CSC791/495-011

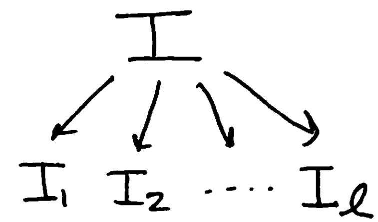
Dr. Blair D. Sullivan

September 1, 2017

Bounded Search Trees

<u>Idea</u>: Build a feasible soln through a sequence of decisions ("guessing"), each of which investigates options.

* Argue that your strategy guarantees yes-instance $\Rightarrow$ some sequence considered yields feas. soln.



<u>Key requirements</u>:

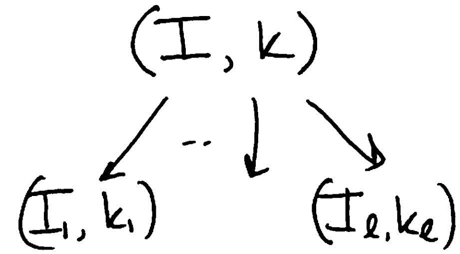
① $\ell \leq \mu(I)$

② $\mu(I_j) \leq \mu(I) - c$, c constant ≥ 1

③ every feasible soln to an I_j yields one for I and the set of feasible solns for $\{I_j\}_{j=1}^{\ell}$ contains at least one (optimum) soln for I.

To use this for an fpt algorithm:

(a) ② (depth of tree) must have ~~$\ell \leq f(k)$~~ # nodes in tree $\leq f(k)$



(b) each branch step (& eval. @ leaves) must run in poly-time (in $|I|$)

<u>Typical Strategy</u>: Figure out a set S so that <u>some</u> element of S is in every optimal soln.

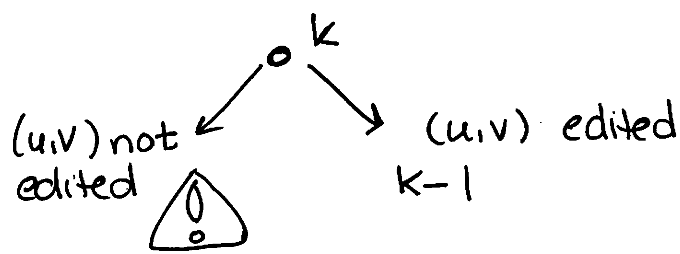
Cluster Editing, Take 2

Recall we gave an $O(k^2)$-vertex kernel for Cluster Editing $\Rightarrow O\left(\binom{k^2}{k} \cdot n^{O(1)}\right)$ algorithm.

What does the branching algorithm based on our P_3 observation yield?

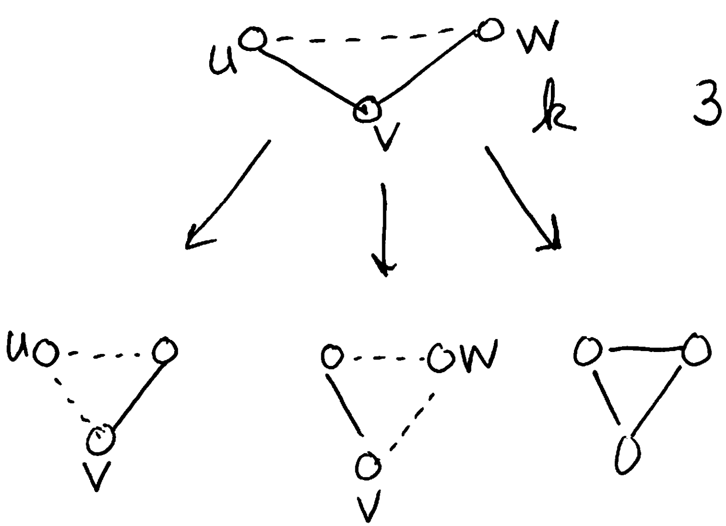
What set S of $\underbrace{\text{things}}_{\text{pairs of vertices}}$ must have at least one $\underbrace{\text{occur}}_{\text{edit edge}}$ in every feasible solution?

Suggestion pick (u,v)



Try Again break up/complete a P_3.

