BU CS 332 – Theory of Computation

Lecture 22:

- NP completeness
- Clique, subset sum is NP-c

Reading:

Sipser Ch 7.3-7.5

Ran Canetti

December 1, 2020

Polynomial-time reducibility

Definition:

A function $f: \Sigma^* \rightarrow \Sigma^*$ is polynomial-time computable if there is a polynomial-time TM M which, given as input any $w \in \Sigma^*$, halts with only $f(w)$ on its tape.

Definition:

Language A is polynomial-time mapping reducible to language B, written

$$A \leq_{\mathrm{p}} B$$

if there is a polynomial-time computable function $f: \Sigma^* \rightarrow \Sigma^*$ such that for all strings $w \in \Sigma^*$, we have $w \in A \Leftrightarrow f(w) \in B$

Implications of poly-time reducibility

Theorem: If $A \leq_{\mathrm{p}} B$ and $B \in P$, then $A \in P$.

Theorem: If A $\leq_{\mathrm{P}}$ B and B$\leq_{\mathrm{P}}$ C, then A $\leq_{\mathrm{P}}$ C.

NP-complete languages: The hardest in NP

A language B is **NP-complete** if

1. $B \in NP$
2. B is **NP-hard**, i.e., $\forall A \in NP, \;\; A \leq_p B$

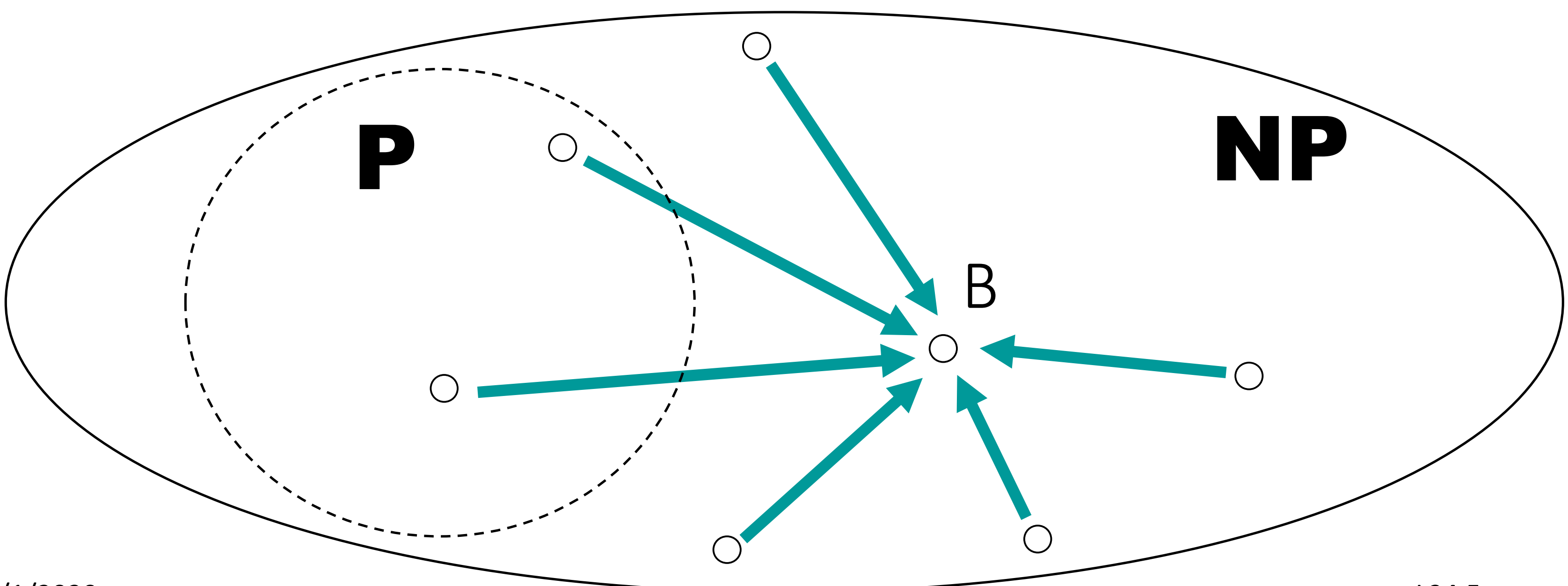
(every language in NP is poly-time reducible to B.)

