Large-scale Knowledge Processing

Lecture 9

Kazuhisa Seto

Today's Lecture

Study algorithms for NP-complete problems.

- 2-Approximation Algorithm for the Minimum Vertex Cover
- Exact Algorithms and FPT Algorithms
- FPT Algorithms for Vertex Cover

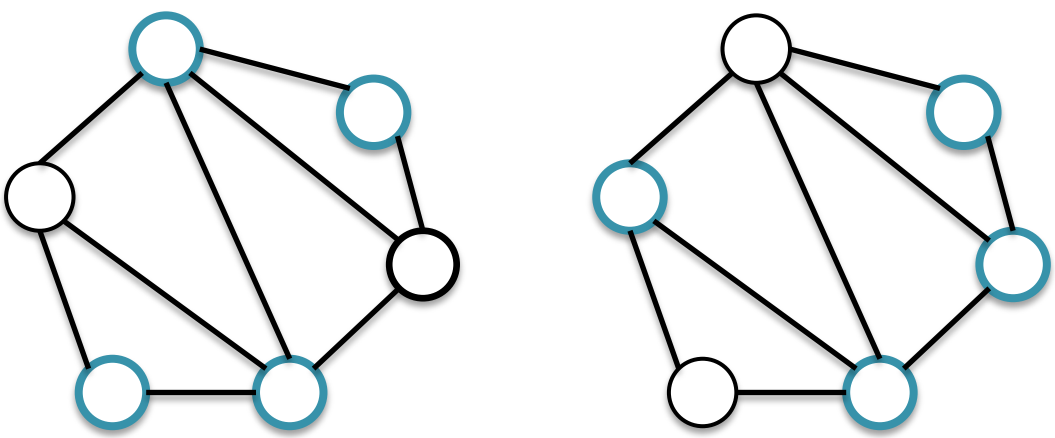
2-Approximation Algorithm for the Minimum Vertex Cover

Review: Vertex Cover (VC)

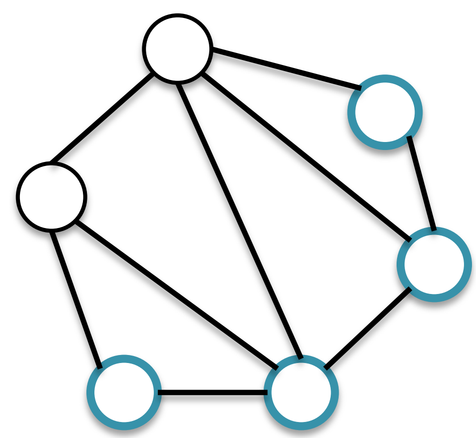
Input : A graph G and a positive integer k

Ask : Is there a vertex cover of size k in G ?

Vertex Cover : A set of vertices C such that for every edge e, at least one endpoint of e in C.