Q: can we find an induced P_3 in poly-time?
YES: n^3



3 ways to "resolve"

$$\left.\begin{array}{ll} \text{B1} & \text{delete } uv \\ \text{B2} & \text{delete } vw \\ \text{B3} & \text{add } uw \end{array}\right\} \text{all } k \to k-1$$

Stopping criteria: no P_3's left YES if $k \geq 0$
if $k=0 \Rightarrow$ say NO when P_3 exists.

Running time: $O(3^k) \cdot n^3$

observe: # nodes $\leq 3^{k+1} - 1 \leq \underbrace{3}_{\text{constant}} \cdot 3^k$

Min-Ones-r-SAT

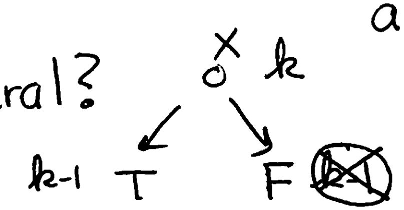
<u>Problem</u> Given an r-CNF formula φ and $k \in \mathbb{Z}^+$, is there a satisfying assignment for φ with at most k variables set to TRUE?

[sidebar: CNF = conjunctive normal form $(a \vee b \vee \neg c) \wedge (d \vee c) \wedge (\neg b)$; r-CNF $\Rightarrow$ r literals per clause]

Treat r as fixed constant.

<u>Goal:</u> check if some subset of $\leq k$ literals satisfies all clauses (if set to TRUE)

<u>Idea:</u> branch on a literal?

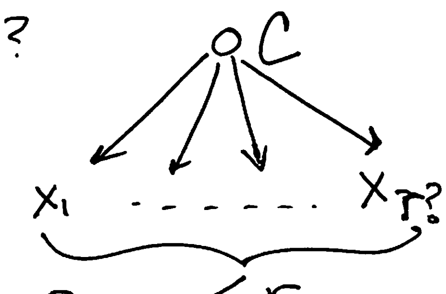


Same issue.

<u>Observation</u> every clause must be satisfied $\Rightarrow$ maybe we can pick a literal to set true from the clause.

Q: can we set all r to TRUE & satisfy C?

<u>NO</u> so restrict your attention to the <u>positive</u> literals in C



branch into $\leq r$ subproblems w/ 1 less clause & 1 less TRUE available.

$*$ pick an unsatisfied clause to branch on } poly-time ✓

{ If $k = 0$ & unsatisfied clause $\Rightarrow$ NO
If no unsatisfied clause $\Rightarrow$ YES

<u>Key information</u>: start w/ all literals FALSE.

$O(r^k) \cdot n^{O(1)}$

$f(k) \cdot n^c$

FPT ✓

In-Class Exercise

Problem Given a graph G and $k \in \mathbb{Z}^+$ does G contain a set of k vertices that are pairwise non-adjacent (an independent set).

Thm k-Independent Set is FPT on graphs of max degree d (a constant).

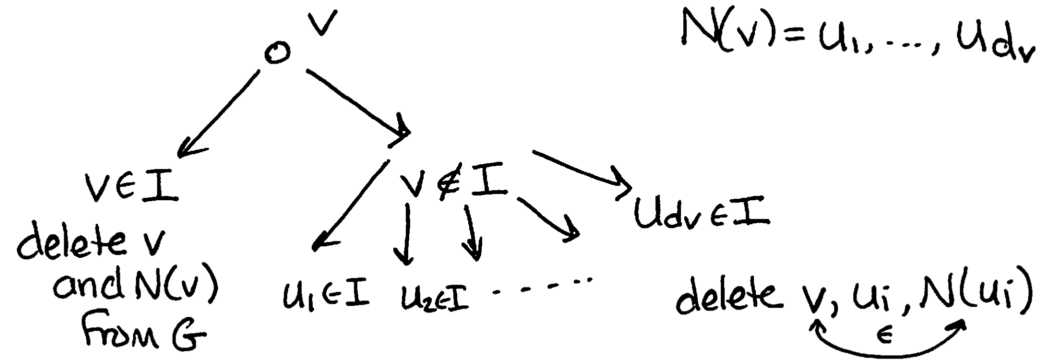
Prove this using a branching algorithm! Give the overall running time of your approach.

$N(v) = u_1, \dots, u_{d_v}$

Algorithm

branching rule:

pick a vertex v & branch on which of $\underbrace{v \cup N(v)}_{S}$ is in the indep. set.



Why is this "safe"? Every maximal indep. set contains ≥ 1 of S?