NP-complete languages: The hardest in NP

A language B is **NP-complete** if

1. $B \in NP$
2. B is **NP-hard**, i.e., $\forall A \in NP,\ \ A \leq_p B$

(every language in NP is poly-time reducible to B.)

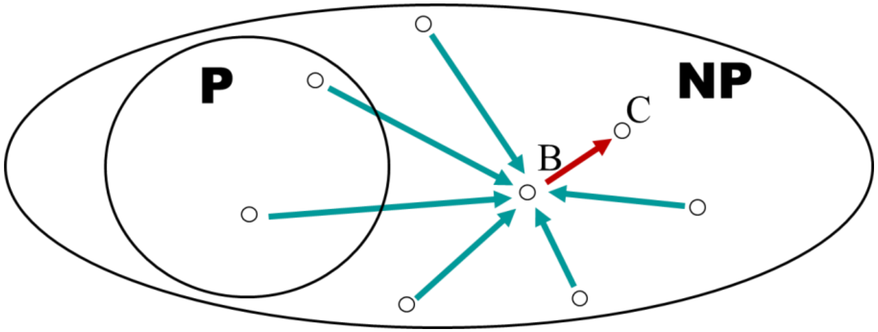


Implication of poly-time reductions

Theorem. If

- B is **NP**-complete,
- C$\in \mathbf{NP}$ and
- B$\leq_p$C

then C is **NP**-complete.



Theorem. If B is **NP**-complete and B$\in \mathbf{P}$ then

$$\mathbf{P} = \mathbf{NP}.$$

(So, if B is **NP**-complete and $\mathbf{P} \neq \mathbf{NP}$

then there is no poly-time algorithm for B.)

An NP-Complete problem

$$T_{NTM} = \{(N, x, 1^t)\text{: } NTM\ N\ accepts\ x\ within\ \ t\ steps\}$$

T_{NTM} *Is NP-complete:*

- $T_{NTM} \in NP$
- For all $L \in NP$, $L \leq_p T_{NTM}$ *:*

Cook-Levin Theorem

Theorem: SAT (Boolean satisfiability) is NP-complete

Proof: Already know $SAT \in \mathrm{P}$. Need to show every problem in NP reduces to SAT

Stephen A. Cook (1971)

Leonid Levin (1973)

Proof of Cook-Levin Theorem

- Proof idea
  - For each language A in NP, with a given input x for A, produce a Boolean formula ϕ that simulates the verification machine V for A on input x,w.

➔ ϕ is satisfiable if and only if there exists w such that V(x,w)=1.