Blue circles are vertex cover

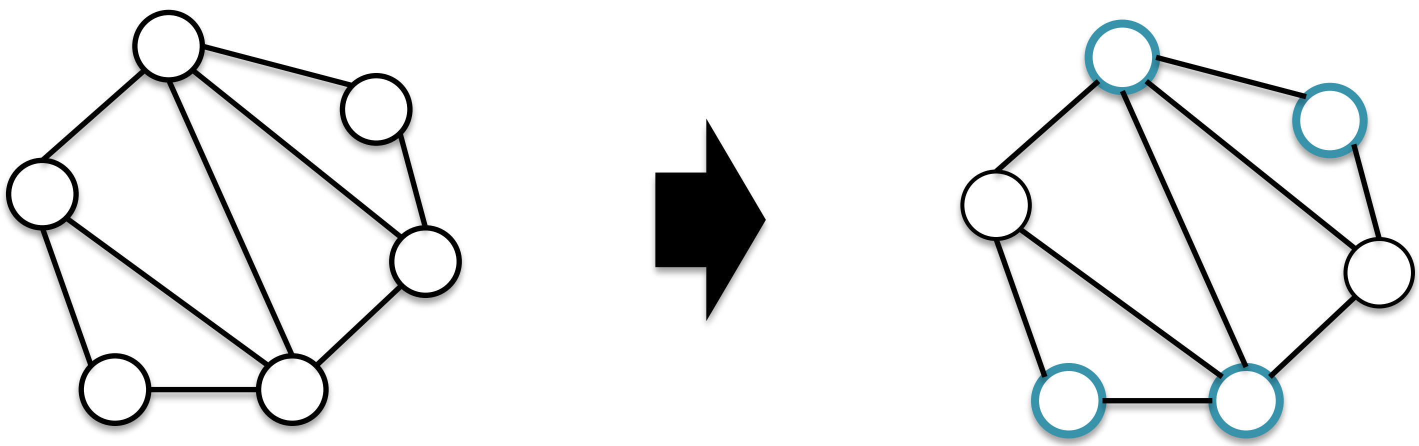


Not vertex cover

The definition of Minimum Vertex Cover Problem（MVC）

Input：A graph G

Output：A vertex cover U whose size is minimum.



G

U: blue vertices

There is no vertex cover of size 3.

History of Approximation ratio for MVC

Polynomial-time approximation algorithms

- 2-approximation [Gavril, Yannakakis]
- $2 - 1/\Theta\left(\sqrt{\log V}\right)$ approximation [Karakostas 04]

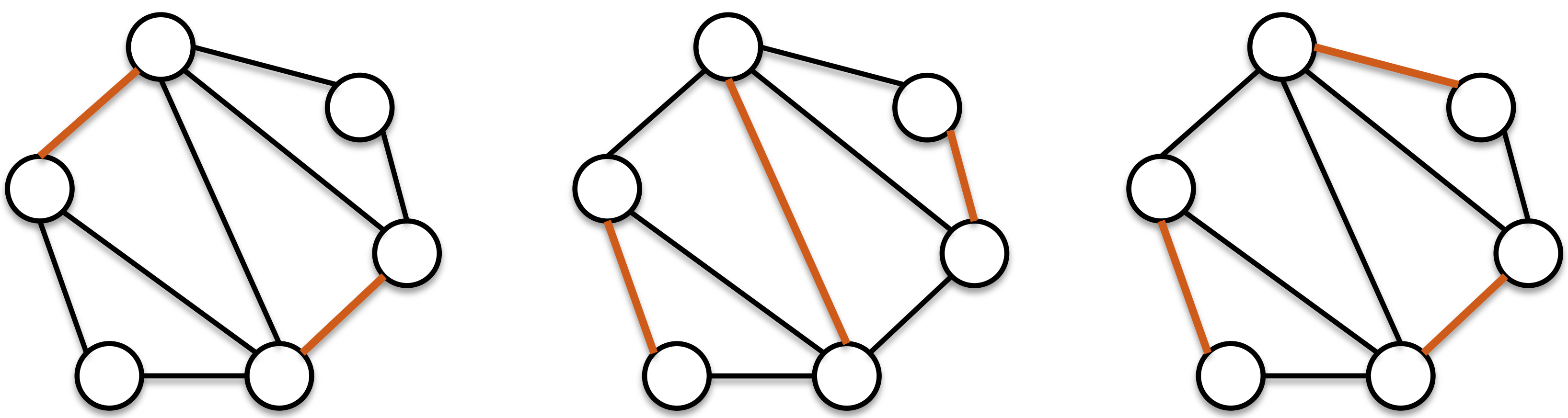
Hardness of approximation

- Unless P=NP, there in no 1.3606-approximation polynomial-time algorithm. [Dinur and Safre 2005]
- If the Unique Games Conjecture holds, then there is no 2- ε approximation polynomial-time algorithm [Knot and Regev 2003]

The definition of Matching

<u>Matching</u>

A matching M is a set of edges of a graph G such that any pair of edges in M cannot share any vertex.

