Algorithms and Complexity

Lecture 9

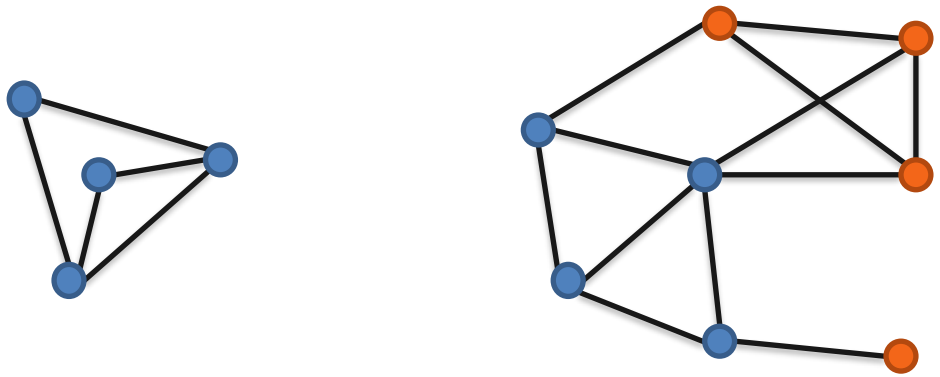
More on Treewidth

and

Randomized Algorithms (and more)

Subgraph Isomorphism

- Given a *pattern* graph P and *host* graph G
- Decide whether G has subgraph isomorphic to P



- Let $n = |V(G)|$ and $k = |V(P)|$

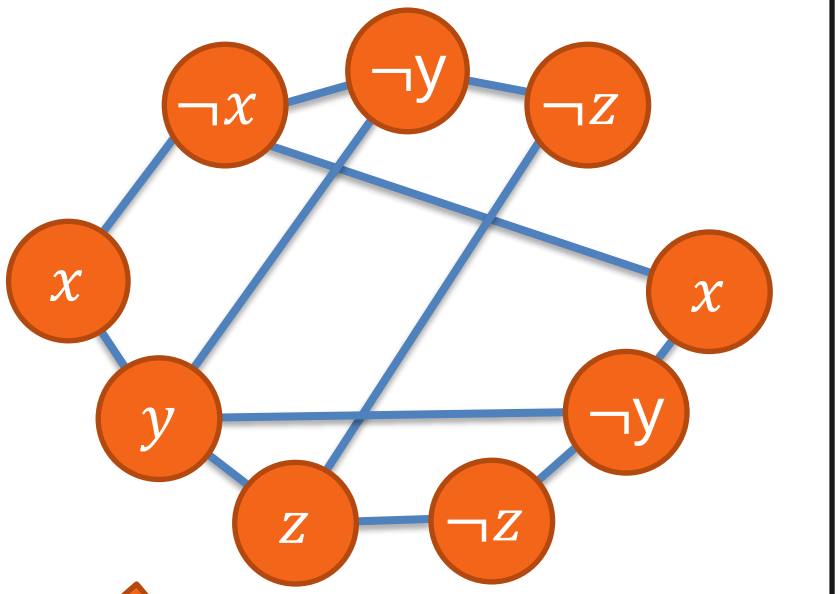
Subgraph Isomorphism

- Trivial algorithm: try all $\binom{n}{k}$ options
  - > gives a $2^{O(n \log n)}$-time algorithm

- Planar graphs: $2^{O(k)}n$-time algorithm
  - [Dorn 2009]: “embedded dynamic programming”

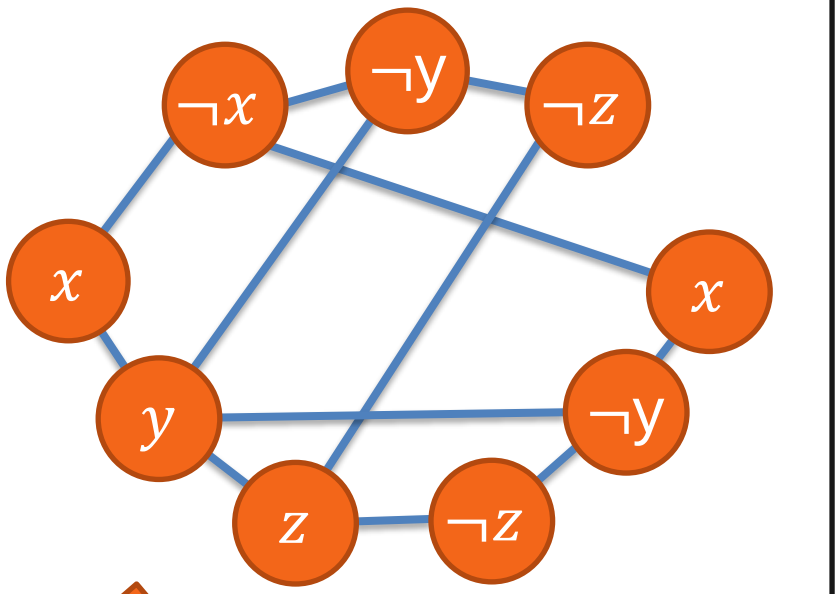
What is the best we can do?

Lower Bound Techniques

3-SAT	Independent Set
$(x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (x \vee \neg y \vee \neg z)$	

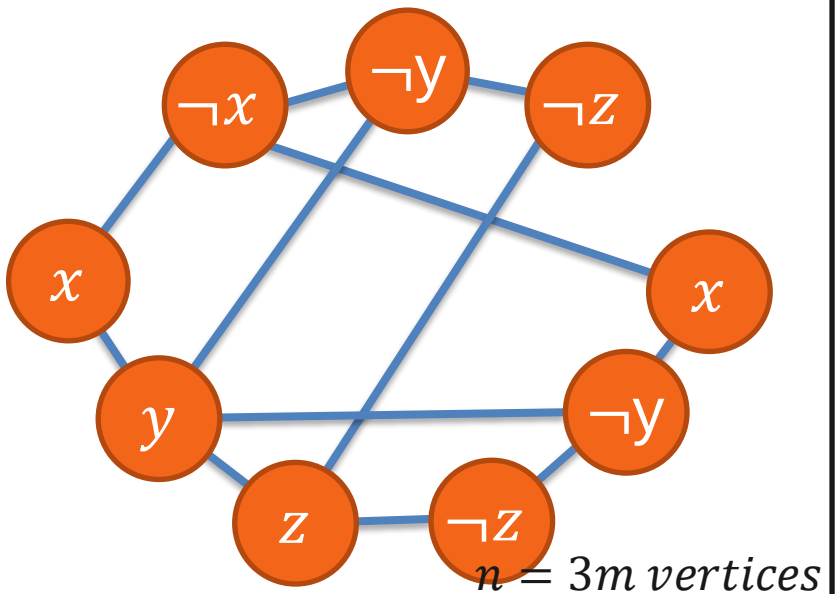
NP-completeness

Exponential Time Hypothesis

3-SAT	Independent Set
$(x \vee y \vee z)$ $\wedge$ $(\neg x \vee \neg y \vee \neg z)$ $\wedge$ $(x \vee \neg y \vee \neg z)$ m clauses	

ETH: No $2^{o(m)}$-time algorithm for 3-SAT.

