CS 341: Algorithms

Lecture 21: NP-completeness

Éric Schost

based on lecture notes by many other CS341 instructors

David R. Cheriton School of Computer Science, University of Waterloo

Fall 2024

Aside: the rock's statement

NP $\cap$ co-NP

These are the problems where we can certify **both** yes and no instances efficiently.

MaxFlowDecision:

- **input:** integer-weighted graph G, source s, sink t, K
- **output:** is there a flow of value **at least** K?

MinCutDecision:

- **input:** integer-weighted graph G, source s, sink t, K
- **output:** is there a cut of capacity **at most** K?

Claim: max flow = min cut $\implies$ both problems in **NP $\cap$ co-NP**

- MaxFlowDecision is **NP**

  certificate that there is flow of value at least K: a flow of value at least K
- MaxFlowDecision is **co-NP**

  certificate that there is no flow of value at least K: a cut of capacity at most $K - 1$

$\mathbf{NP} \cap \mathbf{co\text{-}NP} = \mathbf{P}$?

Flow and cuts

- in **P**! (Edmonds-Karp)

Linear programming

- optimize a linear function while satisfying linear inequalities
- also have a max (something) = min (something else), so **NP** $\cap$ **co-NP**
- in **P**!! (ellipsoid)

Primality

- certificates for non primes (easy) and for primes (not so easy), so **NP** $\cap$ **co-NP**
- in **P**!!! (AKS)

Factoring

- HasFactor is in **NP** $\cap$ **co-NP**
- ?

NP-completeness

NP-complete problems

Definition

A decision problem $\textsc{Prob}$ is NP-complete if

- $\textsc{Prob}$ is in **NP**
- for any $\textsc{Prob}$' in **NP**, $\textsc{Prob}' \leq_P \textsc{Prob}$

polynomial time for $\textsc{Prob}$ would give **P=NP** (so polynomial time for SAT, $\textsc{IndependentSet}$, $\textsc{VertexCover}$, $\textsc{Clique}$, …)

Remark: NP-hard problems = the second part of the definition

- decision problem $\textsc{Prob}$ such that for any $\textsc{Prob}$' in **NP**, $\textsc{Prob}' \leq_P \textsc{Prob}$

Exercise

find an NP-hard problem that is provably not in **NP**

The Cook-Levin theorem

Claim

CircuitSAT is NP-complete

Remark 1: we already know it is in NP

Remark 2:

- we proved CircuitSAT $\leq_P$ 3SAT
- so 3SAT is NP-complete (it is in NP)
- we won't use CircuitSAT too much after that

World map

$$A \leftarrow B \equiv A \le_p B$$

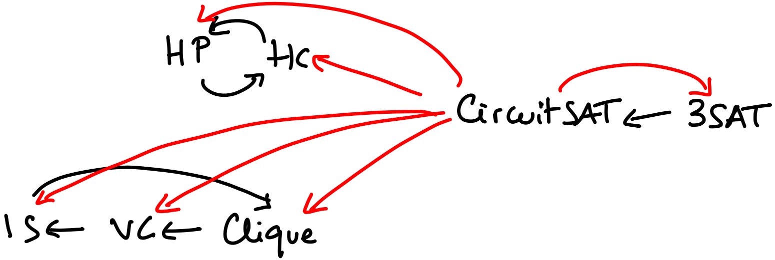
$$\begin{tikzcd} HP \arrow[r, bend right] & HC \arrow[l, bend right] \end{tikzcd}$$

$$\text{CircuitSAT} \leftarrow \text{3SAT}$$

$$\begin{tikzcd} IS \arrow[rr, bend left] & VC \arrow[l] & \text{Clique} \arrow[l] \end{tikzcd}$$

World map

$$A \leftarrow B \equiv A \le_p B$$



Sketch of proof

take $\textsc{Prob}$ in **NP** (so there is a certifying algorithm B), want $\textsc{Prob} \leq \textsc{CircuitSAT}$
$\rightsquigarrow$ must transform an instance x of $\textsc{Prob}$ into a circuit

Idea

- given x, verification algorithm $B(x, y)$ can be turned into a circuit with y as input
- we call $\textsc{CircuitSAT}$ to find y

Example

- **problem** $\textsc{Prob}$: $\textsc{IndependentSet}$
- **instance** x: complete graph with 3 vertices (aka a triangle), $K = 2$
- **certificate** y: 3 bits y_1, y_2, y_3 (yes/no for each vertex)
- circuit for $B(x, y)$ computes the "formula"

$$(y_1 + y_2 + y_3 \geq 2) \;\wedge\; \overline{y_1 \wedge y_2} \;\wedge\; \overline{y_1 \wedge y_3} \;\wedge\; \overline{y_2 \wedge y_3}$$