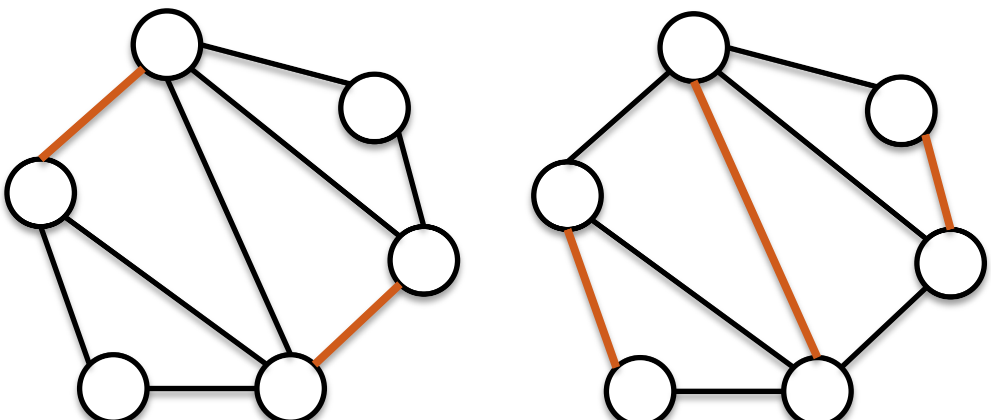


Orange lines are a matching

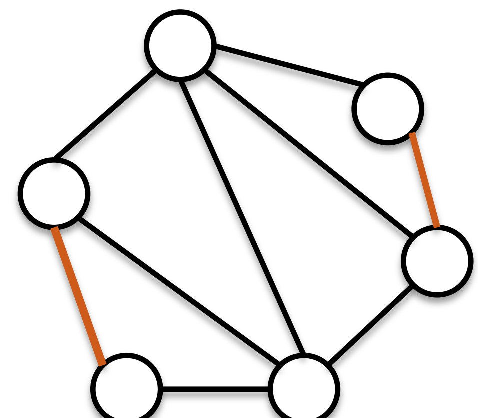
The definition of Maximal Matching

Maximal Matching

- A **maximal matching** is a matching M of a graph G that is not a subset of any other matching



Maximal Matching

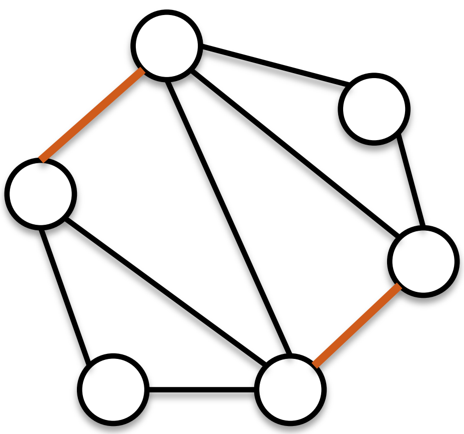


Not Maximal Matching

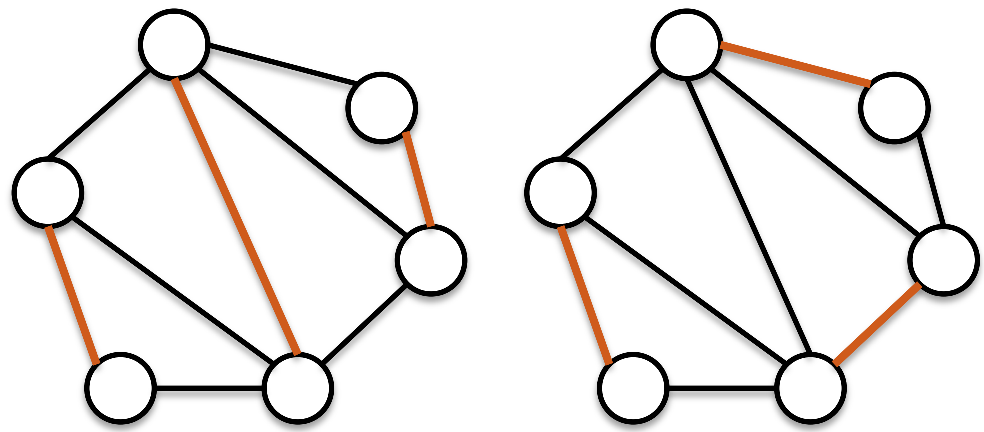
The definition of Maximum Matching

<u>Maximum Matching</u>

- A Maximum Matching is a maximal matching M such that the number of edge in M is maximum.