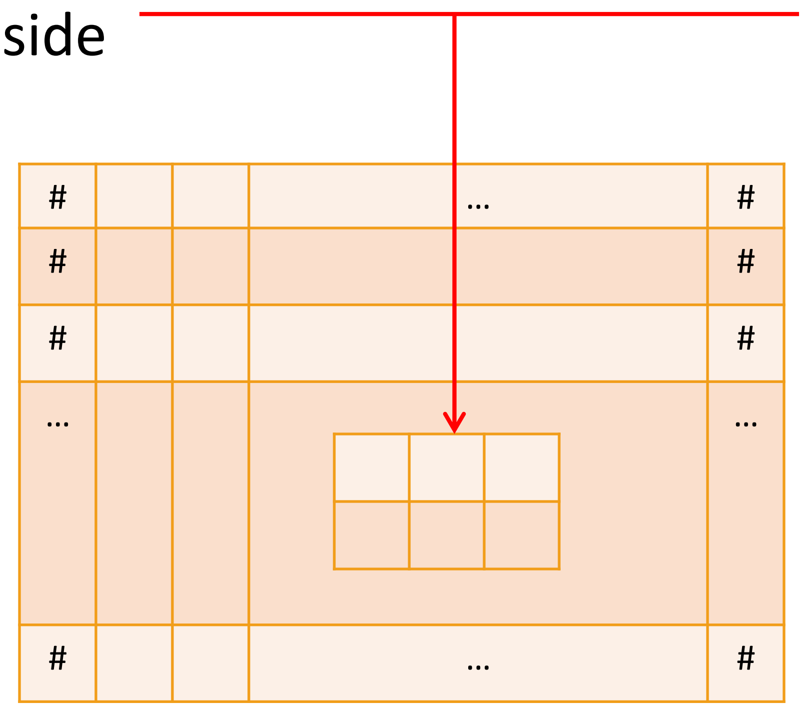
Proof of Cook-Levin Theorem

- Proof idea:
  - -The tableau of the computation of V(x,w) is polysize
  - Have a variable describing each cell in the tableau
  - Can verify that the tableau is a legal accepting computation by checking only local conditions (windows of 2x3 cells)

    - all checks are constant side

    - poly-many checks

  → Can combine the checks to a poly-size CNF formula



New NP-complete problems from old

Lemma: If $A \leq_{\mathrm{p}} B$ and $B \leq_{\mathrm{p}} C$, then $A \leq_{\mathrm{p}} C$

(poly-time reducibility is <u>transitive</u>)

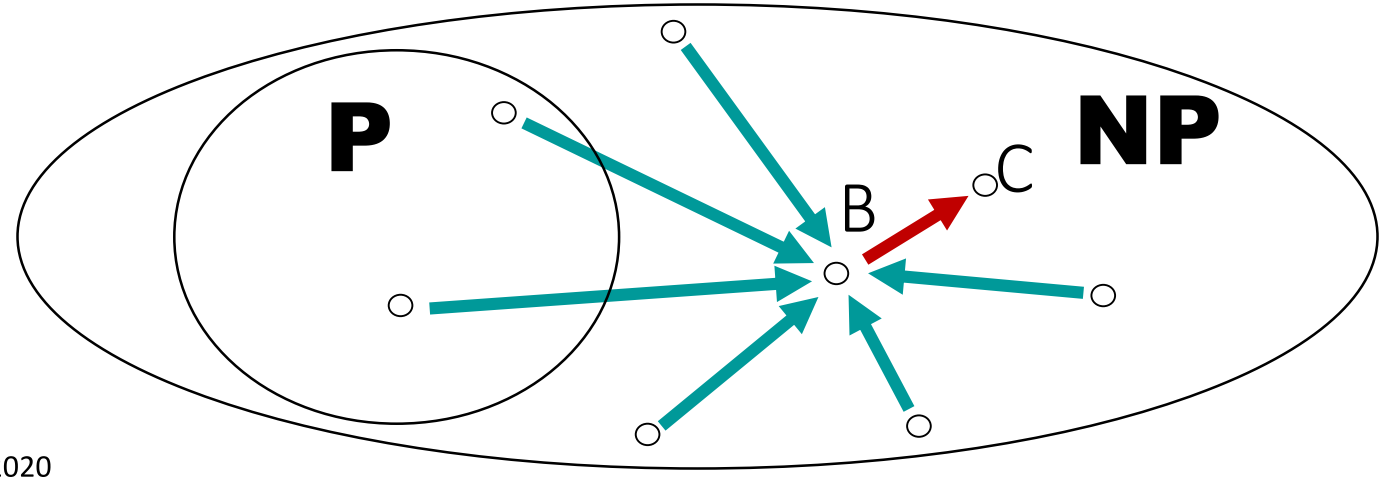
Theorem: If $C \in \mathrm{NP}$ and $B \leq_{\mathrm{p}} C$ for some NP-complete language B, then C is also NP-complete

