CSE515 Advanced Algorithms
Lecture 16
Algorithms for Vertex Cover

Antoine Vigneron
antoine@unist.ac.kr

Ulsan National Institute of Science and Technology

April 22, 2021

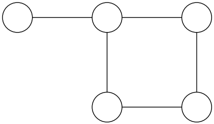
Introduction

- In Lecture 14, we saw that VERTEX COVER is **NP**-hard, hence a polynomial-time algorithm is currently out of reach.
- In this lecture, we will present two ways of dealing with it.
- I will not be following the textbooks closely in this lecture.
- **References**:
  - Chapter 35.1 of the textbook Introduction to Algorithms by Cormen, Leiserson, Rivest and Stein.
  - Chapter 10.1 of the textbook Algorithm Design by Kleinberg and Tardos.

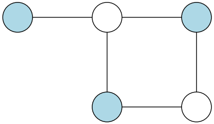
Vertex Cover

Definition (vertex cover)

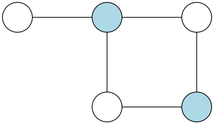
Given a graph $G(V, E)$ with vertex set V and edge set E, a *vertex cover* is a subset $V' \subseteq V$ of vertices such that each edge $e \in E$ is incident to at least one vertex in V'.



input graph



a vertex cover

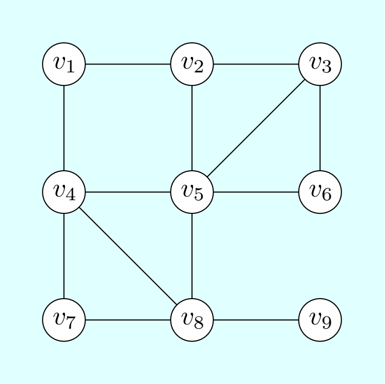


minimum vertex cover

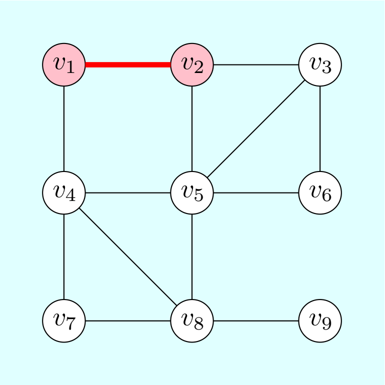
Problem (VERTEX COVER)

The vertex cover problem is to find a vertex cover of smallest cardinality.

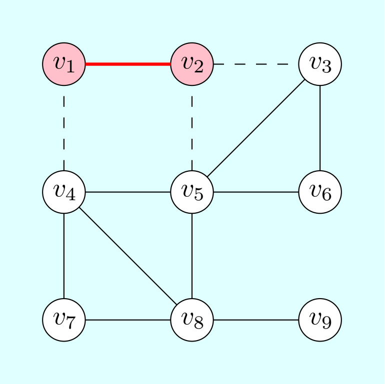
An Algorithm



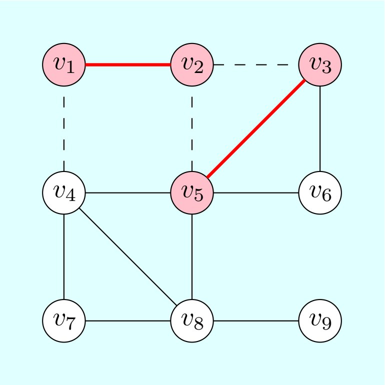
An Algorithm



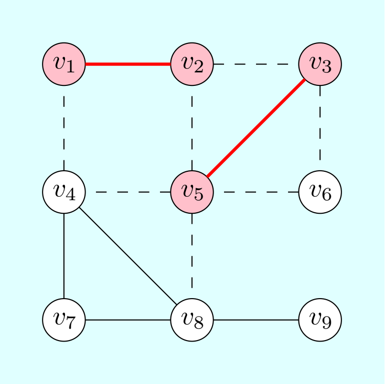
An Algorithm



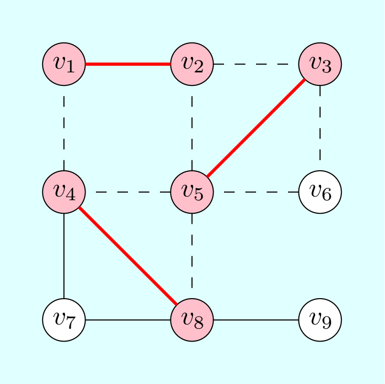
An Algorithm



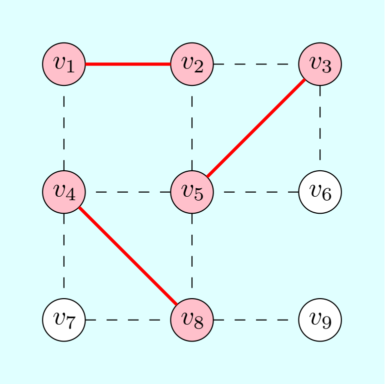
An Algorithm



An Algorithm



An Algorithm



An Algorithm

- So we obtained a vertex cover $\{v_1, v_2, v_3, v_4, v_5, v_8\}$ of size 6.

