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

streaming algs

poly time

Exp time

running time:	n	n^2	n^3	$n!$
less than 10^9:	10^9	$10^{4.5}$	10^3	12

~30K

RUNNING TIME

running time:	n	n^2	n^3	$n!$
less than 10^9:	10^9	$10^{4.5}$	10^3	12

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

Exp-time algs

running time:	$\boxed{n!}$	$\boxed{4^n}$	2^n	$\underline{1.308^n}$
less than 10^9:	12	14	29	77

Lecture 1. Hard

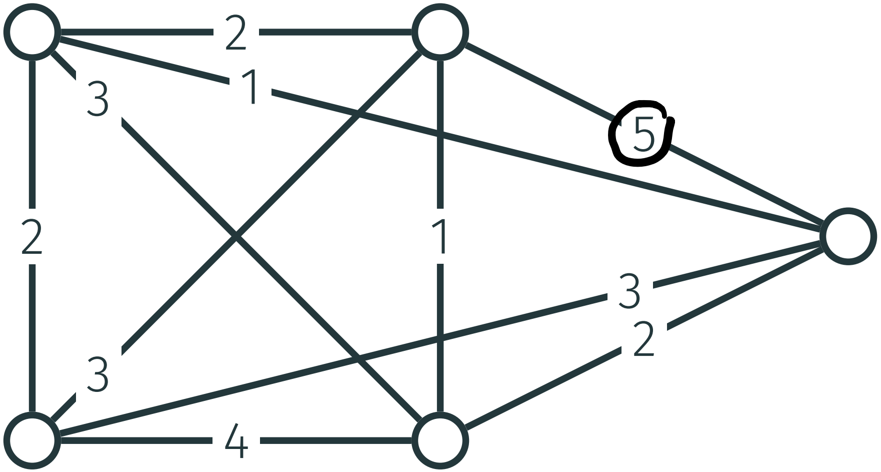
Lecture 2. Approximate soln in poly-time

Today. Exactly. Hard. How hard is it?

Traveling Salesman Problem (TSP)

TRAVELING SALESMAN PROBLEM

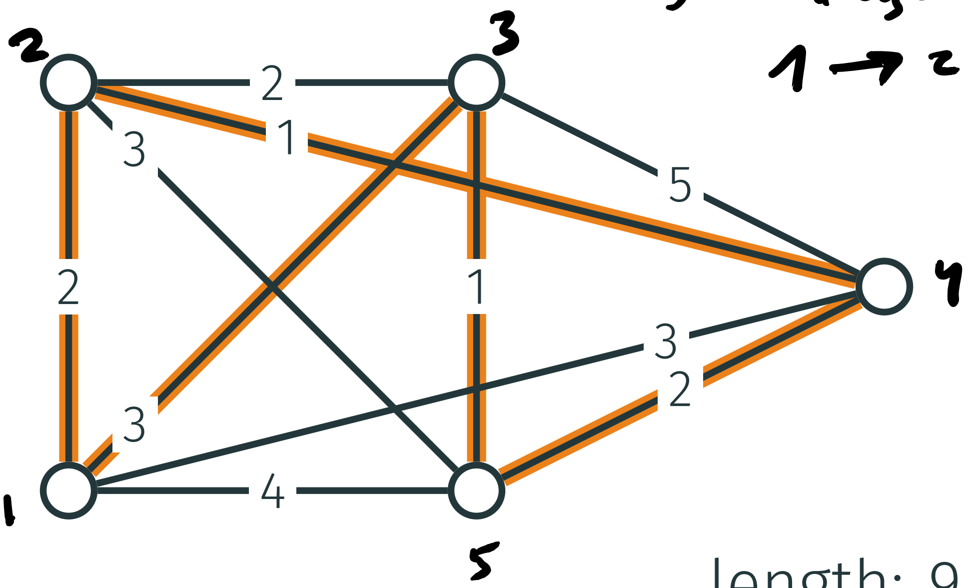
Given a complete weighted graph, find a cycle (or a path) of minimum total weight (length) visiting each node exactly once



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



Find a cycle of min weight

1 → 2 → 4 → 5 → 3
↓
1

length: 9

ALGORITHMS

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

ALGORITHMS

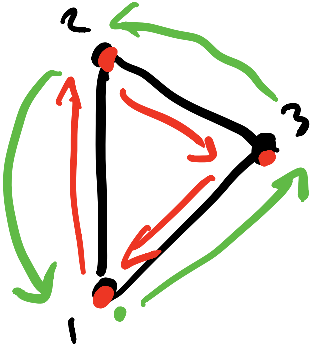
- Classical optimization problem with countless number of real life applications (see Lecture 1)
- No polynomial time algorithms known