If none of $N(v)$ are in $I \Rightarrow$ we can add v for free, making it bigger.

stopping criteria: $k = 0 \Rightarrow$ YES

G has no vertices (& $k > 0$) $\Rightarrow$ NO.

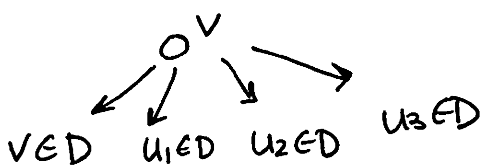
run-time: $\underbrace{O(d^k) \cdot n}_{\text{NOT QUITE}}$ b/c # branches $= \overbrace{|N(v)|}^{\leq d} + 1 \Rightarrow O((d+1)^k) \cdot n$

Domination

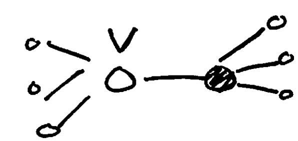
Problem Given a graph G and $k \in \mathbb{Z}^+$, is there a set of at most k vertices in G s.t. every vertex is either in the set or has a neighbor in the set?
(does G have a dominating set of size $\leq k$?)

Thm k-DominatingSet is FPT in graphs of max degree 3.

Idea pick a vertex $\Rightarrow$ either v must be in D or one of its neighbors is in D.



can we delete v from subproblems? ✓
What about $N(v)$? can't be sure they're not needed in dom. set.



problem: look at $(G \setminus \{v\}, k-1)$. Still have to dominate all vertices. Problem is that $N(v)$ was already dominated, so this might be a no-instance when original was a yes.

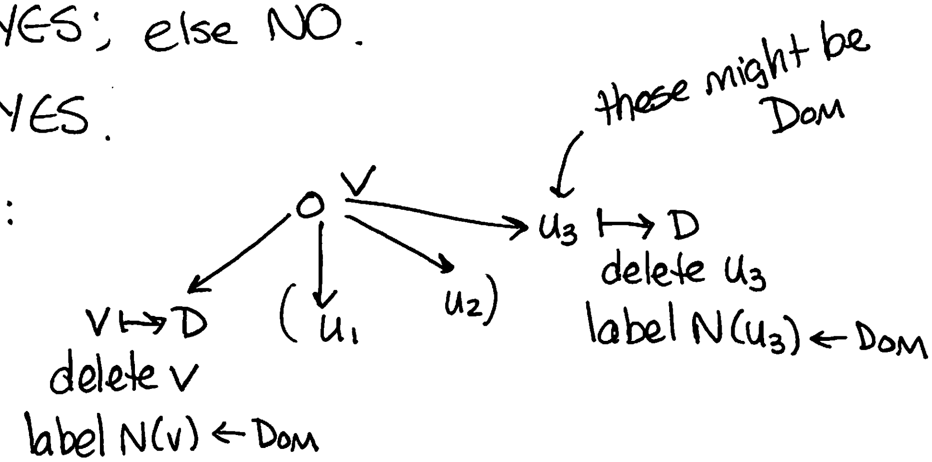
solution remember which vertices are already dominated - add labels. Un & Dom to every vertex. Now solve: Given labelled G is there a set S of $\leq k$ vertices s.t. every vertex labelled Un is in S or adjacent to a member of S?

Domination, cont

Initialization $f(v) = Un \ \forall v$

Algorithm Given G + $f: V \to \{Un, Dom\}$, $k \in \mathbb{Z}^+$

1. If $k=0 \Rightarrow$ If all labels are Dom $\Rightarrow$ YES; else NO.
2. $k \geq 0 \Rightarrow$ If all labels are Dom $\Rightarrow$ YES.
3. pick an undominated vertex & branch:



note: $k \to k-1$ in all branches.

running time: $O(4^k) \cdot n$

Vertex Cover, revisited

Naïve branching:

At every vertex, either $v \in$ cover or $N(v) \subseteq$ cover

v of max. degree

$k \to k-1$ add v to C & delete it

add $N(v)$ to C & delete them. $k \to ?$ $k - |N(v)|$

only better if $|N(v)| \geq 1$

stopping criterion: add (max degree 1 $\Rightarrow$ solve w/ BF) rute force

$$C(2^k)\, n^{O(1)}$$

this is sloppy analysis

Can we do better?

v

$v \in C$ $k \to k-1$

$N(v) \in C$ $k \to k - |N(v)| \leq k-2$

$T(k)$ = # leaves in a tree w/ param k.