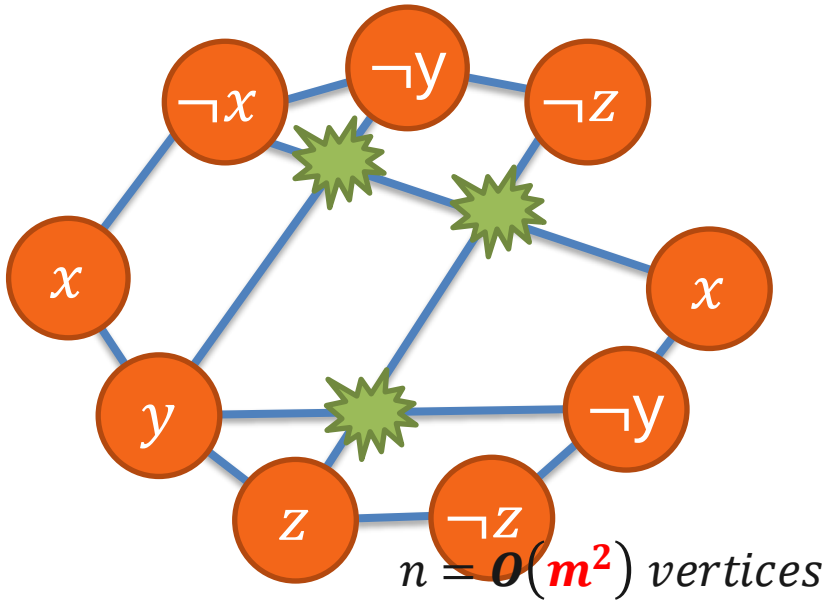
Exponential Time Hypothesis

3-SAT	Independent Set
$(x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (x \vee \neg y \vee \neg z)$ m clauses	 $n = 3m$ vertices

ETH: No $2^{o(m)}$-time algorithm for 3-SAT.

No $2^{o(n)}$-time algo for independent set.

Exponential Time Hypothesis

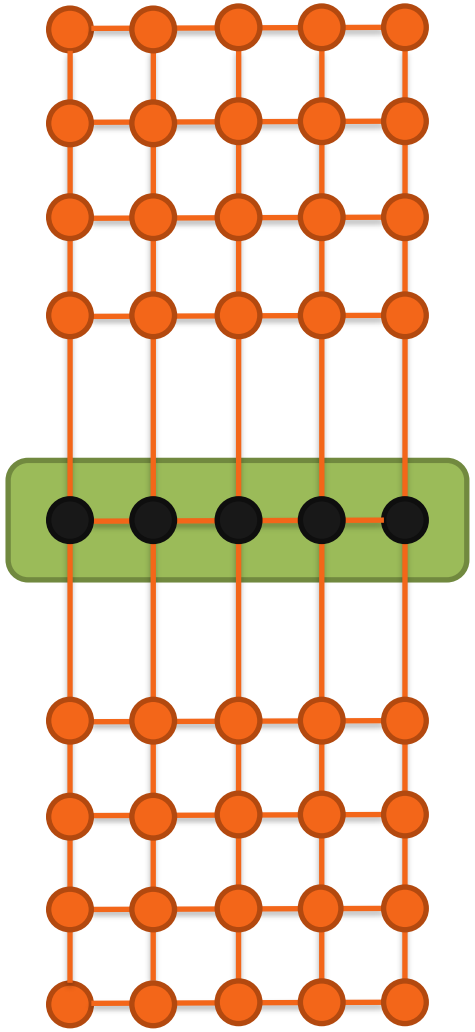
3-SAT	Independent Set in Planar Graphs
$(x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (x \vee \neg y \vee \neg z)$ m clauses	 $n = O(m^2)$ vertices

ETH: No $2^{o(m)}$-time algorithm for 3-SAT.

No $2^{o(\sqrt{n})}$-time algo for independent set.

The Square Root Phenomenon

- ETH: there are no $2^{o(\sqrt{n})}$-time algorithms for many problems on planar graphs
- Using *dynamic programming* and *treewidth*, we can design **$2^{o(\sqrt{n})}$-time algorithms**
- *Planar Separator Theorem*: every planar graph has a balanced separator of size $O(\sqrt{n})$



What about Subgraph Isomorphism?

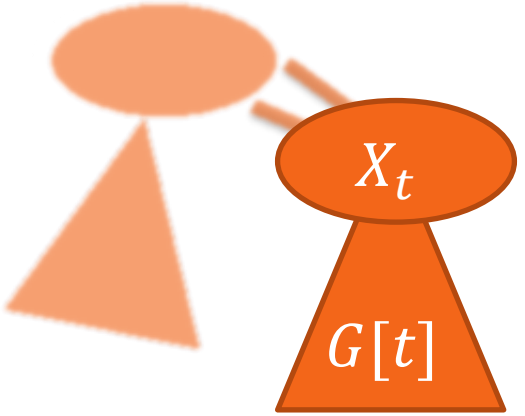
- ETH tells us that $2^{\Theta(\sqrt{n})}$ is optimal for many problems on planar graphs

- Surprise! The "right" running time for Subgraph Isomorphism on n-vertex planar graphs is

$$2^{\Theta(n\,/\,\log n)}$$

Algorithm for Subgraph Isomorphism

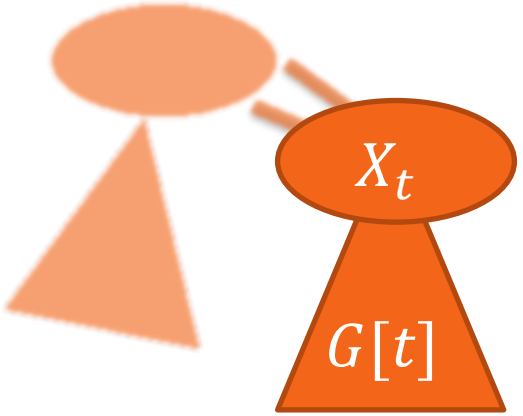
- Algorithm uses dynamic programming on a tree decomposition of the host graph G (i.e. the larger graph)
- When doing DP with TD, we are always looking at a subgraph $G[t]$ induced by vertices in bag X_t and the bags below it



Algorithm for Subgraph Isomorphism

- Algorithm uses dynamic programming on a tree decomposition of the host graph G (i.e. the larger graph)
- When doing DP with TD, we are always looking at a subgraph $G[t]$ induced by vertices in bag X_t and the bags below it

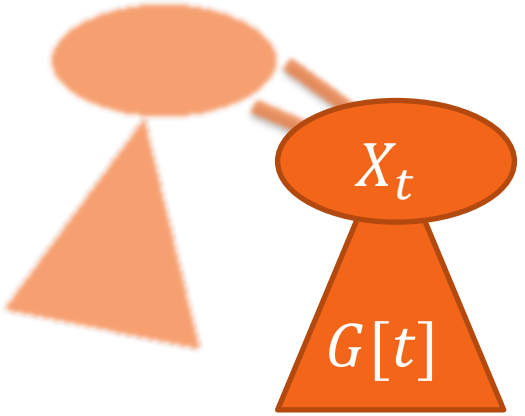
- We need a notion of *partial solution*
- Want: a mapping of vertices of G to P



Algorithm for Subgraph Isomorphism

- Algorithm uses dynamic programming on a tree decomposition of the host graph G (i.e. the larger graph)
- When doing DP with TD, we are always looking at a subgraph $G[t]$ induced by vertices in bag X_t and the bags below it