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

BRUTE FORCE SOLUTION

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



$n = 3$

1 2 3

1 3 2

2 1 3

2 3 1

3 1 2

3 2 1

Brute Force Solution

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

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

We'll see

$n! \approx n^n$

- 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

1962
still remains
best known
for TSP

was invented for TSP

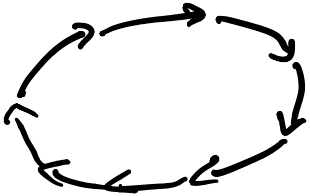
- Dynamic programming is one of the most powerful algorithmic techniques

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

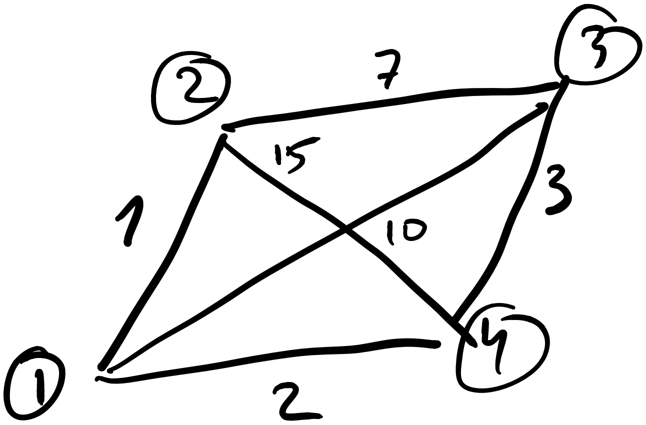
- Dynamic programming is one of the most powerful algorithmic techniques
- Rough idea: express a solution for a problem through solutions for smaller subproblems
- Solve subproblems one by one. Store solutions to subproblems in a table to avoid recomputing the same thing again



SUBPROBLEMS

Start at 1.
End at i

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

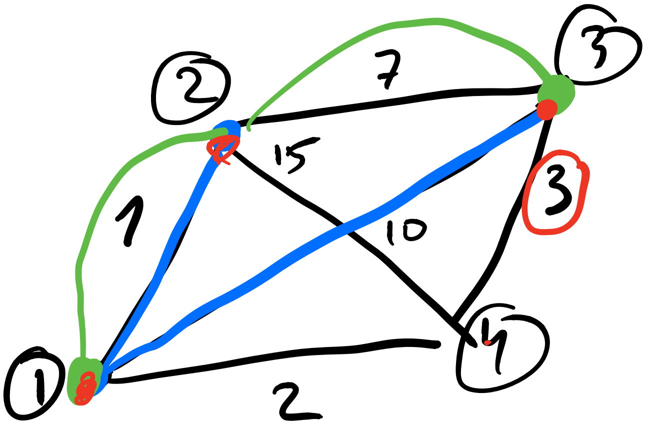


For any set $S \subseteq \{1, \dots, n\}$

For any vertex $i \in \{1, \dots, n\}$

$C(S, i)$ = length of shortest path that.

1. Starts at 1
2. Ends at i
3. Visits every vertex from S exactly once



$C(\{1,2\},2)=\underline{1}$

$C(\{1,3\},3)=10$

$C(\{\underline{\underline{1}}\},1)=0$

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

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

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

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

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

SUBPROBLEMS

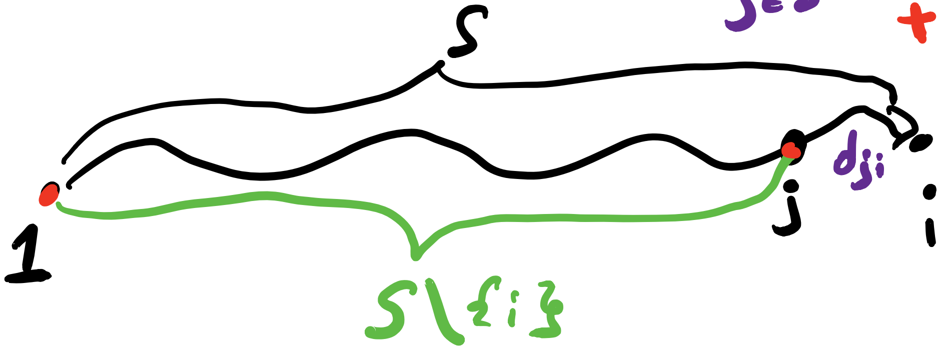
- For a subset of vertices $\boxed{S} \subseteq \{1, \ldots, n\}$ containing the vertex 1 and a vertex $\boxed{i} \in S$, let $\underline{C(S, i)}$ be the length of the shortest path that starts at 1, ends at i and visits all vertices from S exactly once
- $C(\{\underline{1}\}, \underline{1}) = \underline{0}$ and $C(\underline{S}, \underline{1}) = \underline{+\infty}$ when $|S| > 1$