Sketch of proof

Turing machines

- RAM model too complicated, use Turing machines instead
- have a **pointer** to memory and a **state** ($\simeq$ line in the source code)
- each step, pointer can write a new symbol, move left / right and change state

From machine to circuit

- on input bit vector x of size n, introduce a large table T of size $n^k \times n^k$ (k=exponent in runtime of B)
- cell (i, j) records contents of jth memory cell at time i, whether the pointer was there, and the machine state
- cells at row $i+1$ are given by a boolean circuit taking row i as input (big, but polynomial size)
- output of the circuit = output of the Turing machine at the last time step

Some NP-complete problems

- CircuitSAT
- 3SAT, SAT
- independent set, vertex cover, clique
- (directed) Hamiltonian cycle, Hamiltonian path
- traveling salesman
- subset sum, 0/1 knapsack

(2SAT is polynomial time)

IndependentSet, VertexCover, Clique are NP-complete

We already know they are in **NP**

Claim

3SAT $\leq_P$ IndependentSet

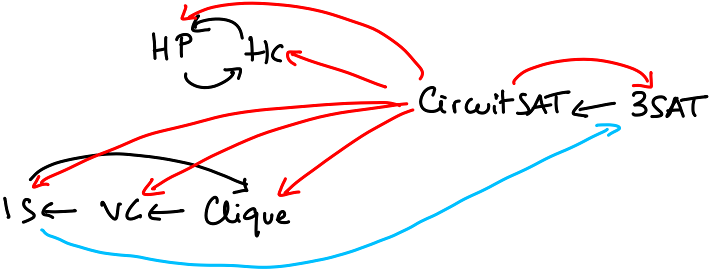
Reduction (transform an instance F of 3SAT with s clauses into an independent set instance)

- build a graph G with one vertex per literal
- connect all literals in any given clause
- connect all pairs x_i, $\overline{x_i}$

Remark: reduction takes polynomial time

World map

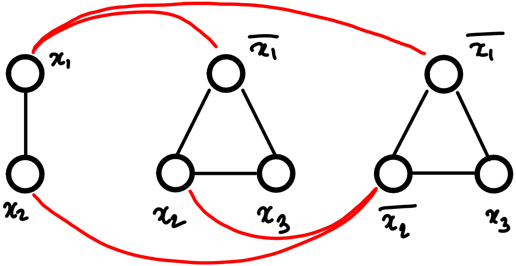
$$A \leftarrow B \equiv A \leq_p B$$



Example

A 3CNF formula with $s = 3$

$$F = (x_1 \vee x_2) \ \wedge \ (x_2 \vee x_3 \vee \overline{x_1}) \ \wedge \ (\overline{x_1} \vee x_3 \vee \overline{x_2}).$$



Proof

Claim

F satisfiable iff G has an independent set of size at least $\boldsymbol{s}$

If F satisfiable

- pick **one** true literal in each clause as set S, so $|S| = s$
- no edge within clauses
- no edge $\{x_i, \overline{x_i}\}$ either

If G has an independent set S of size at least s

- S has (exactly) one vertex per clause
- make these literals true (for any variable we did not assign, arbitrary choice)
- no conflict, because any $x_i, \overline{x_i}$ cannot be both in S

DirectedHamiltonianCycle, HamiltonianCycle, HamiltonianPath are NP-complete

Definition: DirectedHamiltonianCycle

- **input:** directed graph G
- **output:** does G have a directed cycle that visits each vertex once?
- **NP**

Claim

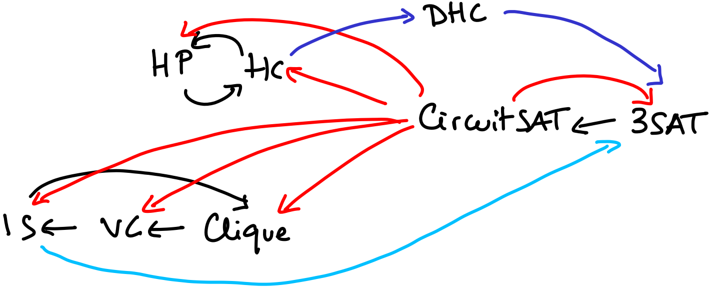
3SAT $\leq_P$ DirectedHamiltonianCycle $\leq_P$ HamiltonianCycle

start with 3SAT $\leq_P$ DirectedHamiltonianCycle, so we are given a formula in 3CNF

