Lecture 9: Approximation Algorithms

Motivation:

If prob is NP-hard

Can't expect polytime algorithms that exactly solve the problem.

But.. it maybe possible to relax our goal and find approximate solutions efficiently.

"Optimization" Problems

Minimize a cost measure
or
maximize a quality measure

over a space of candidate solutions

e.g. Max Flow

Input: directed G, capacities $c(u,v)$, s, t

Candidates: all valid flows $f: E \to \mathbb{R}_+$

Quality: total flow leaving s

e.g. Minimum cut

Input: $G = G(V, E)$

Candidates: all non-trivial cuts $S \subseteq V$, $S \neq \emptyset, V$

Cost: $\text{cut}(S) = \{\{u, v\} \mid |\{u, v\} \cap S| = 1\}$

Both these problems admit an efficient algo that we studied in previous lectures.

Let's take a key example we will study today.

Minimum Vertex Cover

Definition: A set of vertices $S \subseteq V$ is a vertex cover in a graph $G(V, E)$ if for every edge $\{u, v\} \in E$, at least one of u or v is in S.

Informally, S "covers" or "touches" every edge.

Problem (Min VC)

Input: $G(V, E)$

Goal: compute a vertex cover S of G with minimum possible size.

Candidates: all possible vertex covers of G

Cost: size (= number of vertices)

Fact: Min VC is NP-hard. 

Theorem 0: There is an $O(m+n)$ time algo to compute a $S \subseteq V$ such that

1) S is a vertex cover.

2) ~~$|S| \leq 2 \cdot |S^*|$ where S^* is any min VC in G.~~ $|S| \leq 2 \cdot OPT$

Let's develop some basic vocabulary & understand this statement

Definition: $OPT(G) := $ minimum cost (for minimization problem)

$OPT(G) := $ maximum quality (for maximization problem)

Approximation Ratio

Let $\mathcal{A}$ be an algorithm to compute vertex cover.

For any graph G, let's write $\mathcal{A}(G)$ for the cost of the VC computed by $\mathcal{A}$.

Then: approx. ratio of $\mathcal{A}$

$$= \max_{\text{all possible graphs } G} \frac{\mathcal{A}(G)}{OPT(G)}$$

"Worst-case" multiplicative slack in the cost over and above the optimum cost.

Theorem 0 (same thm, new language)

There is an algorithm A for VC that

1) runs in time $O(m+n)$, and,

2) has approx. ratio ≤ 2.

Remark: we've chosen to measure the loss of quality or extra cost paid over and above the optimum in a multiplicative way.

We could have asked for "additive error": sol^n of cost OPT + EXTRA-COST & then taken the worst-case extra-cost paid as our measure to focus on. But this would become

less meaningful if OPT is tiny

Still, additive approx. can be sensible in some settings. We will not see them in this course.

Remark 2: Here's a basic conundrum of approx. algo design: we want to reason about the quality of solution found by the algorithm & compare it to OPT. But, we don't know OPT! Indeed, we're looking for approx. algos because!! we don't know how to compute OPT efficiently!

Key Idea "Certificates" of lower or upper bounds on OPT.

We will develop this important conceptual idea by focusing on VC → our current example/goal.

Bounding OPT

Let's ask the following question that seems to not be directly related to the goal of computing a min VC at first.

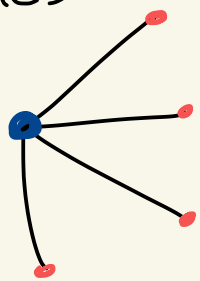
> Try to find structures in G that

Empty lower bounds on any VC.

Let's play with some examples.

1. Red vertices are a VC

Blue vertex is also a VC

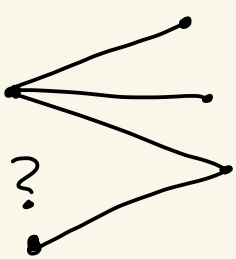


Blue vertex is also a min VC.

How do you prove this?

Easy: take any edge. Any VC must contain one of its end points & thus has size ≥ 1.

2. What about this?



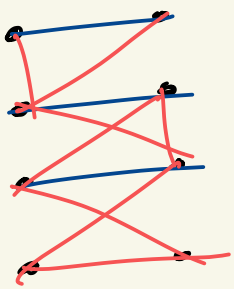
Can you find a min VC here?

How do you convince yourself that no better VC can exist?

Definition (Matching)

A matching $M \subseteq E$ in a graph $G(V, E)$ is a collection of edges s.t. no vertex participates in more than one (can be zero) edge in M.

e.g.



Blue edges form a matching

Caution: Matching may not touch all vertices of G.

Matchings yield a lower bound on Vertex Covers.

Lemma 1: Let $M \subseteq E$ be any matching. Then every vertex cover of G has size at least $|M|$.

Proof: Let S be a vertex cover of G. Then S must contain at least one vertex from every edge in M. Since the edges in M do not share vertices, $|S| \geq |M|$. □

What's the best lower bound we can hope to obtain on the VC in G this way?

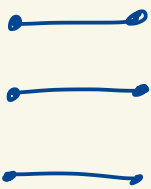
Corollary:

$$VC(G) \geq \text{Max-Matching}(G)$$

how good is such a certificate?

Can we use matchings to compute VCs?

e.g.



$G = K_6$

Clearly, all vertex covers of K_6 must use 5 vertices.

(All possible edges on 6 vertices)

But maximum matching is of size 3.

So if we just took a max matching & kept one vertex from each edge, we will definitely not get a vertex cover.

Key Observation: If we take both vertices of every edge in M, then we must get a vertex cover.