$C(S, i)$

RECURRENCE RELATION

- Consider the second-to-last vertex *j* on the required shortest path from 1 to *i* visiting all vertices from S

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



subpath of shortest path is shortest

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 $C(\underline{\underline{S}}, i) = \underset{j}{\min}\{C(S - \{i\}, \underline{\underline{j}}) + \underline{\underline{d_{ji}}}\}$, 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 $\underline{C(S - \{i\}, j)}$ have already been computed

ORDER OF SUBPROBLEMS

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

ALGORITHM

$$\underline{C(\overset{S}{*}, \overset{i}{*})} \leftarrow +\infty$$

$$C(\{1\}, 1) \leftarrow 0$$

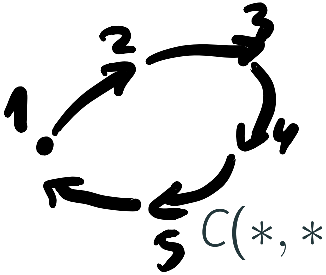
ALGORITHM

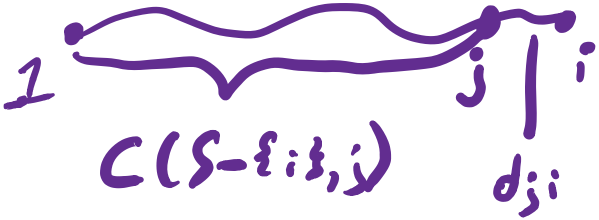
```
$C(*, *) \leftarrow +\infty$
$C(\{1\}, 1) \leftarrow 0$
for s from 2 to $n$:
    for all $1 \in S \subseteq \{1, \ldots, n\}$ of size $s$:
```

size of S: $s = |S|$

$C(S, i)$

ALGORITHM



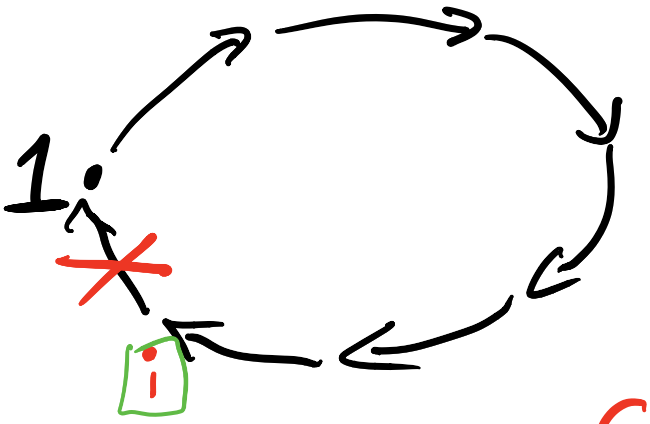


```
$C(*,*) \leftarrow +\infty$
$C(\{1\},1) \leftarrow 0$
for s from 2 to $n$:
    for all $1 \in S \subseteq \{1,\dots,n\}$ of size s:
        for all $i \in S, i \neq 1$:
            for all $j \in S, j \neq i$
                $\boxed{\underline{C(S,i)} \leftarrow \min\{C(S,i), \underline{C(S-\{i\},j)} + \underline{d_{ji}}\}}$
```

the last vertex of path

second-to-last

C(S, i) - always at 1



length of cycle $= C(\{1, \ldots, n\}, i) + d_{i,1}$

length of <u>shortest</u> cycle $=$

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

ALGORITHM

Run-time $\leq 2^n \cdot n^3$

```
$C(*,*) \leftarrow +\infty$
$C(\{1\},1) \leftarrow 0$
for $s$ from 2 to $n$:                                   $n$
    for all $1 \in S \subseteq \{1,\ldots,n\}$ of size $s$:       $2^n$
        for all $i \in S, i \neq 1$:                      $n$
            for all $j \in S, j \neq i$                   $n$
                $C(S,i) \leftarrow \min\{C(S,i), C(S-\{i\},j)+d_{ji}\}$
return $\min_i\{C(\{1,\ldots,n\},i)+d_{i,1}\}$
```

go over all subsets of $\{1,\ldots,n\}$

2^n

Run-time $\leq 2^n \cdot n^2$

shortest cycle

shortest path

$\approx 2^n$

Satisfiability Problem (SAT)