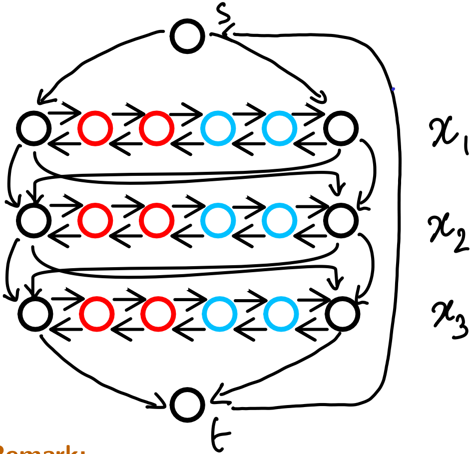
(**Remark:** almost the same construction works for DirectedHamiltonianPath)

World map

$$A \leftarrow B \equiv A \le_p B$$



Starting the construction



Rules

- source s, sink t
- **one row** of vertices per variable x_i
- on row i, **2** outside vertices (black) and **2** vertices $\boldsymbol{v_{i,j,1}}, \boldsymbol{v_{i,j,2}}$ per clause C_j
- example with $(\boldsymbol{x_1} \vee \boldsymbol{x_2} \vee \overline{\boldsymbol{x_3}}) \wedge (\overline{\boldsymbol{x_1}} \vee \boldsymbol{x_2})$

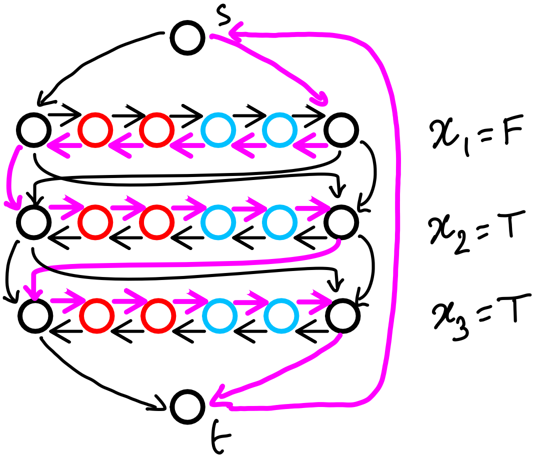
  (we're not done yet)

Remark:

- enough to consider only the x_i's that show up in our formula
- so we can assume $n \in O(\ell)$ (ℓ = number of clauses)
- size of the graph and construction time polynomial in $n\ell$

Hamiltonian cycles = variable assignments

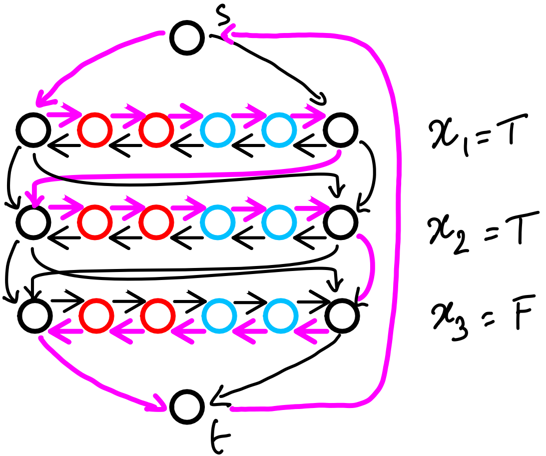
convention: T = left to right, F = right to left



so far, 2^n Hamiltonian cycles

Hamiltonian cycles = variable assignments

convention: T = left to right, F = right to left

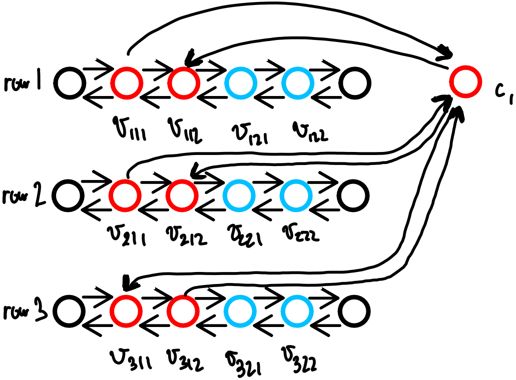


so far, 2^n Hamiltonian cycles

Using the clauses to finish the graph

For any clause C_j

- add a new vertex, also called c_j
- for any literal $\boldsymbol{x_i}$ in C_j, add edges $(\boldsymbol{v_{i,j,1}}, \boldsymbol{c_j})$ and $(\boldsymbol{c_j}, \boldsymbol{v_{i,j,2}})$
- for any literal $\overline{\boldsymbol{x_i}}$ in C_j, add edges $(\boldsymbol{c_j}, \boldsymbol{v_{i,j,1}})$ and $(\boldsymbol{v_{i,j,2}}, \boldsymbol{c_j})$



