Cpts 515. 11/30/2020.

Thm. #DNF is #P-complete.

Proof. (last time) Every Boolean formula can be efficiently (in poly-time) written into CNF, but not necc. into DNF.

Observation:

① F is CNF then $\bar{F}$ is DNF

② $\#F = 2^n - \#\bar{F}$, n is the # of vars in F.

$\#F$: # of sat. assignments in F

Hence, #SAT can be computed through #DNF:

Given: a DNF F

Question: What is the # of sat. assignments in F?

Notice that, the decision problem for #DNF:

Given: a DNF F

Question: F is sat.?

$(\#F > 0 ?)$

is solvable in poly-time (is in P). Looking at an example:

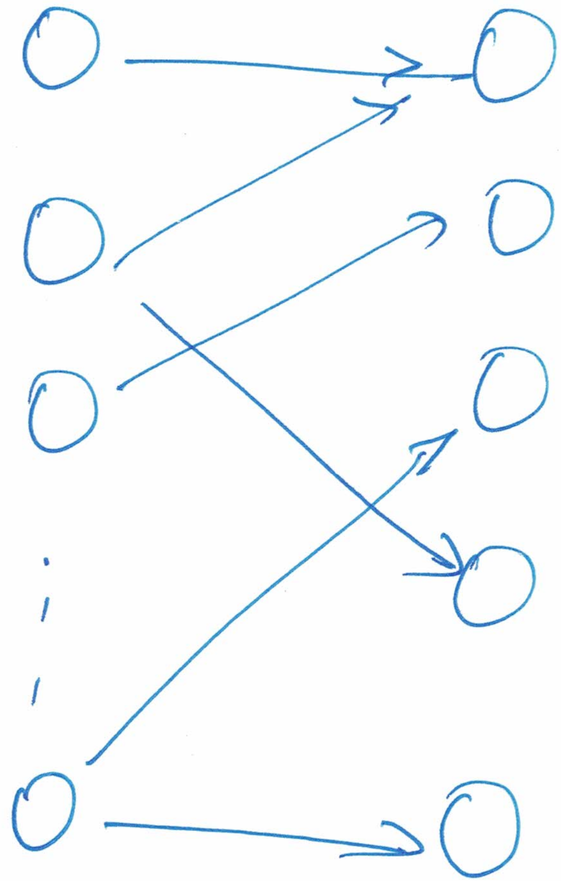
$$(\bar{a} \wedge \bar{b} \wedge c) \vee (a \wedge b)$$

is sat.

But counting the # of sat. assignments in DNF is hard, since each $(\_ \wedge \_ \wedge \_ \wedge \ldots)$ may not contain all var's.

A classic example of #P-completeness.

Consider bi-partite graph matching:

n nodes  nodes.

Given: a bipartite graph G, we know that finding a max-matching in G of size n can ~~so~~ be solved in poly-time. But, how many such matchings?

The counting problem is #P-complete. why?

Define $A[i,j] = \begin{cases} 1 & \text{if } a_i \to b_j \text{ is an edge in } G \\ 0 & \text{o.w.} \end{cases}$

Let $\Pi(n)$ be the set of all permutations of $\{1, \ldots, n\}$.

Define $\mathrm{perm}(A) = \sum_{g \in \Pi(n)} A[1, g(1)] \cdot \ldots \cdot A[n, g(n)]$.

Each permutation can be understood as a 1-1 mapping from $\{1, \ldots, n\}$ to itself.