Not Maximum Matching



Maximum Matching

2-approximation algorithm for the Minimum Vertex Cover

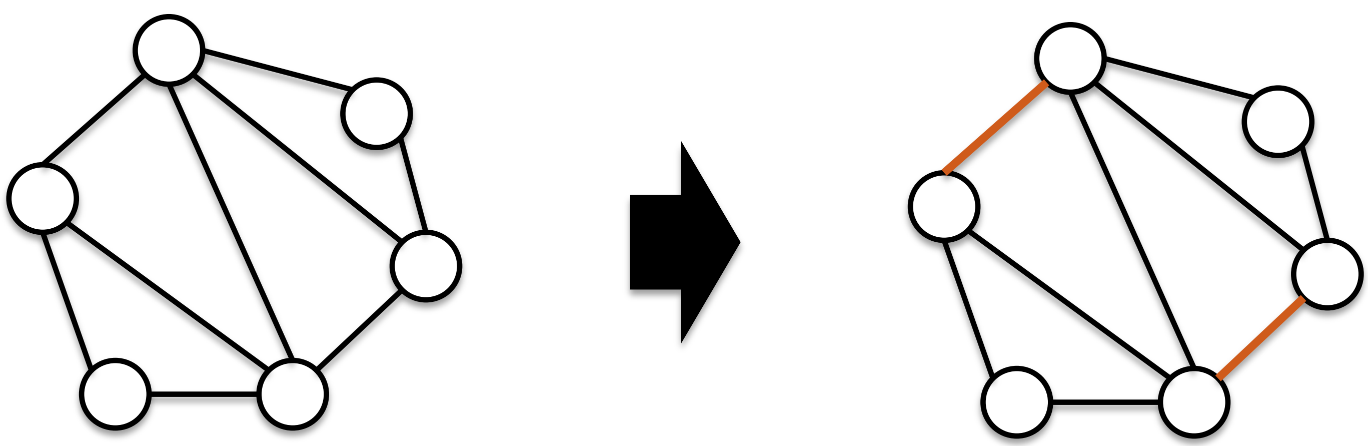
The algorithm is based on a maximal matching.

1. Compute a maximal matching M of a graph G.
2. Include both endpoints of each edge in a vertex cover U'
3. Outputs U'

It's quite simple algorithm!

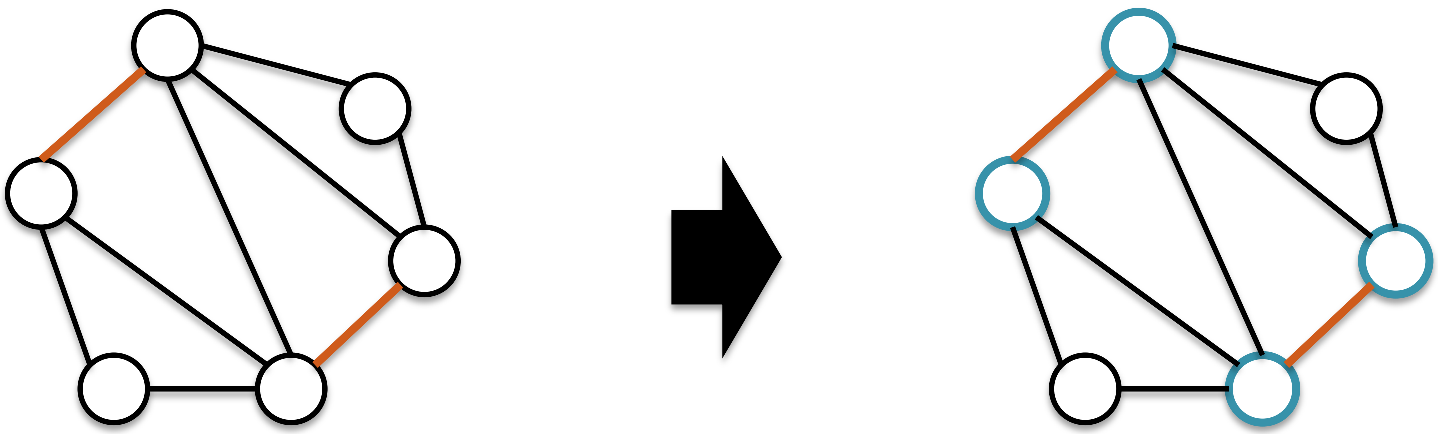
An example of algorithm's procedure

1. Compute a maximal matching M of a graph G
   - ➢ We repeat the following operation until checking all edges.
     - • We pick up an edge e at random.
       - ✓ If both endpoints of e are not included in M, we add e to M.
       - ✓ Otherwise, we discard e.