In fact this is true even if M is just maximal as opposed to maximum

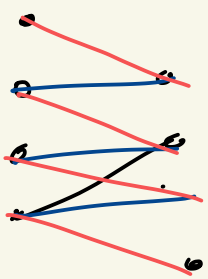
Definition (Maximal Matching)

A matching $M \subseteq E$ in $G(V,E)$ is maximal if for every edge $e \in E \setminus M$, $M \cup \{e\}$ is not a matching.

equivalently, every edge e in $E \setminus M$ touches some vertex already used in M.

Informally, you cannot grow M by adding any edge from the unused pile

e.g. blue edges form a maximal matching of size 3



red edges also form a maximal matching but of size 4.

Caution: 1) maximal matching can be smaller in size than a maximum matching (e.g. above)

2) A maximum matching is maximal.
(but not vice-versa, in general).

One cool observation is that it is very easy to compute a Maximal Matching.

Lemma 2: There is a $O(m+n)$ time algorithm to compute a maximal matching in $G(V,E)$ (assuming E is given as adj list)

Let $E_u = \{\{u,v\} \mid \{u,v\} \in E\}$

Input: $G(V,E)$

Alg:

Initialize $M := \phi$.

$E_{curr} := E$

$size := |E_{curr}|$

If $size > 0$

pick any $e = \{u,v\} \in E_{curr}$

$M \leftarrow M \cup \{e\}$

$E_{curr} = E \setminus E_u \cup E_v$

size ← size − |E_u| − |E_v| + 1

else, return M.

Informally: repeatedly add an edge {u,v} from $E_{curr} = E$ to M & remove all edges incident on u or v in E_{curr}. Stop when E_{curr} is empty.

Claim 1: runtime of the algo is O(m+n)

Proof:

Computing size & initializing E_{curr} takes O(m) time.

Picking an edge takes O(1) time.

Removing E_u & E_v takes O(1) time. Updating size takes O(|E_u| + |E_v|) time.

In total, since for each vertex u, E_u can be removed at most once,

$$\text{total cost} = O(m + \sum_u |E_u|)$$

$$= O(m)$$ □

Correctness: Let M be the output of the algo.

Claim 2: M is a matching.

Proof:

We argue by induction on # iterations of while loop.

Clearly ϕ is a matching → this completes the base case.

Inductively assume M is a matching after first i iterations.

In the next iteration, if any edge e is added, it must be incident on vertices not already touched by an edge in M since we remove all edges from E_{curr} that are incident on such vertices.

So $M \cup \{e\}$ must be a matching.

If no such edge exists, nothing to prove. □

Claim 3: M is maximal.

Proof:

Suppose not for a contradiction.

Then, there must be an edge

$e = \{u, v\}$ that is in E but no edge incident on u or v is part of M. But then

$$E_{curr} = E \Big| \bigcup_{\substack{u : \exists v \\ \{u,v\} \in M}} E_u \neq \emptyset.$$

Contradicting the termination condition of the algorithm. □

This completes the analysis of the algorithm □

Vertex Covers from Maximal Matchings

Key: If you take both vertices of every edge in a maximal matching, then it's a vertex cover.

Lemma 3: Let $M \subseteq E$ be a maximal matching. Then,

$$S = \{u \mid \exists v : \{u, v\} \in M\}$$

is a vertex cover of $G(V, E)$.

Proof: Suppose S is not a V.C. of G. Then there must exist

an edge $\{u,v\} \in E$ such that

$$S \cap \{u,v\} = \emptyset.$$

But then $M \cap E_u \cup E_v = \emptyset$ since S contains all vertices touched by any edge in M. In that case $M \cup \{\{u,v\}\}$ is a matching & thus M is NOT maximal. This is a Contradiction.

□

We have thus arrived at a natural algorithm to compute a VC in G by relating VCs to matchings.

Algo: Input: $G(V, E)$

Operation: 1) Use algo above to find a maximal matching M in G

2 Output $S = \{u \mid \exists v : \{u, v\} \in M\}$

Lemma 4: S output by the algo is a VC

Proof: Immediate from Lemma 3.

Lemma 5: $|S| \leq 2\,OPT(G)$.

Proof: Let M be the matching computed in step 1.

Then by lemma 1,

$$OPT(G) \geq |M|.$$

OTOH, clearly, $|S| \leq 2 \cdot |M|$

So: $|S| \leq 2 \cdot |M| \leq 2 \cdot OPT(G)$.

□

We have thus established the Theorem 0.

Take-aways:

One can think of this as a greedy algorithm (since M is computed greedily) for approx V.C.

Key Insight: 1) matchings can be used to give a lower bound on all V.C.s.

2) maximal matchings can be

used to construct a vertex cover.

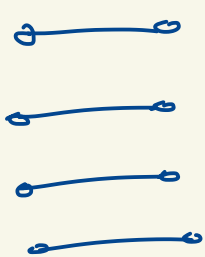
3) maximal matchings are easily computable.

Did we analyze the algorithm optimally?

Could our algorithm do better than 2-approx.?

→ Sometimes

e.g. if the graph itself is a matching, our



algo has approx. ratio of $\underline{\underline{1}}$.

Here's another example where the algo does quite well.

Exercise: Let K_{2n} be the complete graph on $2n$ vertices. Then,

1) any VC of K_{2n} has size $2n-1$, and,

2) every maximal matching of K_{2n} has size n.

So our algo gets an approx. ratio of $\frac{2n}{2n-1} = 1 + \frac{1}{2n-1} \to 1$ as $n \to \infty$.

But in the worst-case our algo is tight!

Exercise: Let $K_{n,n}$ be the complete bipartite graph with left & right vertex sets each of size n. Then

1) $\text{Min-VC}(K_{n,n}) = n$.

2) Every maximal matching of $K_{n,n}$ has size n.

Thus our algo for $K_{n,n}$ will output a VC of size $2n$ even though there's a VC of size n in the graph.

— x —