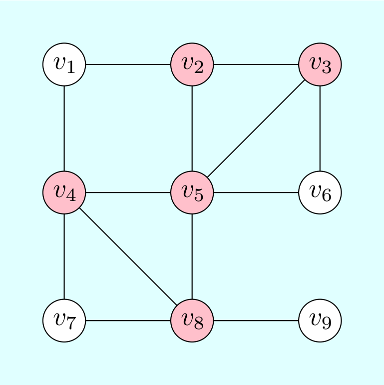
Pseudocode

```
procedure ApproxVertexCover(G)
    C ← ∅
    while E ≠ ∅ do                          ▷ E is the set of edges
        let (u, v) be an arbitrary edge in E
        C ← C ∪ {u, v}
        remove from E every edge incident to u or v
    return C
```

- How good is this algorithm?

An Algorithm

- ApproxVertexCover can be implemented to run in time $O(n+m)$, where n is the number of vertices and m is the number of edges.



- So this is *linear* time.
- But it does not always return a *minimum* vertex cover.
- In the example above, the optimal size is 5, and we returned a cover of size 6.

An Algorithm

- Let C^* denote a minimum vertex cover.

Theorem

$\textsc{ApproxVertexCover}$ *returns a vertex cover C of size at most $2|C^*|$.*

Proof.

See lecture notes. $\square$

- We say that $\textsc{ApproxVertexCover}$ is a *2-approximation algorithm*.
- Is our analysis tight? For instance, is it a 1.99-approximation algorithm? (See lecture notes.)

Approximation Algorithms

Definition

An algorithm for a *minimization* problem is an *α-approximation algorithm* iff on every input, it returns in polynomial time a solution whose value is at most α times the minimum.

- So in the definition above, $\alpha > 1$.
- This notion also applies to maximization problems:

Definition

An algorithm for a *maximization* problem is an *α-approximation algorithm* iff on every input, it returns in polynomial time a solution whose value is at least α times the maximum.

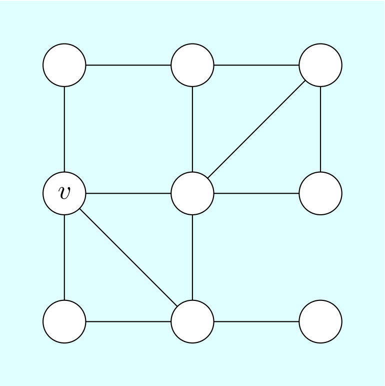
- So in for maximization problems, $0 < \alpha < 1$.
- We will see examples of maximization problems later in this semester.

Approximation Algorithms: Remarks

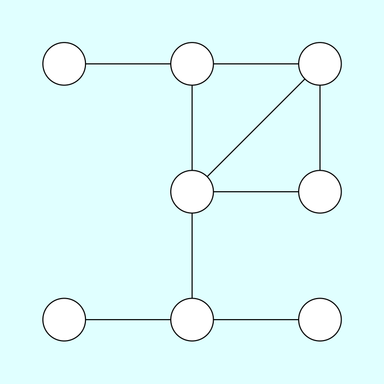
- You cannot say that an algorithm is an approximation algorithm if you cannot prove that it returns a solution within a factor α from the optimum for all input.
- It differs from *heuristics*: A heuristic (such as simulated annealing or a genetic algorithm) may return good results on some input, but in the worst case, we cannot prove that the solution is within a factor α from the optimum.
- In this course we require approximation algorithms to run in *polynomial time*. In some textbooks the definition does not require it.

Notation

- If G is a graph and v is a vertex of G, then $G \setminus \{v\}$ is the graph obtained from G by deleting v and all incident edges.