Hence, $\mathrm{perm}(A)$ = the # of matchings mentioned earlier in G;

However, computing $\mathrm{perm}(A)$ is #P-complete (when A is 0-1 matrix)

Approximation on NP-complete problems.

taken from:

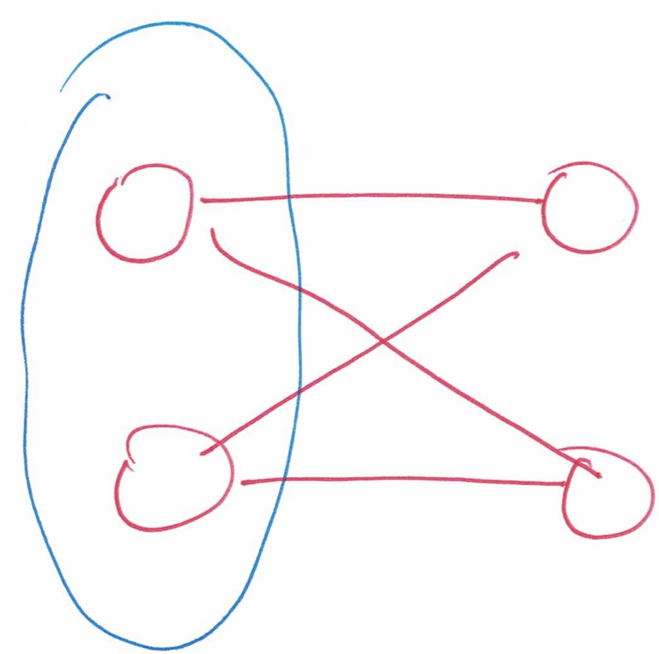
Design of Approx. Algs

by David Williamson & David Shmoys.

(2011, Cambridge Press).

Motivation: Given an NP-complete problem, we want to find a poly-time alg that identifies a "c-times worse" solution for some constant c.

Approx. of VC. // Vertex-Cover.



Given: an undirected graph $G = (V, E)$

Goal: Find a VC with minimal size.

That is, find a minimal $C^* \subseteq V$ s.t. each edge in E will touch at least a node in C^*.

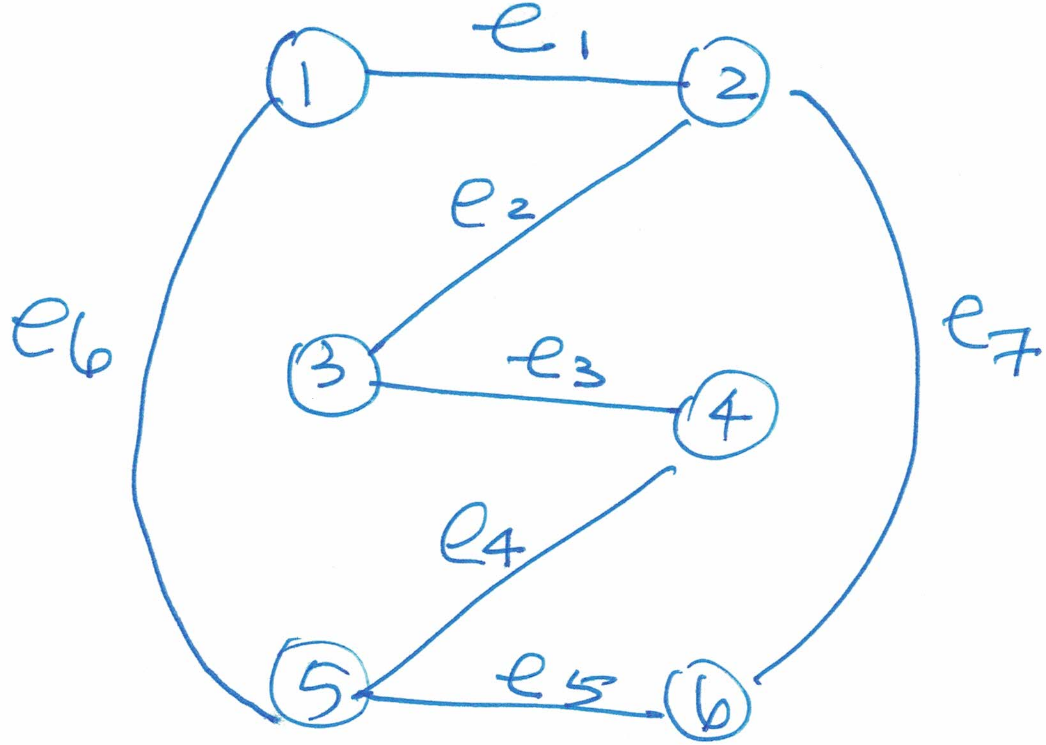
The decision version of the above VC is NP-complete

We have the following efficient alg that finds a vertex cover C with $|C| \leq 2 \cdot |C^*|$ (at most 2-times worse). This is called 2-approximation.