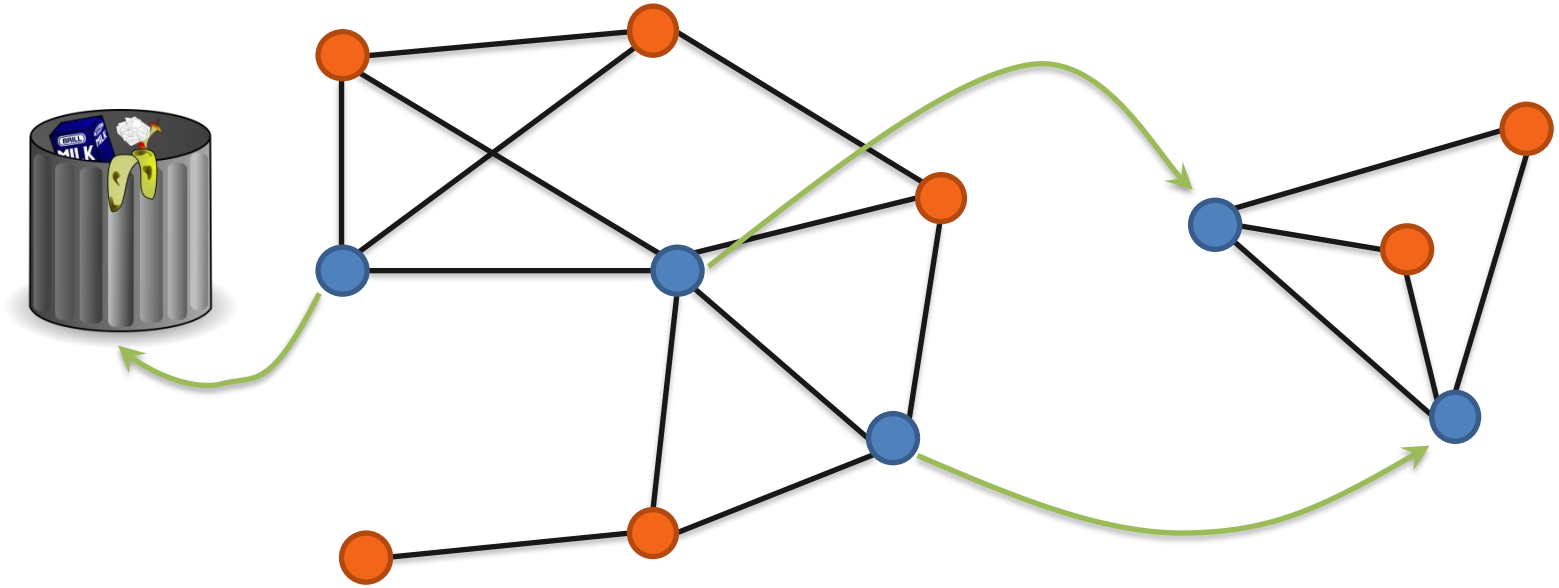
- We need a notion of *partial solution*
- Want: a mapping of vertices of G to P
- Partial solution: mapping of vertices of $G[t]$ to a subgraph of P



Algorithm for Subgraph Isomorphism

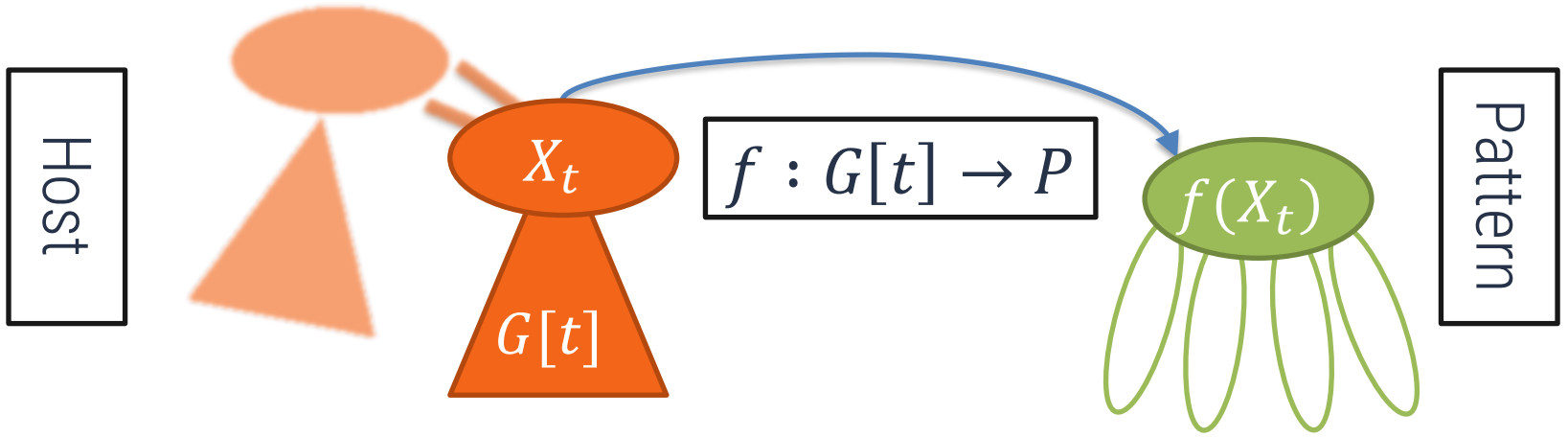
- Host graph G has a separator X_t of size $O(\sqrt{n})$
- Guess the mapping of X_t to vertices of P