SAT

n 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 1 1 1

SAT

SAT

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

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

UNSAT

k-SAT

$$\begin{aligned}\phi(x_1, \ldots, x_n) = &(x_1 \vee \neg x_2 \vee \ldots \vee x_k) \quad \wedge \\ &\qquad\qquad \ldots \qquad\qquad\quad \wedge \\ &(x_2 \vee \neg x_3 \vee \ldots \vee x_8)\end{aligned}$$

k-SAT

$$\begin{aligned}\phi(x_1, \ldots, x_n) = & (x_1 \vee \neg x_2 \vee \ldots \vee x_k) \quad \wedge \\ & \qquad \ldots \qquad\qquad \wedge \\ & (x_2 \vee \neg x_3 \vee \ldots \vee x_8)\end{aligned}$$

ϕ is satisfiable if

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

Otherwise, ϕ is unsatisfiable

k-SAT

$$\begin{aligned} \phi(x_1, \ldots, x_n) = &(x_1 \lor \neg x_2 \lor \ldots \lor \underline{\underline{x_k}}) \quad \land \\ &\qquad\qquad \ldots \qquad\qquad\quad \land \\ &(x_2 \lor \neg x_3 \lor \ldots \lor x_8) \end{aligned}$$

ϕ is satisfiable if

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

Otherwise, ϕ is unsatisfiable

$\boxed{n}$ Boolean vars, $\boxed{m}$ clauses

k-SAT

$$\begin{aligned}\phi(x_1,\ldots,x_n) = & (x_1 \vee \neg x_2 \vee \ldots \vee x_k) \quad \wedge \\ & \qquad \ldots \qquad\qquad \wedge \\ & (x_2 \vee \neg x_3 \vee \ldots \vee x_8)\end{aligned}$$