Implication of poly-time reductions

Theorem. If

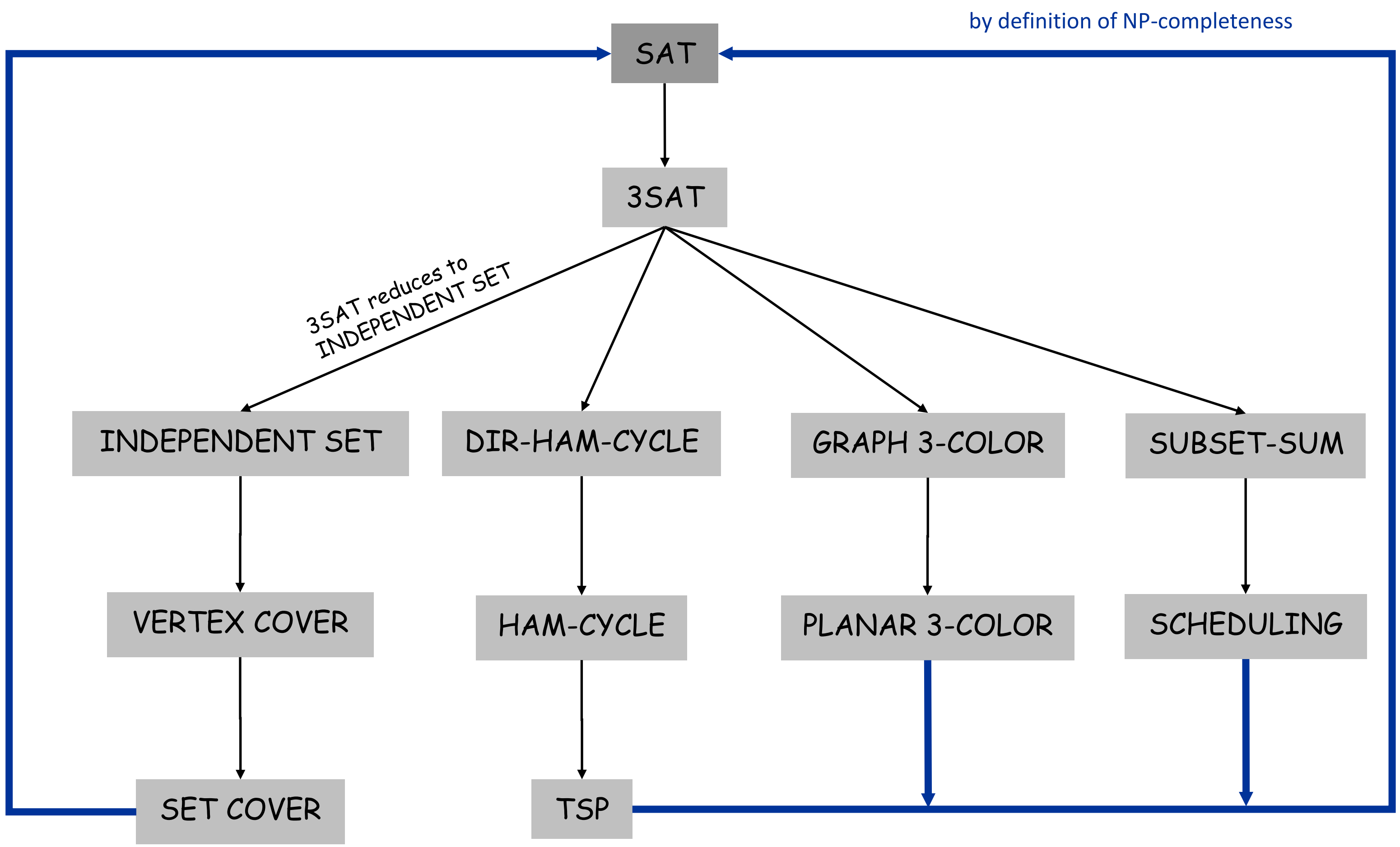
- B is **NP**-complete,
- C$\in$ $\mathbf{NP}$ and
- B$\leq_p$C

then C is **NP**-complete.



New NP-complete problems from old

All problems below are NP-complete and hence poly-time reduce to one another!