→ $n^{O(\sqrt{n})} = 2^{O(\sqrt{n}\log n)}$ cases

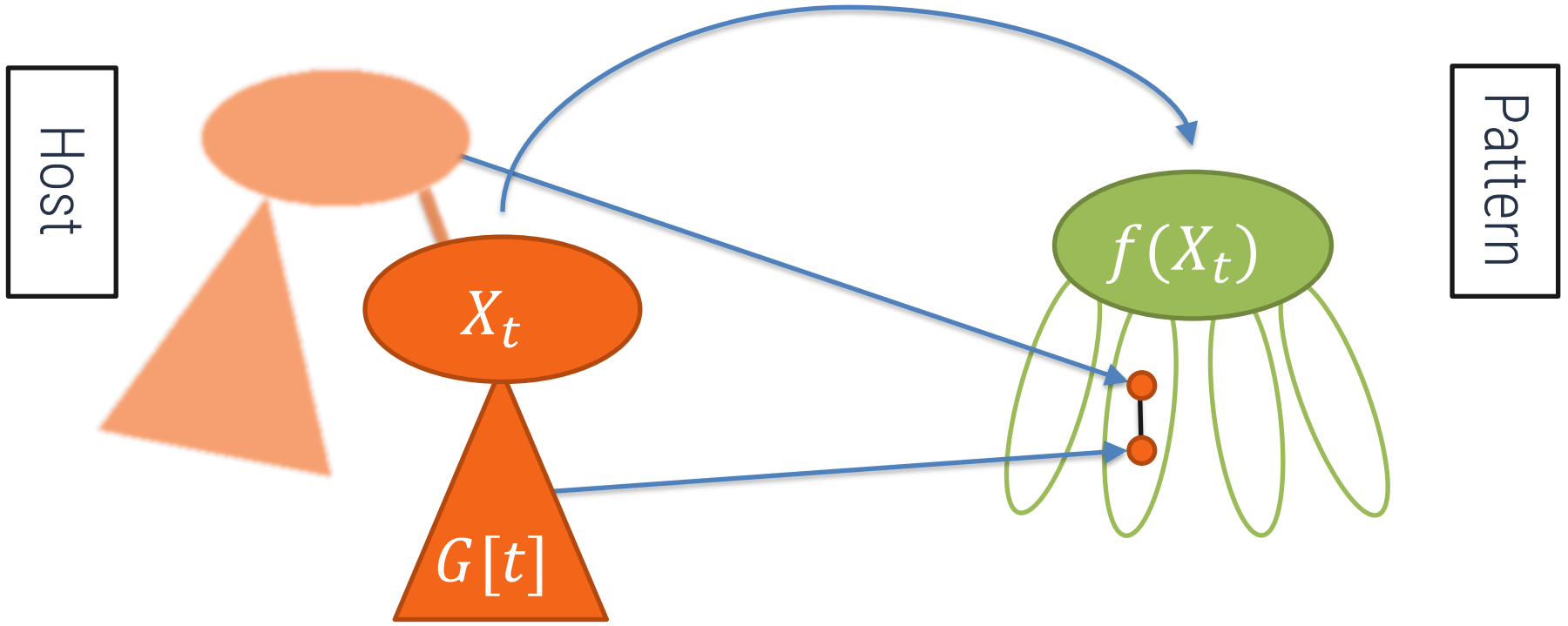


Algorithm for Subgraph Isomorphism

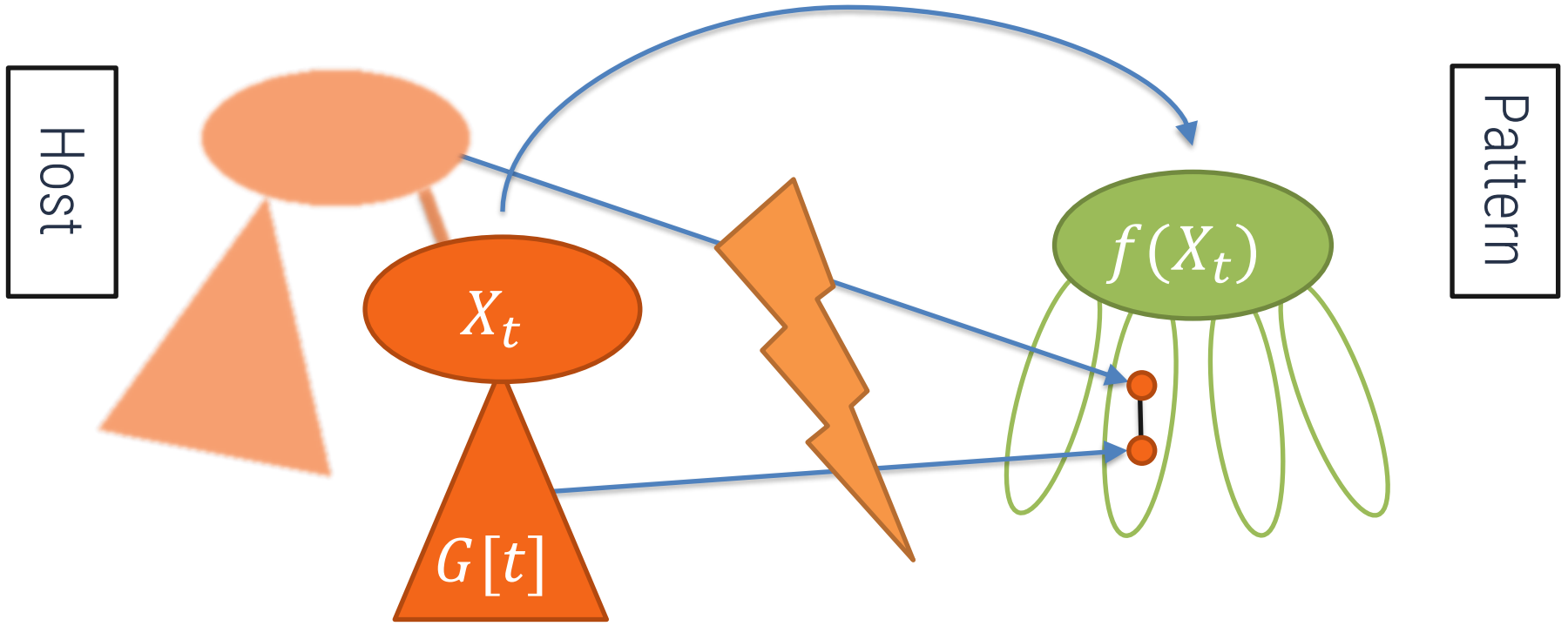
- Partial solution: mapping of vertices of X_t to a subgraph of P
- $f(X_t)$ should also be a separator of P



Connected Components



Connected Components



Connected Components

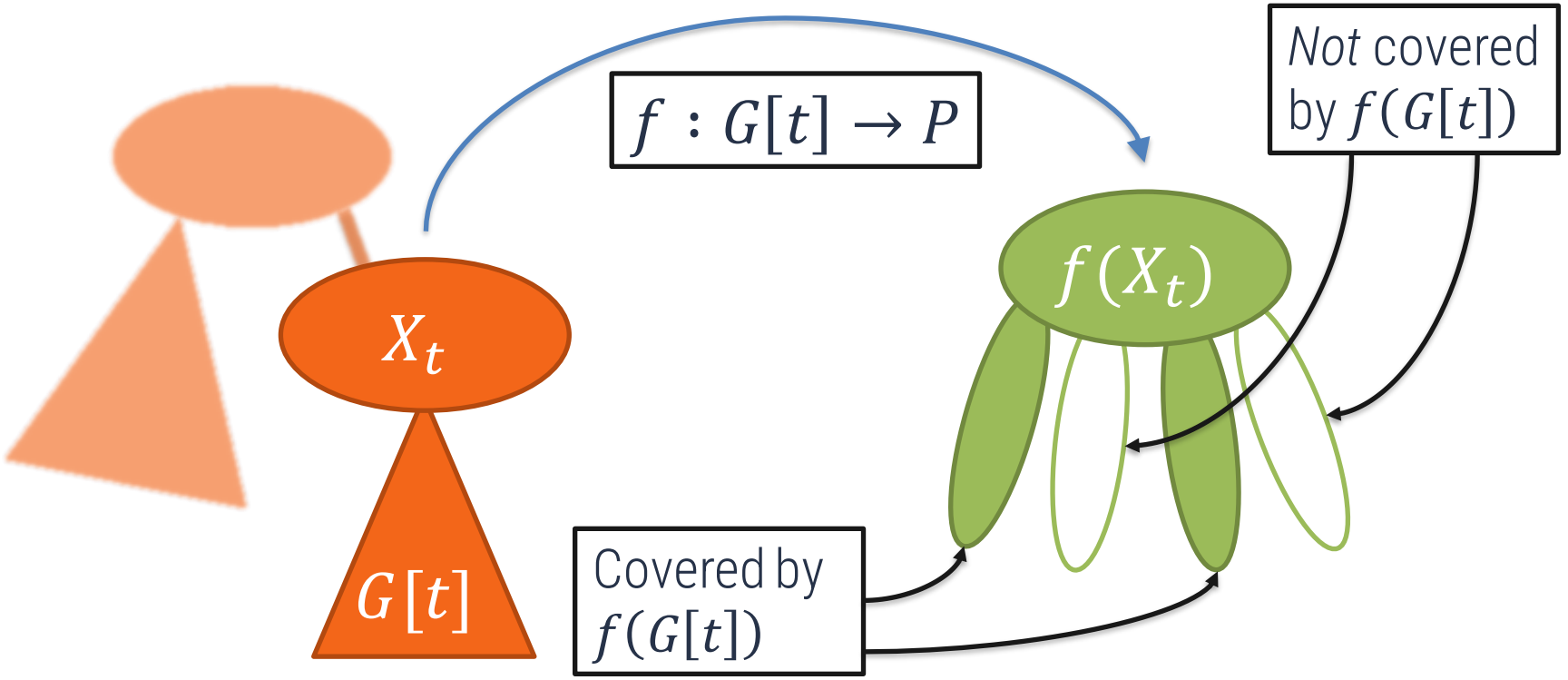
- Observation: connected component of

$$P \setminus f(X_t)$$

is either *completely* in $f(G[t])$ or completely outside of $f(G[t])$.