ϕ is satisfiable if

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

Otherwise, ϕ is unsatisfiable

n Boolean vars, m clauses

k-SAT is SAT where clause length $\leq k$

k-SAT. EXAMPLES

3-SAT

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_2 \vee \neg x_3)$$

k-SAT. EXAMPLES

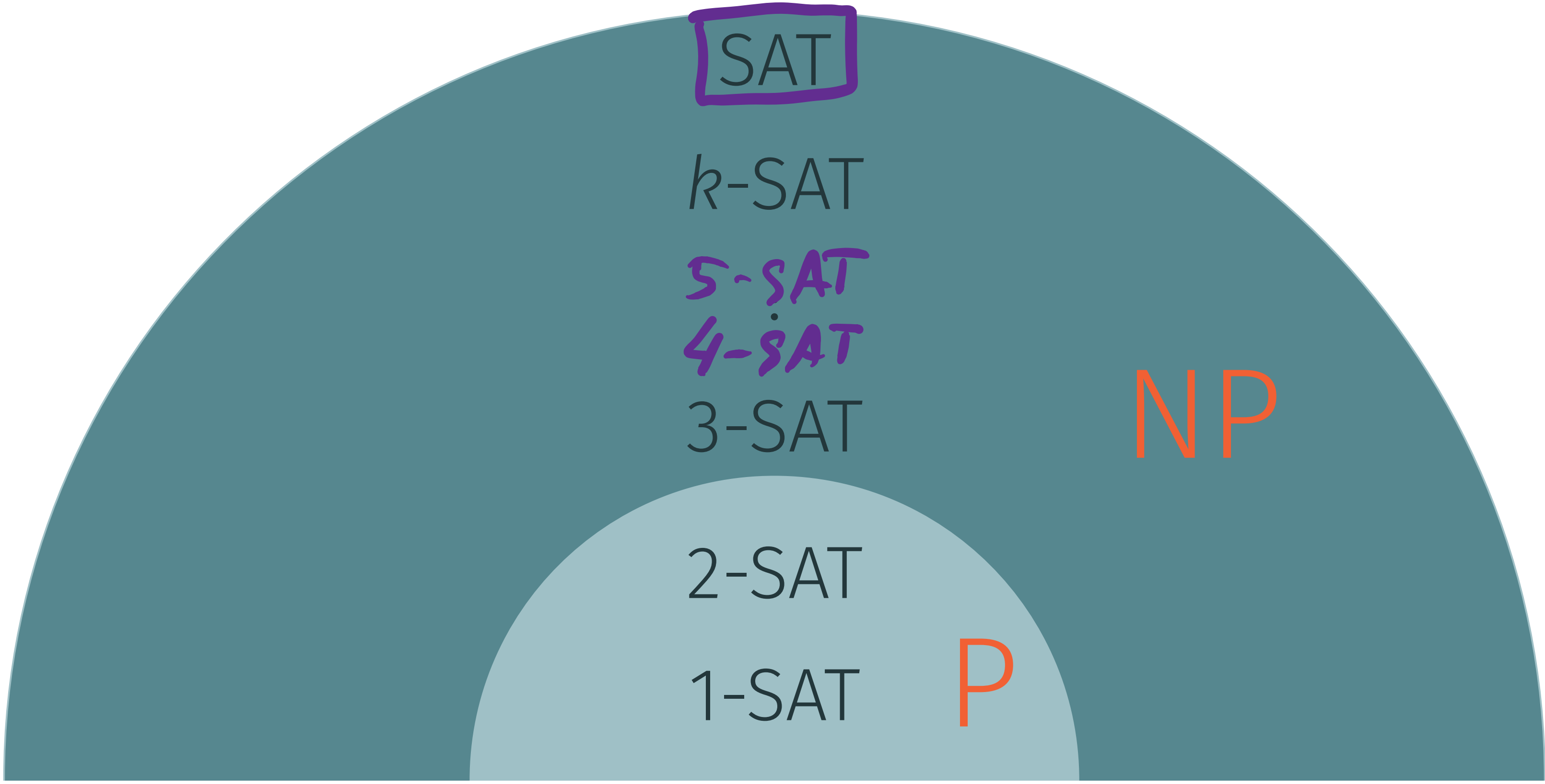
$$(x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2) \land (\neg x_1 \lor x_3) \land (x_2 \lor \neg x_3)$$

1-SAT

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

Complexity of SAT

COMPLEXITY OF SAT



SAT

k-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:

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

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 $\underline{O^*(2^n)}$

$x_1, \ldots, x_n \in \{0, 1\}$

$\sim 2^n$ such assignments

For each assignments 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 $\gtrsim 2^n$

3-SAT

- $(x_1 \vee x_2 \vee x_9) \wedge \ldots \wedge (x_2 \vee \neg x_3 \vee x_8)$

3-SAT

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

x_1	x_2	x_9
~~0~~	~~0~~	~~0~~
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Instead of checking $\{0,1\}^n$
can check only those
that don't have $x_1=x_2=x_9=0$

$2^n \cdot \frac{7}{8}$ 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: $x_1=0$ $x_2=0$ $x_9=1$

3-SAT

- $\overset{1}{\cancel{(x_1 \lor x_2 \lor x_9)}} \land \ldots \land (x_2 \lor \neg x_3 \lor x_8)$ $(\overset{0}{\cancel{\neg x_1}} \lor x_7 \lor x_5)$

- Consider three sub-problems:

Case I: $\underline{x_1 = 1}$ Replace $x_1 \to 1$; $\neg x_1 \to 0$ 3-SAT$(n-1)$

Case II: $x_1 = 0, x_2 = 1$ Replace $x_1 \to 0$; $\neg x_1 \to 1$; $x_2 \to 1$; $\neg x_2 \to 0$ 3-SAT$(n-2)$

