2^n n^{10} m^2 = O^*(2^n) \]

- SAT can be solved in time $O^*(2^n)$

- We don’t know how to solve SAT exponentially faster: in time $O^*(1.999^n)$

*Conjecture:* Every alg for SAT takes time $\geq 2^n$
3-SAT

\[ (x_1 \lor x_2 \lor x_9) \land \ldots \land (x_2 \lor \neg x_3 \lor x_8) \]
3-SAT

\[ (x_1 \lor x_2 \lor x_9) \land \ldots \land (x_2 \lor \neg x_3 \lor x_8) \]

Instead of checking \(0,1,3^n\) can check only those that don’t have \(x_1 = x_2 = x_3 = 0\)

\[2^n \cdot \frac{3}{8}\] assignments

Can be extended run-time \((7)^{n/3} \approx 1.92^n\)

Case 1: \(x_1 = 1\)
Case 2: \(x_1 = 0\) \(x_2 = 1\)
Case 3: \(x_1 = 0\) \(x_2 = 0\) \(x_3 = 1\)
3-SAT

- \((x_1 \lor x_2 \lor x_3) \land \ldots \land (x_2 \lor \neg x_3 \lor x_8)\)

- Consider three sub-problems:
  
  **Case I**:  
  \[x_1 = 1\]  
  Replace \(x_i \rightarrow 1; \neg x_i \rightarrow 0\) \(3\text{-SAT}(n-1)\)

  **Case II**:  
  \[x_1 = 0, x_2 = 1\]  
  Replace \(x_i \rightarrow 0; \neg x_i \rightarrow 1\); \(x_2 \rightarrow 1; \neg x_2 \rightarrow 0\) \(3\text{-SAT}(n-2)\)

  **Case III**:  
  \(x_1 = 0, x_2 = 0, x_9 = 1\)  
  \(3\text{-SAT}(n-3)\)
3-SAT

- \((x_1 \lor x_2 \lor x_9) \land \ldots \land (x_2 \lor \neg x_3 \lor x_8)\)

- Consider three sub-problems:
  - \(x_1 = 1\)
  - \(x_1 = 0, x_2 = 1\)
  - \(x_1 = 0, x_2 = 0, x_9 = 1\)
- The original formula is SAT iff at least one of these formulas is SAT
$3$-SAT *(Formula)*

Pick a clause $(x \lor y \lor z)$

$\rightarrow$  $3$-SAT *(formula, x = 1)*

$\rightarrow$  $3$-SAT *(formula, x = 0, y = 1)*

$\rightarrow$  $3$-SAT *(formula, x = 0, y = 0, z = 1)*

If one of these is TRUE,
Then RETURN TRUE

Else
Then RETURN FALSE

$T(n)$ - Run-time on flags with $n$ variables

$T(n) \leq T(n-1) + T(n-2) + T(n-3)$
3-SAT. Analysis

- \( T(n) \leq T(n - 1) + T(n - 2) + T(n - 3) \)

Claim \( T(n) \leq 1.85^n \)

Prove by induction on \( n \)

- \( T(n-1) \leq 1.85^{n-1} \)
- \( T(n-2) \leq 1.85^{n-2} \)
- \( T(n-3) \leq 1.85^{n-3} \)
3-SAT. Analysis

- \( T(n) \leq T(n - 1) + T(n - 2) + T(n - 3) \)
- \( T(n) \leq 1.85^n \)
3-SAT. Analysis

- $T(n) \leq T(n - 1) + T(n - 2) + T(n - 3)$
- $T(n) \leq 1.85^n$:

\[
\begin{align*}
T(n) & \leq T(n - 1) + T(n - 2) + T(n - 3) \\
& \leq 1.85^{n-1} + 1.85^{n-2} + 1.85^{n-3} \\
& = 1.85^n \left( \frac{1}{1.85} + \frac{1}{1.85^2} + \frac{1}{1.85^3} \right) \\
& < 1.85^n (0.991) \\
& < 1.85^n
\end{align*}
\]

A constant, $k$-SAT in $(2 - \varepsilon_k)^n$ SAT in time $2^n$
3-SAT. Analysis

- $T(n) \leq T(n - 1) + T(n - 2) + T(n - 3)$
- $T(n) \leq 1.85^n$:

$$T(n) \leq T(n - 1) + T(n - 2) + T(n - 3)$$
$$\leq 1.85^{n-1} + 1.85^{n-2} + 1.85^{n-3}$$
$$= 1.85^n \left( \frac{1}{1.85} + \frac{1}{1.85^2} + \frac{1}{1.85^3} \right)$$
$$< 1.85^n \times (0.991)$$
$$< 1.85^n$$

- There are even faster algorithms: $1.308^n$
  \[\text{[HKZZ19]}\]
How hard can SAT be?
Algorithmic Complexity of SAT

2-SAT $O(m)$

1-SAT $O(m)$
Algorithmic Complexity of SAT

- 3-SAT: $1.308^n$
- 2-SAT: $O(m)$
- 1-SAT: $O(m)$
Algorithmic Complexity of SAT

\[ k\text{-SAT } 2^n(1-O(1/k)) \]

\[ : \]

\[ 3\text{-SAT } 1.308^n \]

\[ 2\text{-SAT } O(m) \]

\[ 1\text{-SAT } O(m) \]
Algorithmic Complexity of SAT

\begin{align*}
\text{SAT} & \quad 2^n \\
k\text{-SAT} & \quad 2^{n(1-O(1/k))} \\
\vdots & \\
3\text{-SAT} & \quad 1.308^n \\
2\text{-SAT} & \quad O(m) \\
1\text{-SAT} & \quad O(m)
\end{align*}