$$T(k) = T(k-1) + T(k-2) \quad \&$$

$$T(i) = \begin{cases} T(i-1) + T(i-2) & i \geq 2 \\ 1 & \end{cases}$$

Claim: $T(k) \leq 1.6181^k$ How would you prove it? Induction

$\left(\frac{1+\sqrt{5}}{2}\right)$ where did this come? Want upper bound $c \cdot \lambda^k$

$$c \cdot \lambda^k \geq c \cdot \lambda^{k-1} + c \cdot \lambda^{k-2} \Longrightarrow \lambda^2 \geq \lambda + 1$$

Recurrence Relations

Solve for bounds on size of search tree using recurrences. Ours will have a (very) nice form – linear recurrences w/ constant coefficients.

$$T(k) = T(k-d_1) + T(k-d_2) + \cdots + T(k-d_r)$$

$T(k)$: # leaves

r = # branches

Solution to this is characterized by branching vector $(d_1, \ldots, d_r)$

& branching # (base of your α^k runtime) is the largest root of the characteristic polynomial

$$\lambda^d + \lambda^{d-d_1} + \cdots + \lambda^{d-d_r}$$

$d = \max\{d_1, \ldots, d_r\}$

In vertex cover we had vector $(1,2)$ & $\alpha = 1.61\ldots$

solved char poly. using quadratic formula

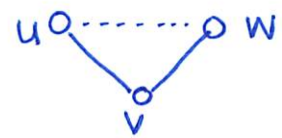
Branching Vectors (& Numbers)

(i, j)	1	2	3	4	5
1	2.0000	1.6181	1.4656	1.3803	1.3248
2		1.4143	1.3248	1.2721	1.2366
3			1.2560	1.2208	1.1939
4				1.1893	1.1674
5					1.1487

(l, i, j)	1	2	3	4
1	3.0000	2.4142	2.2056	2.1069
2	2.4142	2.0000	1.8929	1.7549

Cluster Editing (third time's the charm?) [(1, 2, 3, 3, 2)]

Design an improved branching strategy for Cluster Editing.

Consider an induced P_3: 

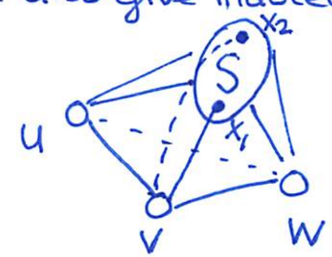
Before, we had 3 cases:

B1: $-(u,v)$
B2: $-(v,w)$
B3: $+(u,w)$

Let's look at structure more closely. Consider $S = (N(u) \cap N(w)) \setminus \{v\}$ (other mutual neighbors of u,w).
(why? b/c they will also give induced P_3's in many cases)

There are 3 cases:

C1: $S = \emptyset$
C2: $\exists\, x_1 \in S$ s.t. (x_1, v) is an edge
C3: $\exists\, x_2 \in S$ s.t. (x_2, v) is a non-edge



To do better branching in each case, we'll keep some extra info around (like in DomSet).

Let's annotate each vertex pair w/ a label $\tau(x,y) \in \{\emptyset, P, F\}$

$\emptyset$: unlabelled / no info
P: permanent / edge which cannot be removed
F: forbidden / non-edge which cannot be added

and apply a "reduction rule" anytime τ gets updated

so that $\tau(u,v) = \tau(u,w) = P \Rightarrow \tau(v,w) = P$ ( ; only way to resolve P_3)

and $\tau(u,v) = P$ + $\tau(u,w) = F \Rightarrow \tau(v,w) = F$ (similarly).

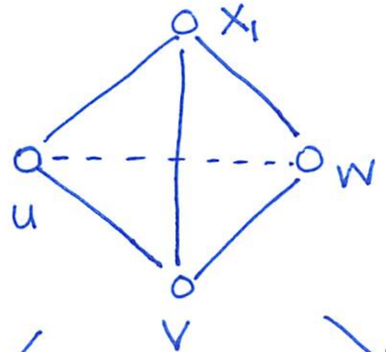
Case C1: Just branch into cases B1 & B2. You need to prove that B_3 cannot provide a better solution in this case (Lemma).

branching vector: $(1, 1) \Rightarrow$ branching # 2.0

CE III (Runtime Analysis)

On your own: work out case C3 to get (1,2,3,3,2)