Case III: $x_1 = 0, x_2 = 0, x_9 = 1$ 3-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 $(x \vee y \vee z)$
  → → 3-SAT(formula $x=1$)
  → → 3-SAT(Formula $x=0$ $y=1$)
  → → 3-SAT(formula $x=y=0$ $z=1$)
      If one of these is TRUE,
          Then RETURN TRUE
      Else
          Then RETURN FALSE
```

$T(n)$ – Run-time on flas with n variables

$$T(n) \leq \underline{T(n-1)} + \underline{T(n-2)} + \underline{T(n-3)}$$

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

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

$$
\begin{aligned}
T(n) &\le T(n-1) + T(n-2) + T(n-3) \\
\text{ind hypothesis} \quad &\le \underline{1.85^{n-1}} + \underline{1.85^{n-2}} + \underline{1.85^{n-3}} \\
&= \underline{1.85^n}\left(\frac{1}{1.85} + \frac{1}{1.85^2} + \frac{1}{1.85^3}\right) \\
&< \underline{1.85^n}\,\underline{(0.991)} \\
&< \underline{1.85^n}
\end{aligned}
$$

$T(n) \le T(n-1) + T(n-2) + \dots + T(n-k)$

$\forall$ constant, k-SAT in $(2-\varepsilon_k)^n$

SAT in time 2^n

3-SAT. ANALYSIS

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

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