$3SAT$ (3-CNF Satisfiability)

Definition(s):

- A literal either a variable of its negation x_5 , $\overline{x_7}$
- A clause is a disjunction (OR) of literals Ex. $x_5 \vee \overline{x_7} \vee x_2$
- A 3-CNF is a conjunction (AND) of clauses where each clause contains exactly 3 literals

Ex. $C_1 \wedge C_2 \wedge \ldots \wedge C_m =$

$$(x_5 \vee \overline{x_7} \vee x_2) \wedge (\overline{x_3} \vee x_4 \vee x_1) \wedge \cdots \wedge (x_1 \vee x_1 \vee x_1)$$

$$3SAT = \{\langle\varphi\rangle | \varphi \text{ is a satisfiable } 3 - \text{CNF}\}$$

$3SAT$ is NP-complete

Theorem: $3SAT$ is NP-complete

Proof idea: 1) $3SAT$ is in NP (why?)

2) Show that $SAT \leq_{\mathrm{p}} 3SAT$

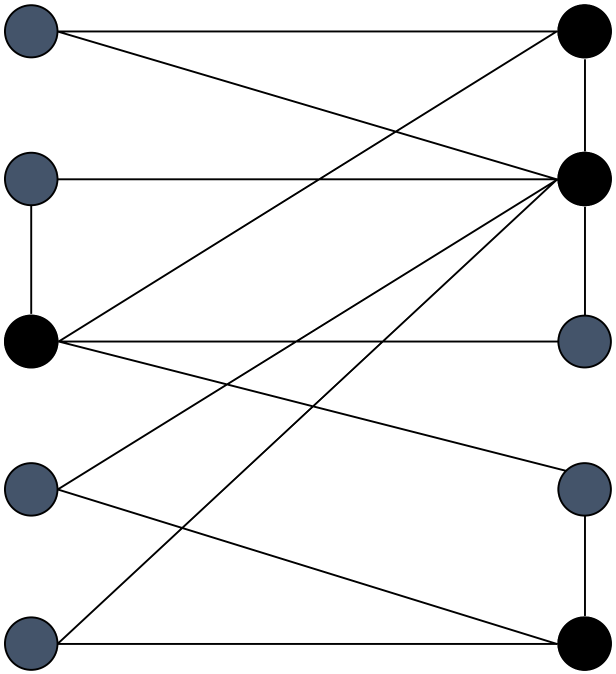
Idea of reduction: Given a poly-time algorithm converting an arbitrary formula φ into a 3CNF ψ such that φ is satisfiable iff ψ is satisfiable

Independent Set

An **independent set** in an undirected graph G is a set of vertices such that no edge has both its endpoints in the set.

$$INDEPENDENT-SET$$
$$= \{\langle G, k\rangle | G \text{ is an undirected graph containing an independent set with} \geq k \text{ vertices}\}$$

- Is there an independent set of size ≥ 6?
- Is there an independent set of size ≥ 7?



Independent Set is NP-complete

1) $INDEPENDENT - SET \in \mathrm{NP}$
2) Reduce $3SAT \leq_{\mathrm{p}} INDEPENDENT - SET$

Proof. "On input $\langle \varphi \rangle$, where φ is a 3CNF formula,

1. Construct graph G from φ
   - G contains 3 vertices for each clause, one for each literal.
   - Connect 3 literals in a clause in a triangle.
   - Connect literal to each of its negations.
2. Output $\langle G, k \rangle$, where k is the number of clauses in φ."

Example of the reduction

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

Clique

An **clique** in an undirected graph G is a set of vertices such that every pair of vertices in the set are connectged via an edge.

