Algorithms: Design and Analysis, Part II

Exact Algorithms for NP-Complete Problems

Smarter Search for Vertex Cover

The Vertex Cover Problem

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

Goal: compute a minimum-cardinality vertex cover (a set $S \subseteq E$ that includes at least one endpoint of each edge of E).

Suppose: given a positive integer k as input, we want to check whether or not there is a vertex cover with size $\leq k$.

[think of k as "small"]

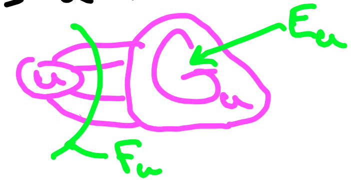
Note: could try all possibilities, would take $\approx \binom{n}{k} = \Theta(n^k)$ time.

Question: Can we do better?

A Substructure Lemma

Substructure Lemma: Consider graph G, edge $(u,v) \in G$, integer $k \geq 1$.
Let G_u = G with u and its incident edges deleted (similarly, G_v).
Then G has a vertex cover of size k $\iff$ G_u or G_v (or both) has a vertex cover of size $(k-1)$

($\Leftarrow$) Suppose G_u (say) has a vertex cover S of size $k-1$.
Write $E = \underbrace{E_u}_{\text{inside } G_u} \cup \underbrace{F_u}_{\text{incident to } u}$



Since S has an endpoint of each edge of E_u, $S \cup \{u\}$ is a vertex cover (of size k) of G.

($\Rightarrow$) Let S = a vertex cover of G of size k. Since (u,v) an edge of G, at least one of u,v (say u) is in S.
Since no edges of E_u incident on u, $S - \{u\}$ must be a vertex cover (of size $(k-1)$) of G_u.
QED

A Search Algorithm

[given undirected graph $G=(V,E)$, integer k]

[ignore base cases]

(1) Pick an arbitrary edge $(u,v) \in E$.

(2) Recursively search for a vertex cover S of size $(k-1)$ in G_u.
If found, return $S \cup \{u\}$.

G_u = G with u + its incident edges deleted

(3) Recursively search for a vertex cover S of size $(k-1)$ in G_v.
If found, return $S \cup \{v\}$.

(4) FAIL. [G has no vertex cover with size k]

Analysis of Search Algorithm

Correctness: straightforward induction, using the substructure lemma to justify the inductive step.

Running time: Total number of recursive calls is $O(2^k)$
[branching factor ≤ 2, recursion depth $\leq k$] (formally, proof by induction on k)

- also, $O(m)$ work per recursive call (not counting work done by recursive subcalls)

$\Rightarrow$ running time $= O(2^k m)$ → polynomial-time as long as $k = O(\log n)$
→ remains feasible even when $k \approx 20$

way better than $\Theta(n^k)$!