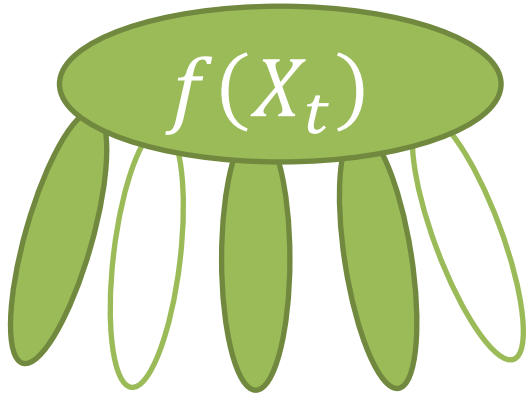
- Dynamic Programming: track $f(G[t])$ and the *subset of connected components*.

Connected Components

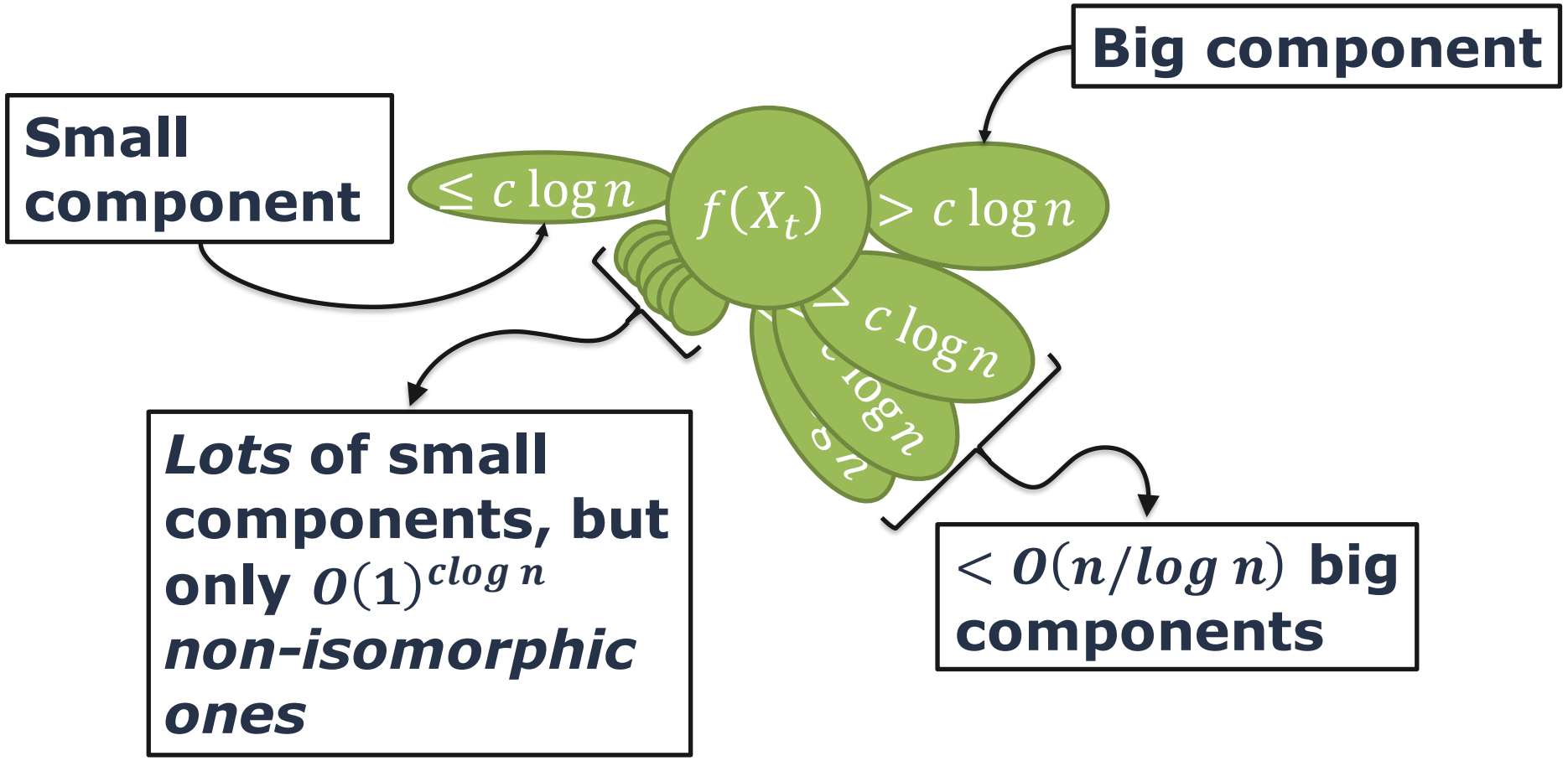


Connected Components

- In the worst case $2^{\Omega(n)}$ subsets of connected components
- …but only if the connected components have size $O(1)$

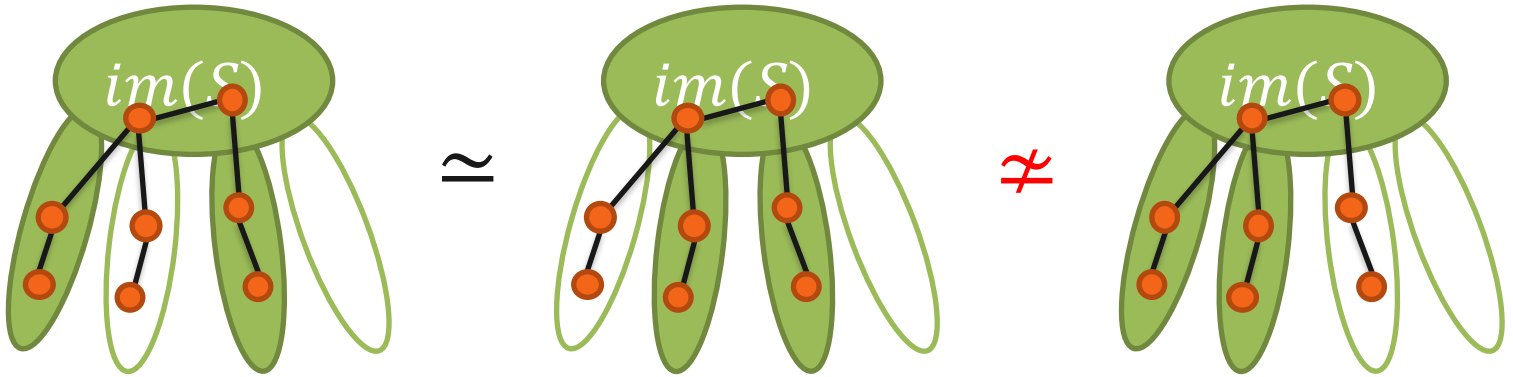


Big and Small Components



Solution equivalence

- Partial solution are **equivalent** if they have the same number of connected components from each **component isomorphism class**



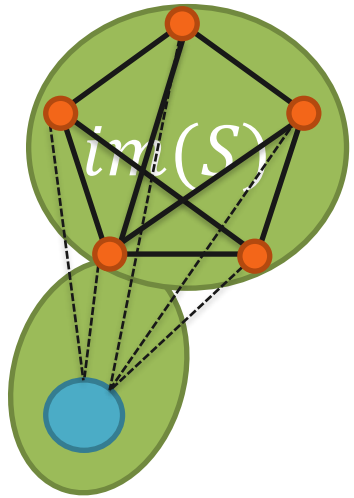
Solution equivalence

- There are $O(1)^n$ non-isomorphic n-vertex planar graphs
- Thus, $O(1)^{c \log n}$ cases for the components themselves (for small c, this is $O(n^{\epsilon})$)
- but X_t can have $\Omega(\sqrt{n})$ vertices… what about neighborhood?