- There are even faster algorithms: $\underline{1.308^n}$ [HKZZ19]

$\uparrow$

10^{-24}

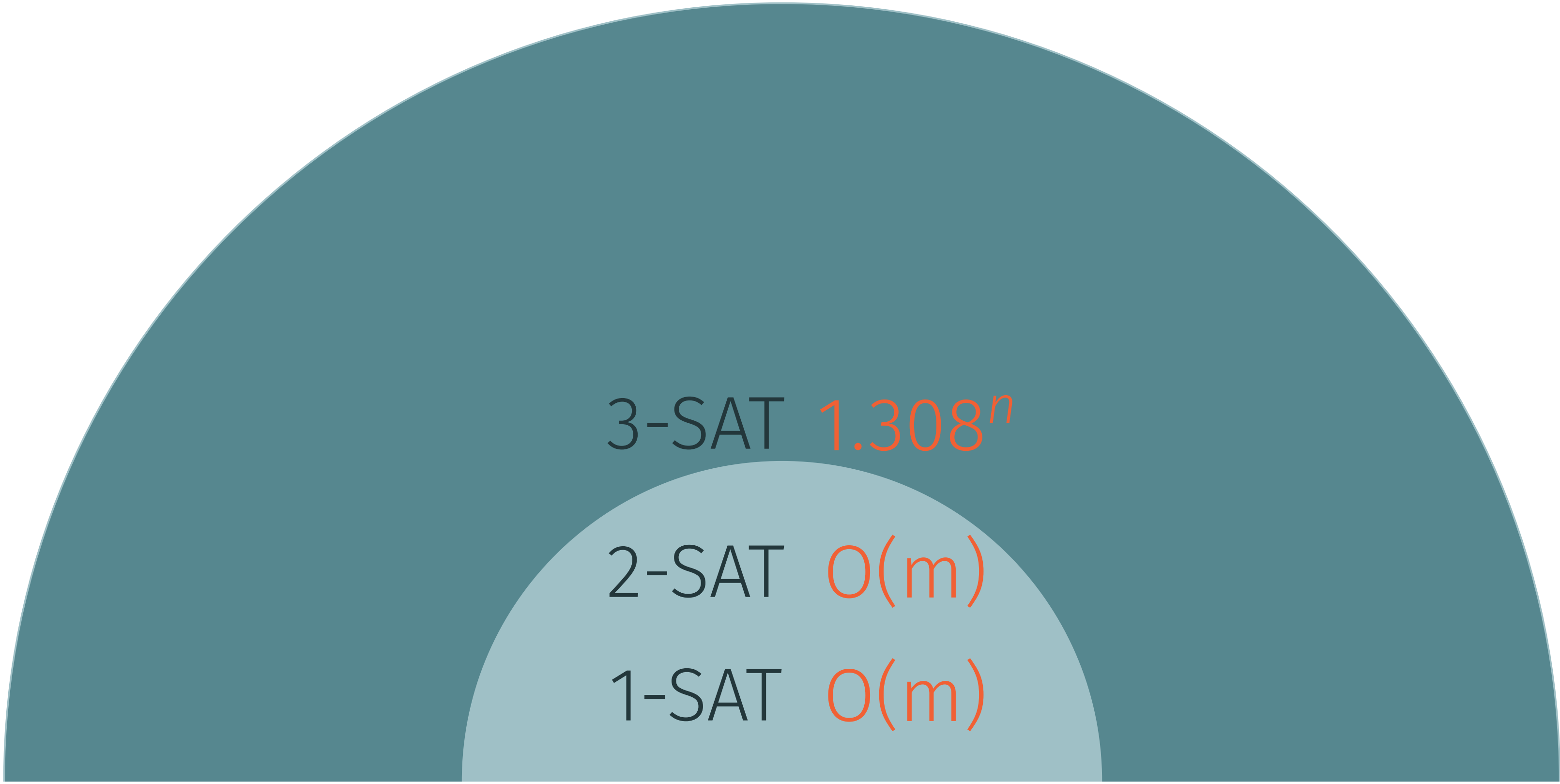
How hard can SAT be?

Algorithmic Complexity of SAT

2-SAT $O(m)$

1-SAT $O(m)$

Algorithmic Complexity of SAT

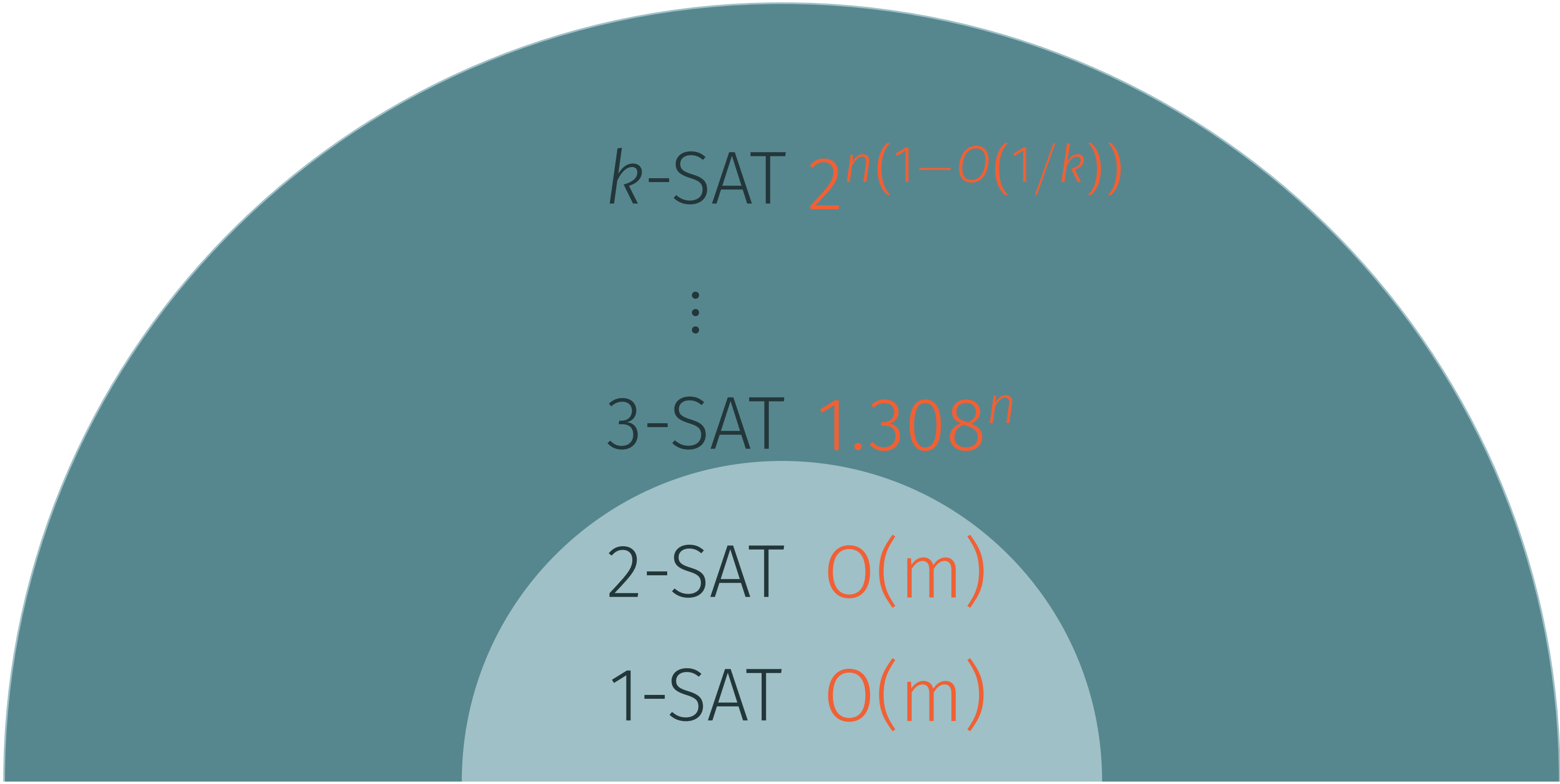


3-SAT 1.308^n

2-SAT O(m)

1-SAT O(m)

Algorithmic Complexity of SAT



k-SAT $2^{n(1-O(1/k))}$

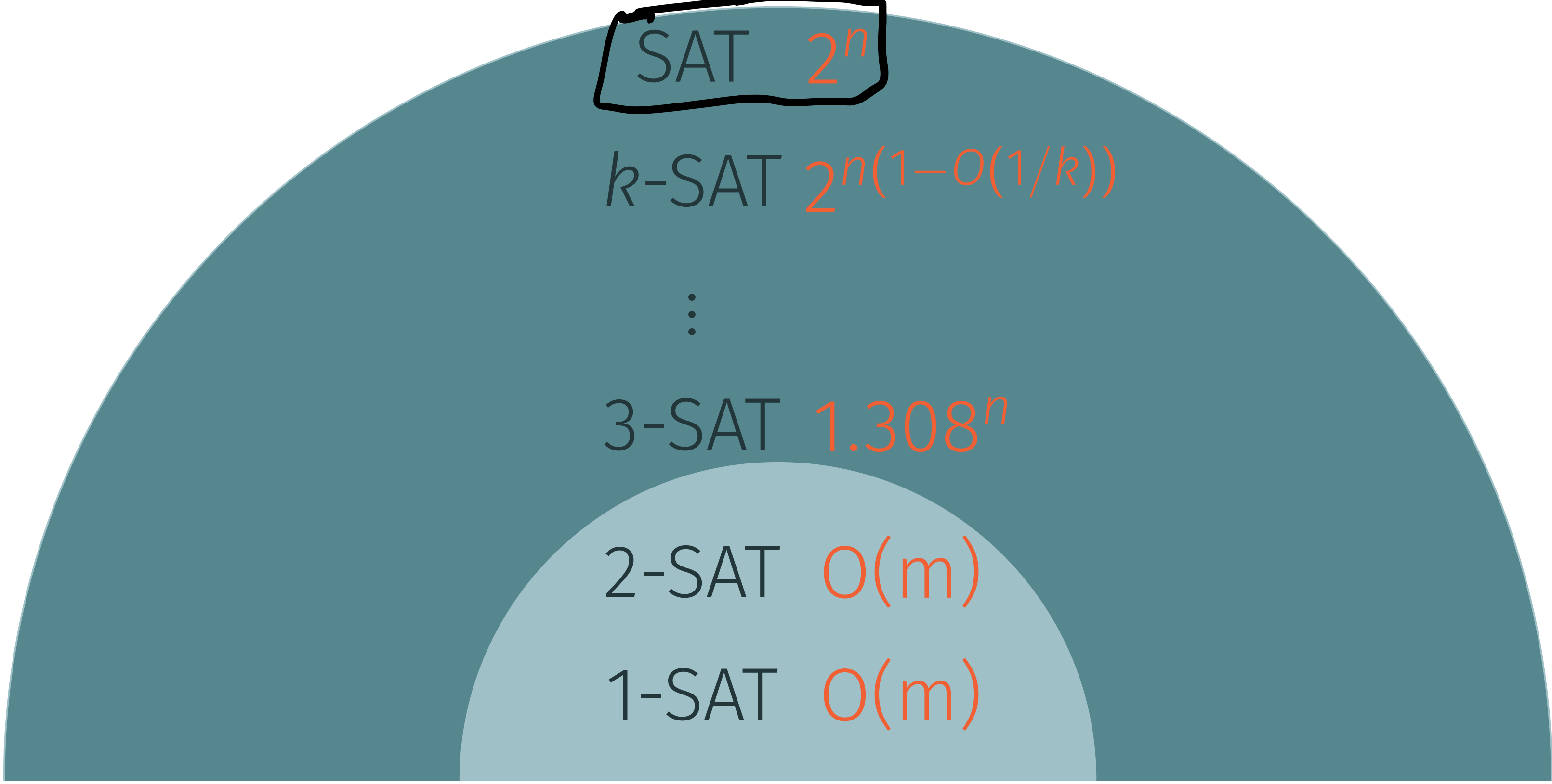
$\vdots$

3-SAT 1.308^n

2-SAT O(m)

1-SAT O(m)

Algorithmic Complexity of SAT



Conj

$\boxed{\text{SAT} \quad 2^n}$

k-SAT $2^{n(1-O(1/k))}$

⋮

3-SAT 1.308^n

2-SAT $O(m)$

1-SAT $O(m)$