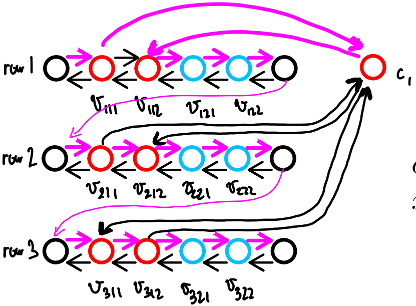
$$\boldsymbol{c_1 = (x_1 \vee x_2 \vee \overline{x_3})}$$

3SAT $\leq_P$ DirectedHamiltonianCycle

Claim

if formula sastisfiable, there is a directed Hamiltonian cycle in G

- variable assignment $\implies$ direction (LtoR for **true** or RtoL for **false**) on each row
- choose **one** literal x or $\bar{x}$ set to true per clause C_j
- detour to visit c_j when we go through the corresponding row (if x **true** we go LtoR, if x **false** we go RtoL)



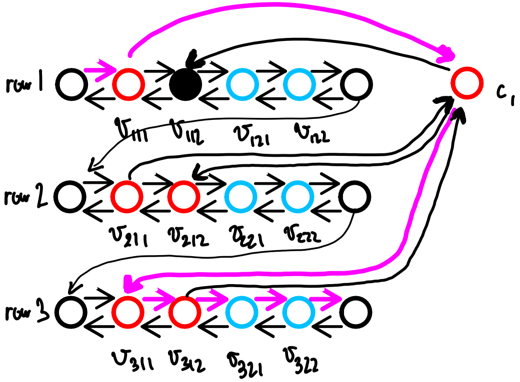
$c_1 = \boldsymbol{x_1} \vee x_2 \vee \overline{x_3}$
$x_1 = x_2 = x_3 = T$

3SAT $\leq_P$ DirectedHamiltonianCycle

Claim

if directed Hamiltonian cycle in G, formula sastisfiable

Key Observation: if cycle goes from $v_{i,j,1}$ to c_j, must come back to $v_{i,j,2}$ (else, cannot put $v_{i,j,2}$ on the cycle), same with $v_{i,j,2} \to c_j \to v_{i,j,1}$



3SAT $\leq_P$ DirectedHamiltonianCycle

Claim

if directed Hamiltonian cycle in G, formula sastisfiable

Key Observation: if cycle goes from $v_{i,j,1}$ to c_j, must come back to $v_{i,j,2}$ (else, cannot put $v_{i,j,2}$ on the cycle), same with $v_{i,j,2} \to c_j \to v_{i,j,1}$

Consequences

- if we remove all vertices c_j from our cycle, we get a cycle on the clause-free graph
- so each row is visited LtoR or RtoL
- gives an assignment for $x_1, \ldots, x_n$
- by design, it satisfies all clauses

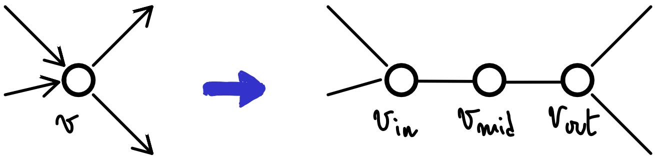
DirectedHamiltonianCycle $\leq_P$ HamiltonianCycle

Reduction

- **given:** a directed graph G
- **build:** an undirected graph G'
- **ensure:** directed Hamiltonian cycle in G $\iff$ Hamiltonian cycle in G'

Gadget:

- replace each vertex v by $v_{\text{in}}, v_{\text{mid}}, v_{\text{out}}$
- make all edges undirected



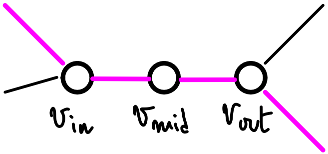
DirectedHamiltonianCycle $\leq_P$ HamiltonianCycle

Claim

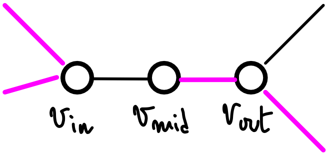
directed Hamiltonian cycle in G $\iff$ Hamiltonian cycle in G'

Proof

- if directed Hamiltonian cycle in G, Hamiltonian cycle in G' (follow the cycle)
- suppose Hamiltonian cycle in G'. Can only have



but not



(v_{mid} would be isolated)

gives a directed Hamiltonian cycle in G