Big neighbourhood

- Claim: at most $O(\sqrt{n})$ components have ≥ 5 vertices in their neighborhood

- …planar graphs are sparse
- …and exclude K_5

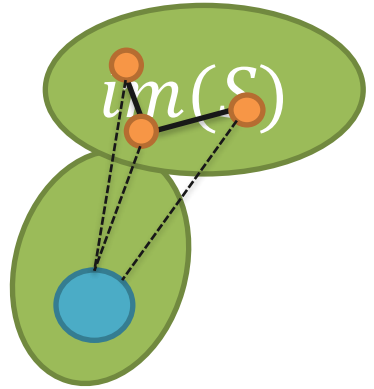


Small neighbourhood

- Contract connected components with small neighborhood until all neighborhoods are full

- …planar graphs have few triangles

More general case shown by [Gajarský et al. 2013]



Analysis

type	# cases
BIG component	$2^{O\left(\frac{n}{c \log n}\right)}$
SMALL component, BIG nbh.	$2^{O(\sqrt{n})}$
SMALL component, SMALL nbh.	$O(1)^{c \log n} \cdot 4^{c \log n} \cdot \sqrt{n}$
$f(X_t)$	$n^{O(\sqrt{n})}$

- Running time of DP algo is dominated by the cases for the big components, $2^{O(n/\log n)}$

Lower Bound

- Reduce from 3-SAT with $m = O(n)$ clauses
- Variables $x_1, \ldots x_n$; Clauses $c_{n+1,\ldots,}c_{n+m+1}$

- No $2^{o(n)}$ algorithm by Sparsification Lemma

Lower Bound

- Reduce from 3-SAT with $m = O(n)$ clauses
- Variables $x_1, \ldots x_n$; Clauses $c_{n+1,\ldots,}c_{n+m+1}$
- Build bitstrings for variables/clauses

00001 11110 01111 10000

Index of C/V in binary

- Length $O(\log n)$

Lower Bound

- Reduce from 3-SAT with $m = O(n)$ clauses
- Variables $x_1, \ldots x_n$; Clauses $c_{n+1,\ldots,}c_{n+m+1}$
- Build bitstrings for variables/clauses

00001 11110 01111 10000

Bitwise complement

- Length $O(\log n)$

Lower Bound

- Reduce from 3-SAT with $m = O(n)$ clauses
- Variables $x_1, \ldots x_n$; Clauses $c_{n+1,\ldots,}c_{n+m+1}$
- Build bitstrings for variables/clauses

00001 11110 01111 10000

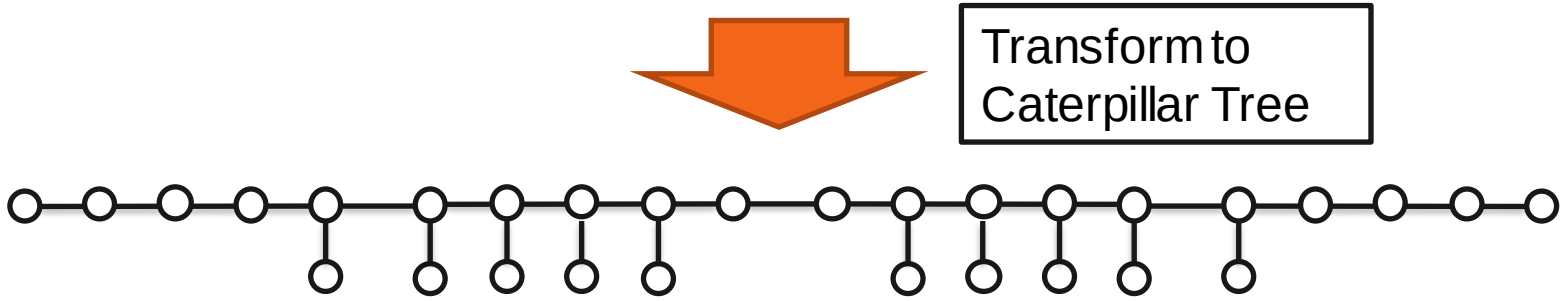
Make Palindromic

- Length $O(\log n)$

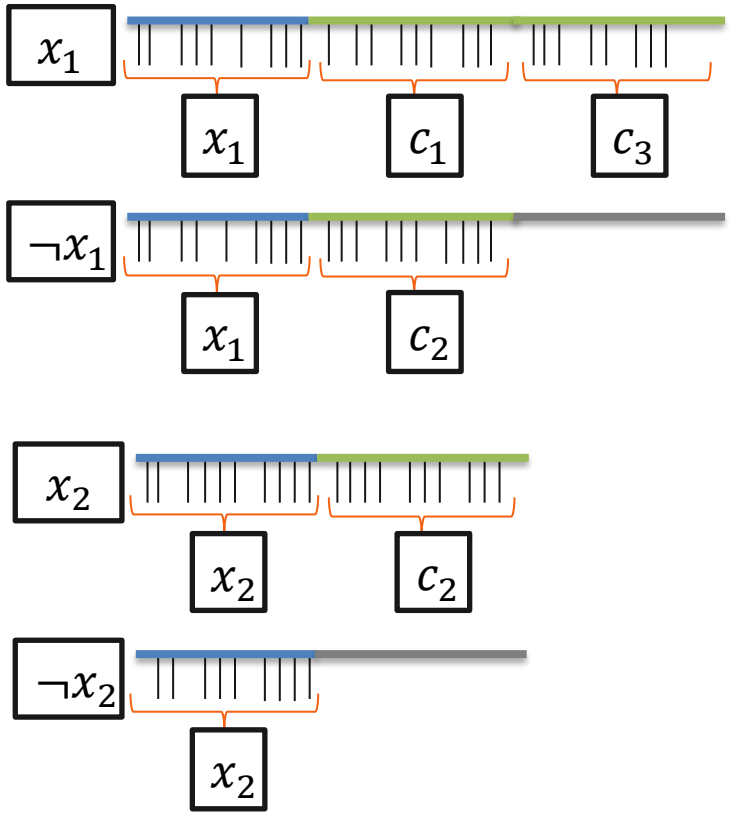
Lower Bound

- Reduce from 3-SAT with $m = O(n)$ clauses
- Variables $x_1, \ldots x_n$; Clauses $c_{n+1,\ldots,}c_{n+m+1}$
- Build bitstrings for variables/clauses