G

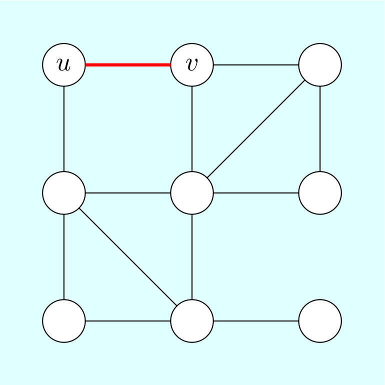


$G \setminus \{v\}$

Another Approach

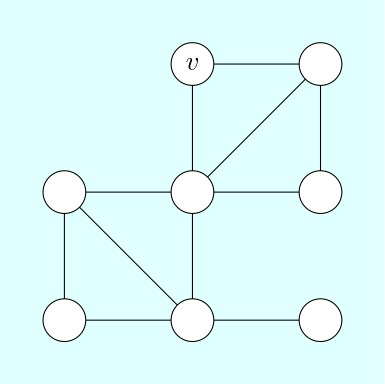
- Each time $\textsc{ApproxVertexCover}$ picks an edge (u, v), it covers both u and v. This is why it only provides a 2-approximation.
- We will try to improve it as follows: We try to cover u *or* v, not necessarily both.
- So we will recurse on $G \setminus \{u\}$ and $G \setminus \{v\}$, and keep the best answer.
- We will try to find a cover of size $\leqslant k$ for some given k.

Algorithm

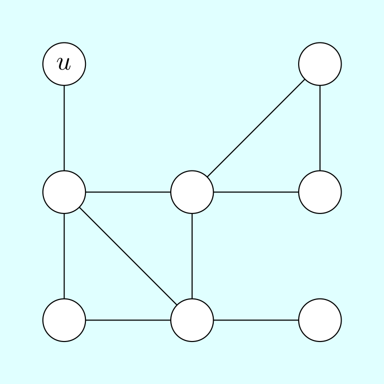


Pick edge (u, v)

Algorithm



Recurse on $G \setminus \{u\}$
then add u to the cover



Recurse on $G \setminus \{v\}$
then add v to the cover

Algorithm

Pseudocode

```
procedure FPVertexCover(G, k)
    if E = ∅ then
        return ∅
    if m ⩾ kn then
        return INFEASIBLE
    pick an edge (u, v) ∈ E
    C ← FPVertexCover(G \ {u}, k − 1)
    if C ≠ INFEASIBLE then
        return C ∪ {u}
    C ← FPVertexCover(G \ {v}, k − 1)
    if C ≠ INFEASIBLE then
        return C ∪ {v}
    return INFEASIBLE
```

- n is the number of vertices, E is the set of edges and $m = |E|$.

Algorithm

- This algorithm returns a vertex cover of size $\leqslant k$ if there is one. Otherwise it returns INFEASIBLE.
- Line 3: Basis step.
- Line 5: We can cover at most $n - 1$ edges with one vertex, so there is no solution if there are at least kn edges.
- Line 9: If C covers $G \setminus \{u\}$ and $|C| \leqslant k - 1$, then $C \cup \{u\}$ covers G and has $\leqslant k$ vertices.
- Line 13: In this case there is no solution. (Proof in lecture notes.)

Theorem

FPVertexCover *returns a vertex cover of size k, if there is one. It runs in time $O(2^k kn)$.*

- (Proof in lecture notes.)

Algorithm

- How good is this running time?
- Suppose you are looking for a cover of size at most 10.
- Then the algorithm runs in time $O(2^{10} \cdot 10 \cdot n) = O(10240n) = O(n)$.
- This is *linear* time.
- More generally, if the size of the cover is constant, i.e. $k = O(1)$, this algorithm still runs in linear time.
- On the other hand, if k can be arbitrarily large, then this algorithm runs in exponential time.
- What is the running time of the brute-force approach? How does it compare? (See lecture notes.)

Algorithm

- More generally, if a problem of size n can be solved in time $O(f(k) \cdot n^c)$ where c is a constant, and k is a parameter, then we say that the problem is *fixed-parameter tractable* with parameter k.
- In particular, the algorithm is polynomial if k is constant.
- This is a good approach when you are only interested in inputs such that k is small.

Concluding Remarks

- From Lecture 14: We do not know any polynomial-time algorithm for **NP**-hard problems.
- We mentioned several ways of dealing with it:
  - Brute-force if the input size is small.
  - Approximation algorithms (i.e. provable approximation factor).
  - Heuristics (not provable).
  - Fixed-parameter algorithm.
- We will not study heuristics in this course.
- We will not study fixed-parameter algorithms further. Instead we will focus on approximation algorithms.