Theorem: $INDSET \leq_{\mathrm{p}} CLIQUE$

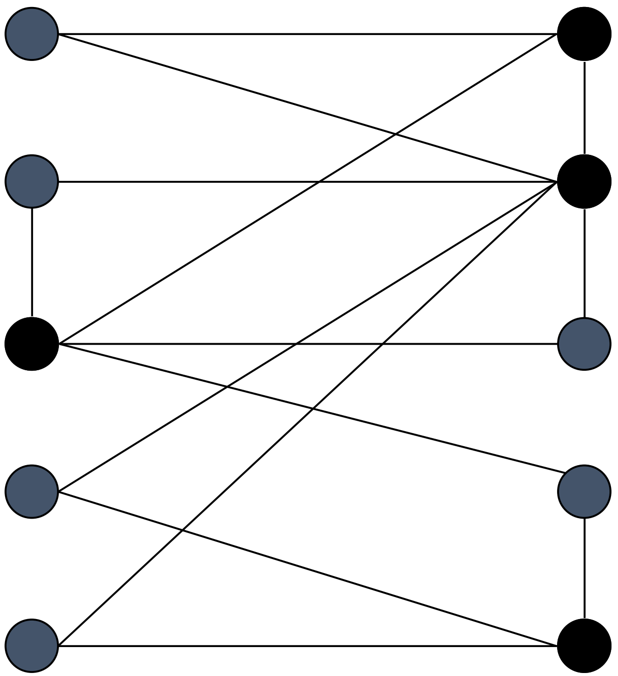
Vertex Cover

Given an undirected graph G, a **vertex cover** in G is a subset of nodes, which includes at ***least*** one endpoint of every edge.

VERTEX COVER$= \{\langle G, k\rangle \mid G$ is an undirected graph which has a vertex cover with k nodes$\}$

- Is there vertex cover of size ≤ 4?

- Is there a vertex cover of size ≤ 3?



Independent Set and Vertex Cover

Claim. S is an independent set iff V – S is a vertex cover.

- $\Rightarrow$
  - Let S be any independent set.
  - Consider an arbitrary edge (u, v).
  - S is independent $\Rightarrow$ u $\notin$ S or v $\notin$ S $\Rightarrow$ u $\in$ V – S or v $\in$ V – S.
  - Thus, V – S covers (u, v).
- $\Leftarrow$
  - Let V – S be any vertex cover.
  - Consider two nodes u $\in$ S and v $\in$ S.
  - Then (u, v) $\notin$ E since V – S is a vertex cover.
  - Thus, no two nodes in S are joined by an edge $\Rightarrow$ S independent set. ▪

INDEPENDENT SET reduces to VERTEX COVER

Theorem. INDEPENDENT-SET $\leq_p$ VERTEX-COVER.

Proof. "On input $\langle G, k \rangle$, where G is an undirected graph and k is an integer,

1. Output $\langle G, n - k \rangle$, where n is the number of nodes in G."

Correctness:

- G has an independent set of size k iff it has a vertex cover of size $n - k$.
- Reduction runs in linear time.

Set Cover

Given a set U, called a *universe*, and a collection of its subsets $S_1, S_2, \ldots, S_m$, a **set cover** of U is a subcollection of subsets whose union is U.

- SET COVER=$\{\langle U, S_1, S_2, \ldots, S_m; k\rangle$ | U has a set cover of size $k\}$

U = { 1, 2, 3, 4, 5, 6, 7 }

k = 2

S_1 = {3, 7} S_4 = {2, 4}

S_2 = {3, 4, 5, 6} S_5 = {5}

S_3 = {1} S_6 = {1, 2, 6, 7}

- Sample application.
  - m available pieces of software.
  - Set U of n capabilities that we would like our system to have.
  - The ith piece of software provides the set $S_i \subseteq U$ of capabilities.
  - Goal: achieve all n capabilities using fewest pieces of software.

VERTEX COVER reduces to SET COVER

Theorem. VERTEX-COVER $\leq_P$ SET-COVER.

Proof. "On input $\langle G, k \rangle$, where $G = (V, E)$ is an undirected graph and k is an integer,

1. Output $\langle U, S_1, S_2, \ldots, S_m; k \rangle$, where U=E and for each $v \in V$,

$$S_v = \{e \in E \mid e \text{ is incident to } v\}"$$

Correctness:

- G has a vertex cover of size k iff U has a set cover of size k.
- Reduction runs in linear time.

Proof of correctness for reduction

Let k = # clauses and l = # literals in φ

Claim: φ is satisfiable iff G has an ind. set of size k

$\Longrightarrow$ Given a satisfying assignment, select one literal from each triangle. This is an ind. set of size k

$\Longleftarrow$ Let S be an ind. set of size k

- S must contain exactly one vertex in each triangle
- Set these literals to true, and set all other variables in an arbitrary way
- Truth assignment is consistent and all clauses satisfied

Runtime: $O(k + l^2)$ which is polynomial in input size