00001 11110 01111 10000

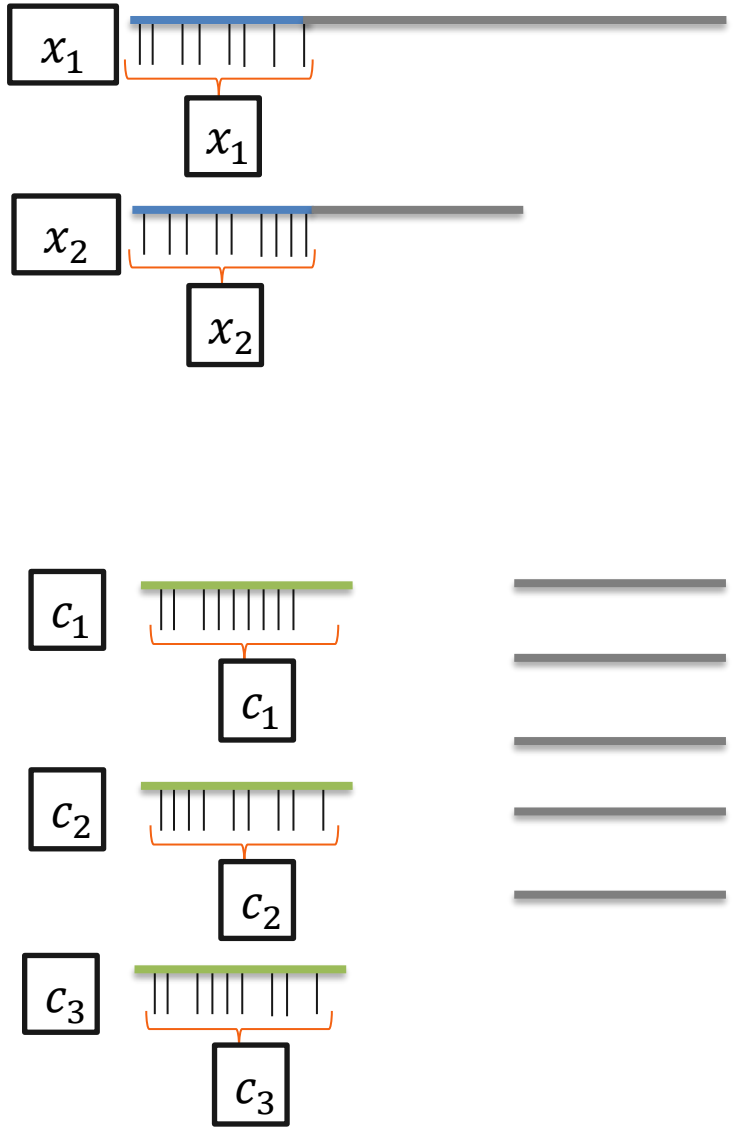


Host graph G



$(x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_5) \wedge (x_1 \vee \cdots$

Guest graph P



Lower Bound

- Forest of caterpillar trees
- Size of instance: $O(n \log n)$
  $\rightarrow 2^{\Omega(n \,/\, \log n)}$ lower bound

- Can add ‘universal’ vertex to make P a tree and G connected, pathwidth 2

Randomized Algorithms

- Algorithm has access to a (hypothetical) subroutine outputting ***a random bit***.
- Performance measures of algorithm (running time, output quality) now are **random variables**:
  - We want guarantees on distributions.
  - Still interested in the worst-case instance!
- Why?!
  - Often simpler, faster, provably better algo's.
  - Error probability often negligible.

Example 1 (a)

- **Zero-Polynomial:** Given a polynomial $p(x)$ of degree d in formula form, is it the zero polynomial?

  $(x^2-x)4 + 2(x^3(3 + x + x^2))$ is a no instance

  $(x^2-x)4 + 4(x - x^2)$ is a yes instance

- How to solve **Zero-Polynomial** quickly?
  - Expanding parentheses may require exponential time!
- **Algo: 1)** Pick $x \in_R \{1, \dots, 2d\}$ uniformly at random;
  2) Return YES iff $p(x) = 0$.
- **Thm:** Let $p(x)$ be a (univariate) polynomial of degree at most d over any field $\mathbb{F}$. Then p has at most d roots.
- If yes instance, $\Pr[Yes] = 1$; if not, $\Pr[Yes] \leq 1/2$

Example 1 (b)

- **Thm:** Let $p(x)$ be a (univariate) polynomial of degree at most d over any field $\mathbb{F}$. Then p has at most d roots.
- Alice has string $\mathrm{a} \in \{0,1\}^n$, Bob has $\mathrm{b} \in \{0,1\}^n$.
- Goal: verify $\mathrm{a} = b$, with little communication.
- In deterministic setting, n bits needed!
- **Algo:**
  - **1)** A field $\mathbb{F}$ of size $\sim 2n$ is fixed (a priori)
  - **2)** Alice picks $x \in_R \mathbb{F}$; sends x and $\mathrm{a}(x) \coloneqq \sum_{i=1}^{n} a_i x^i$
  - **3)** Bob claims equality if $\sum_{i=1}^{n} b_i x^i - a(x) = 0$
- Alice and Bob use $O(\log n)$ bits of communication
- If equal, always correct. If not, $\Pr[corr] \geq \sim 1/2$

Example 1 (c)

- **Thm:** Let $p(x)$ be a (univariate) polynomial of degree at most d over any field $\mathbb{F}$. Then p has at most d roots.
- **Perfect matching:** Let $G = (A \cup B, E)$ be a bipartite graph. Does it have a perfect matching?
- Let $A = \{a_1, \dots, a_n\}$, $B = \{b_1, \dots, b_n\}$ and define

$$M[i,j] = \begin{cases} 0, & \text{if } \{a_i, b_j\} \notin E \\ x_{i,j}, & \text{if } \{a_i, b_j\} \in E \end{cases}$$

Recall $\det(M) = \sum_{\pi \in S_n} (-1)^{sgn(\pi)} \prod_{i=1}^{n} M[i, \pi(i)]$

- $\det(M)$ is a polynomial of degree $\leq n$ in the $x_{i,j}$'s which is non-zero iff G has perfect matching.
- **Algo:** 1) Fix a field $\mathbb{F}$ of size $\sim 2n$, and pick $x_{i,j} \in_R \mathbb{F}$

  2) Use a fast algorithm to compute determinant

Example 2

- Given an n-vertex graph, find a max. size IS.
- A greedy algorithm finds an IS of size $n/(d+1)$, if d is the maximum degree
- A randomized algorithm finds an IS of expected size $\sum_{v \in V} 1/(d_v + 1)$:
  - Pick a random permutation of the vertices
  - Iterate over permuted sequences
    - Include vertex when no neighbor has been included yet

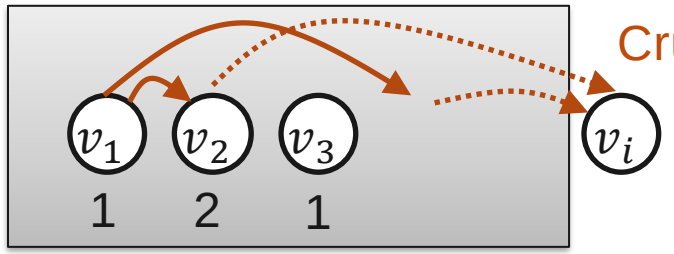
Example 3

- Directed k-path problem: given directed graph G, is there a simple path on at least k vertices.
- In polynomial time if G is acyclic:

Algorithm `kpathDAG`$(G = (V, E), k)$ Assumes G is a directed acyclic graph

Output: Whether there exists a simple path on k vertices in G.

1: Find topological ordering $v_1, \ldots, v_n$ of G i.e. ordering such that for every $(v_i, v_j) \in E$, $i < j$
2: **for** $i = 1, \ldots, n$ **do**
3: **if** $|N^-(v_i)| = 0$ **then**
4: Set $A[v_i] = 1$
5: **else**
6: Set $A[v_i] = \max_{u \in N^-(v_i)} A[u] + 1$
7: **return true** iff $\max_{v \in V} A[v] \geq k$



Crux: cannot collide with previous part of the path

Example 3

- Idea: try to reduce the general case to acyclic case.

Algorithm `kpath1`(G, k) — G is directed

Output: Whether G has a simple path on k-vertices, with constant one-sided error probability.

1: **for** $i = 1 \dots k^k$ **do**
2: Pick for every $v \in V$ a number $c(v) \in \{1, \dots, k\}$ uniformly and independently at random
3: Let $G' = (V, E')$ where $E' = \{(u, v) \in E : c(v) = c(u) + 1\}$
4: **if** `kpathDAG`(G', k) **then return true**
5: **return false**

- If true is returned, G' has a k-path so G as well
- If G has a k-path, look at one iteration:
  - $\Pr_c[G' \text{has k}-\text{path}\,] \geq \Pr_c[\forall i \in \{1, \dots, k\}: c(p_i) = i] = \left(\frac{1}{k}\right)^k$
  - So prob. we won't find a path in the whole loop is at most $\left(1 - \frac{1}{k^k}\right)^{k^k} \leq 1/e$, if k-path exists, return true with prob $1 - \frac{1}{e}$

$1 + x \leq e^x$ for $x = -1/k^k$

- $O^*(k^k)$ time algorithm with constant one-sided err.