Case C2:



first, let's branch on "edge" (u, w)

P & added

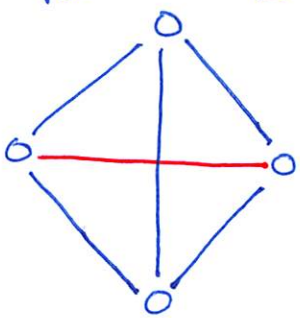
F & deleted

P

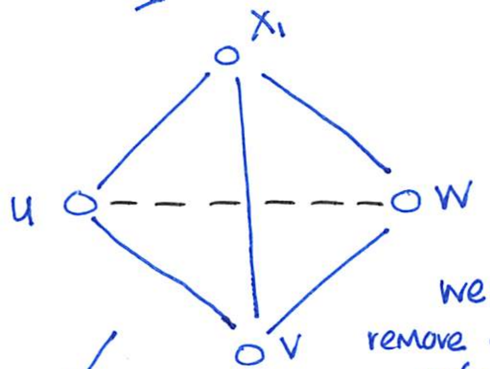
F

these will apply to enforce our branch conditions & as a result of ↑ reduction rules.

all visible P_3's resolved.

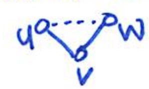


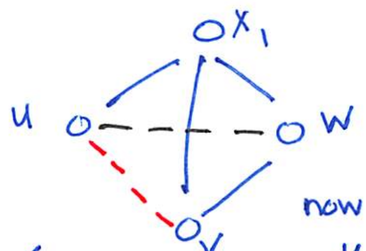
k-1



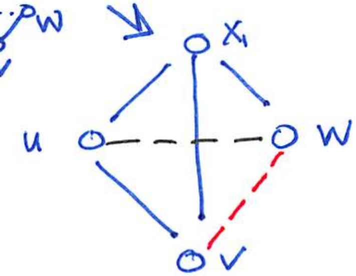
K

we ~~should~~ MUST remove either (u,v) or (v,w) to resolve





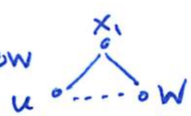
k-1



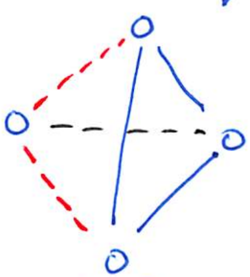
k-1

two more cases (k-2, k-3) as in removing (u,v).

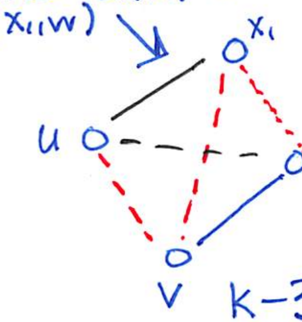
now



must be resolved by deleting either (u, x_1) or (x_1, w)



k-2



k-3

(u, x_1) is P b/c of branching

here, ↑ rule $\Rightarrow (x_1, v) \rightarrow$ F

C2 branch vector $(1, 2, 3, 2, 3) \Rightarrow$ branch# 2.27

Today's Problems

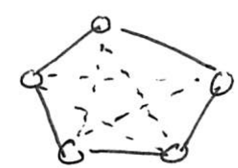
① Give a $2^k n^{O(1)}$ algorithm for Min-2-SAT using branching.

Min-2-SAT: Given a 2-CNF formula φ and $k \in \mathbb{Z}^+$, is there an assignment for φ that satisfies at most k clauses?

This week's proof write-up!

② Defn a graph G is chordal if it does not contain an induced cycle of length >3.

Chordal Completion Given a graph G and $k \in \mathbb{Z}^+$, can you add at most k edges to G to obtain a chordal graph?



* Give an FPT branching algorithm for chordal completion.

Helpful Lemma At least $k-3$ edges are needed to make a k-cycle chordal.

OCT perfection

Bonus Problem :)

<u>Defn</u> a graph is <u>perfect</u> if for every induced subgraph, the clique number is equal to the chromatic number.

<u>Problem</u> Given a graph G and $k \in \mathbb{Z}$ is there a set X of at most k vertices so that $G \setminus X$ is bipartite?

<u>O</u>dd <u>C</u>ycle <u>T</u>ransversal

<u>Thm</u> k-OCT has a $3^k n^{O(1)}$ branching algorithm in perfect graphs.