Alg: repeatedly: take an edge $e = (u, v)$ and add both u and v to the current cover C and drop all edges that touch either u or v from G

stop when there is no edge left.

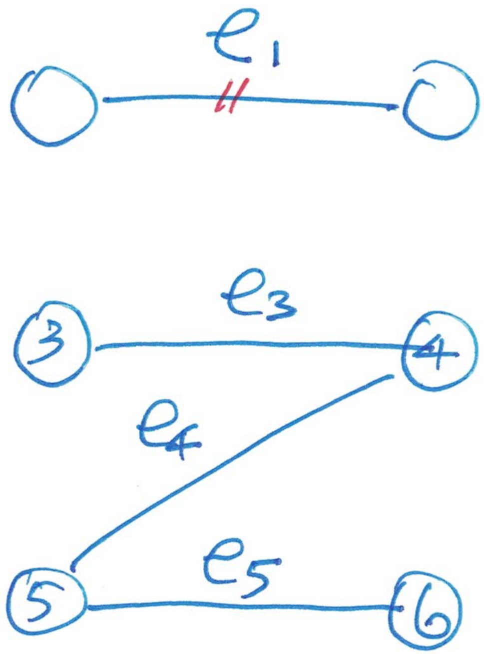
Example:



take e_1.

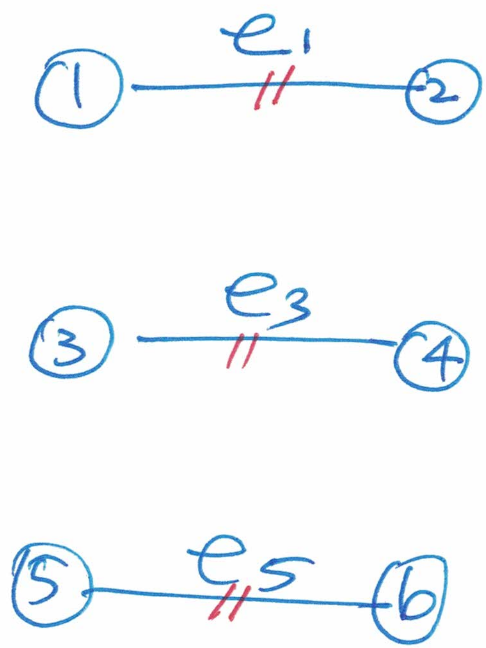
$\Longrightarrow$

and drop all edges that touch ① or ②.



take e_1

Now, take e_5 and drop all edges that touch 5 or 6



last step after
take e_1
take e_5
take e_3.

Now, the C (approx. VC) is the set of all nodes in e_1, e_3, e_5:

$\{1, 2, 3, 4, 5, 6\}$.

Let E' be the edges taken by the alg. Since the edges do not share any nodes (no edges in

E', by defintion, touch each other). We have

$$|C^*| \geq |E'| \quad \text{and obviously,} \quad |C| = 2 \cdot |E'|.$$

best VC

The result $|C| \leq 2|C^*|$ follows.

Recall: TSP (Travelling salesman problem).

Given: a graph $G = (V, V \times V)$ which is a complete graph where each edge has a nonnegative weight $w(u,v) \geq 0$.

Goal: Find a HC^* with minimal total weight.

TSP $\longrightarrow$ impossibility of approx.

Recall VC has 2-approx (that is poly-time computable).

Want to prove: TSP has no ρ-approx for any ρ. (ρ-approx of TSP that is poly-time computable exists only when $P = NP$).