Example 4

- Determine whether a given k-CNF formula on n vars is satisfiable in $O^*\left(\left(\frac{2k}{k+1}\right)^n\right)$ time, for constant k.
  - for $k = 100$, this is about $O^*(1.9802^n)$
- relies on local search subroutine
- Given $x, y \in \{0,1\}^n, H(x,y)$ is *Hamming distance* (i.e., nr. of coordinates where x and y differ)

Example 4

Algorithm `localSearch`(φ, x, d) φ is a k-CNF-formula on n variables, $x \in \{0,1\}^n$, $d \in \mathbb{N}_{\geq 0}$.

Output: Whether there exists $y \in \{0,1\}^n$ that satisfies φ and $H(x,y) \leq d$

1: **if** $d = 0$ and $\varphi(x) =$ **false then return false**
2: **if** $\exists$ clause C_j not satisfied by x **then**
3: For $i = 1, \ldots, \ell = |C_j|$ let $z^i \in \{0,1\}^n$ be the assignment obtained from x by flipping the
4: variable in the i'th literal of C_j
5: **return** $\bigvee_{i=1}^{\ell}$ `localSearch`$(\varphi, z^i, d-1)$
6: **else**
7: **return true**

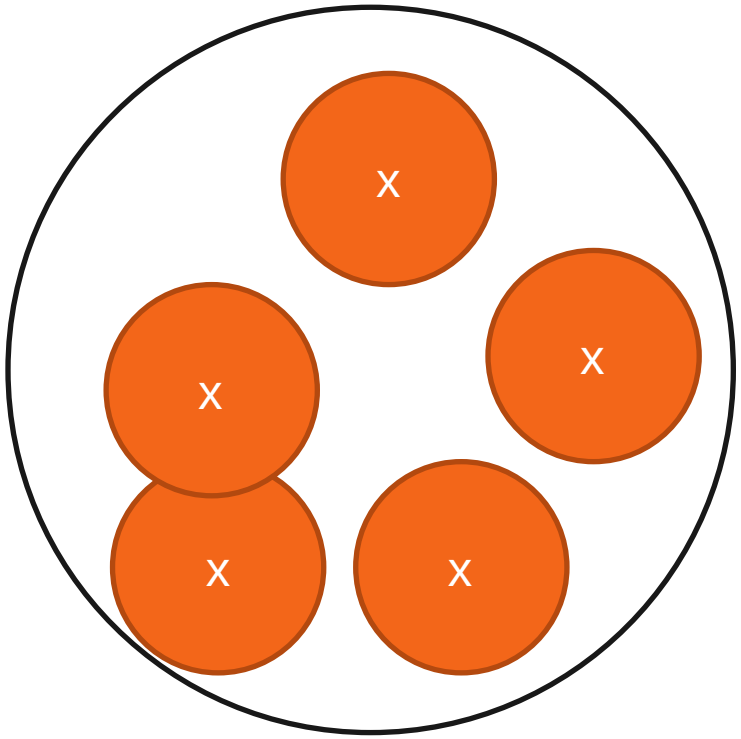
- Run time: recursion tree of depth $\leq d$, $\ell \leq k$ fanout
  - Polynomial time per recursive call so $O^*(k^d)$ time.
- Only considers assignments of HD $\leq d$:
  - decreases d if it flips a coordinate
  - if true is returned, found a solution of HD $\leq d$
- By induction on d: if sol y of HD $\leq d$, it returns true
  - True for $d = 0$, since condition on L2 fails
  - For $d > 0$, x, y must differ on variable occurring in C_j
  - If variable occurs in i'th literal $H(z^i, y) \leq d - 1$

Example 4

Algorithm `kSAT`(φ) — φ is a k-CNF-formula on n variables

Output: Whether $x \in \{0,1\}^n$ is satisfiable, with constant one-sided error probability.

1: $d = n/(k+1)$ — Assume d is integer (otherwise, add at most k dummy variables)
2: **for** $i = 1 \dots \lceil 2^n / \binom{n}{d} \rceil$ **do**
3: Pick $x \in \{0,1\}^n$ uniformly at random
4: **if** `localSearch`(φ, x, d) **then return true**
5: **return false**



Example 4

Algorithm `kSAT`(φ) — φ is a k-CNF-formula on n variables

Output: Whether $x \in \{0,1\}^n$ is satisfiable, with constant one-sided error probability.

1: $d = n/(k+1)$ — Assume d is integer (otherwise, add at most k dummy variables)
2: **for** $i = 1 \dots \lceil 2^n / \binom{n}{d} \rceil$ **do**
3: Pick $x \in \{0,1\}^n$ uniformly at random
4: **if** `localSearch`(φ, x, d) **then return true**
5: **return false**

- If returns true it is correct since so is **localSearch**
- If sol y exists, prob true is returned in one iteration is

$$\Pr_x[H(x,y) \le d] = \frac{\sum_{i=0}^{d} \binom{n}{i}}{2^n} \ge \frac{\binom{n}{d}}{2^n}$$, so in some iteration $\ge 1 - \frac{1}{e}$

$$\binom{n}{d} = \frac{n!}{d!(n-d)!} \ge \frac{(n/e)^n}{3n(d/e)^d 3n((n-d)/e)^{n-d}} = \frac{n^n}{d^d (n-d)^{n-d} 9n^2}$$

- And the running time boils down to:

$$k^d \frac{2^n}{\binom{n}{d}} = \frac{2^n (kd)^d (n-d)^{n-d} 9n^2}{n^n} = \frac{2^n \left(n\frac{k}{k+1}\right)^d \left(n\frac{k}{k+1}\right)^{n-d} 9n^2}{n^n} = O^*\left(\left(\frac{2k}{k+1}\right)^n\right)$$

Conditional expecations and derandomization

- Consider the problem MAX-3-SAT: given a 3-CNF formula ϕ, find an assignment satsifying the maximum number of clauses
- A random assignment will in expectation satisfy $7/8$ clauses

Conditional expecations and derandomization

- Consider the problem MAX-3-SAT: given a 3-CNF formula ϕ, find an assignment satsifying the maximum number of clauses
- A random assignment will in expectation satisfy $7/8$ clauses
- Can deterministically find such a solution, by computing
  - $E[\#\text{of clauses satisfied} \mid x_1 = true]$
  - $E[\#\text{of clauses satisfied} \mid x_1 = false]$

More on randomized algo's

- Randomized rounding
  - Important approach to turn LP-solutions into ILP solutions for obtaining good apx algo's
- Many more cool examples

- LNMB course `Randomized Algorithms'.

Algorithms & Complexity

- We saw:
  - NP, co-NP, NP-completeness, undecidability
  - Approximation algorithms
  - Exponential time (and FPT) algorithms
  - Treewidth
  - Randomized algorithms
- What did we skip?

Algorithms

- Many `basic' algorithms for classic problems on
  - Sequences: Sorting, Pattern Matching,…
  - Graphs: Shortest Path, Matching, Max-Flow, ..
  - Algebra: Matrix multiplication, FFT, number-theoretic
  - Computational Geometry: Convex hull, closest pair
- We've seen many of the important techniques
  - Greedy, DP, Divide&Conquer, LP-rounding, randomization

- Online, quantum, parallel, game-theoretic algorithms
- **Important one we skipped:** Heuristic algorithms

Complexity

- Models of Computation
  - Turing machine, circuits, branching programs…
- Diagonalization
- Polynomial hierarchy, NP-intermediate problems
- Proof systems
  - PSPACE = IP
  - PCP-theorem
- Derandomization
  - L=NL, Adleman's theorem
- Communication Complexity
- `Fine-grained complexity'
  - (S)ETH, W-hardness, conditional hardness results

Further reading