An example of algorithm's procedure

2. Include both endpoints of each edge in a vertex cover U' (blue vertices)
3. Outputs U'



The running time of the algorithm

Let n and m be the number of vertices and edges of a graph G, respectively.

Algorithm:

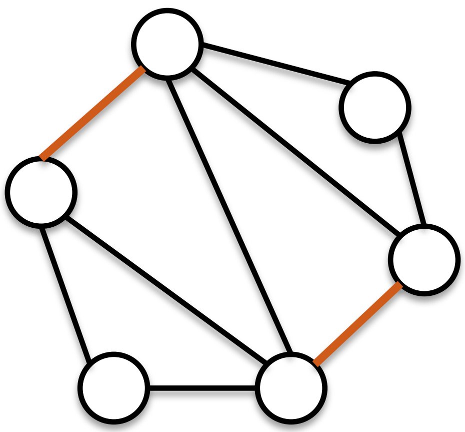
1. Compute a maximal matching M of a graph $G \rightarrow O(m)$
2. Include both endpoints of each edge in a vertex cover U' $\rightarrow O(n)$
3. Outputs $U' \rightarrow O(n)$

The overall running time is $O(n+m)$.

A proof of 2-approximation

Let U be a minimum vertex cover and U' be a vertex cover that the algorithm outputs and M be a maximal matching of G.

Then, the followings hold.



1. $|U| \leq |U'|$ because U is minimum.
2. $|M| \leq |U|$ because any edges must be covered by one element in U.
3. $|U'| = 2|M|$ because the property of the algorithm.

From 2 and 3, $|U'| = 2|M| \leq 2|U|$ holds.

Thus, this algorithm satisfies 2-approximation.

Exact Algorithms and FPT Algorithms

Exact Algorithm for NP-complete Problems

In some case, we want an optimal solution.

- We cannot take an exponential time.
- A solution is assured to be optimal.

A simple exact algorithm for vertex cover : for any subset with size k of vertices, we check whether it is a vertex over or not.

It takes $O(n^k m) = O(n^{k+2})$ time because the number of subsets is $O(n^k)$ and checking procedure takes $O(m)$ time for each subset.

Exact Algorithm for NP-complete Problems

$O(n^{k+2})$ is not efficient!

If k is some constant, it is polynomial of n.

However, when $n = 10000$ and $k = 10$, then $O(n^{k+2})$ is $10000^{12} = 10^{48} \cdots$ It's quite huge!

When k is small, can we seek a solution efficiently?

Improving $O(n^{k+2})$ time to $O(n^{0.5k})$ time is not enough⋯

The form $O(n^{f(k)})$ is not prefered, where $f(k)$ is some function of k.

Fixed Parameterized Tractable

FPT (Fixed Parameterized Tractable)

- Input size: n, a paremeter: k
- FPT algorithm can solve a problem in $O\left(f(k)\times n^{O(1)}\right)$ time.

Is FPT algorithm polynomial-time algorithm?

- NO. Because k may depend a function of n
  - ✓ Though $O(2^k n)$ time is FPT,
    when $k = n$, it takes $O(n2^n)$
- However, when k is small, FPT algorithm can solves a NP-complete problem efficiently.

The importance of FPT algorithms

Why are FPT algorithms important?

- We can obtain an optimal solution.
- If k is small, it becomes an efficient algorithm.
  - ✓ Note that it is polynomial time of n.
  - ✓ An algorithm runs in $O(2^k n)$ time,

    then when $n = 10000$ and $k = 10$, it takes about

    10^7 time $> O\left(n^{k+2}\right) \fallingdotseq 10^{48}$

Does FPT algorithm always exist?

It have yet been proven, but probably it is not true.

There should be a hierarchy of difficulty within the NP complete problem.

- A set of problems that has a FPT algorithm.
- A set of problems that has no FPT algorithm.
  - ✓ W-hierarchy : W[1]-complete, W[2]-complete, ⋯

A History of FPT algorithm for Vertex Cover

FPT algorithm

- $O(2^k n)$ time：simple branch and bound (explain later)
- $O(k^2 2^k + n)$ time by using kernelization (explain later)
- $O(1.2738^k + kn)$ time [Chen, Kanji and Xia 2006]

Hardness

- There exists no $2^{o(k)} n^{O(1)}$ time algorithm if ETH (Exponential Time Hypothesis) is true.

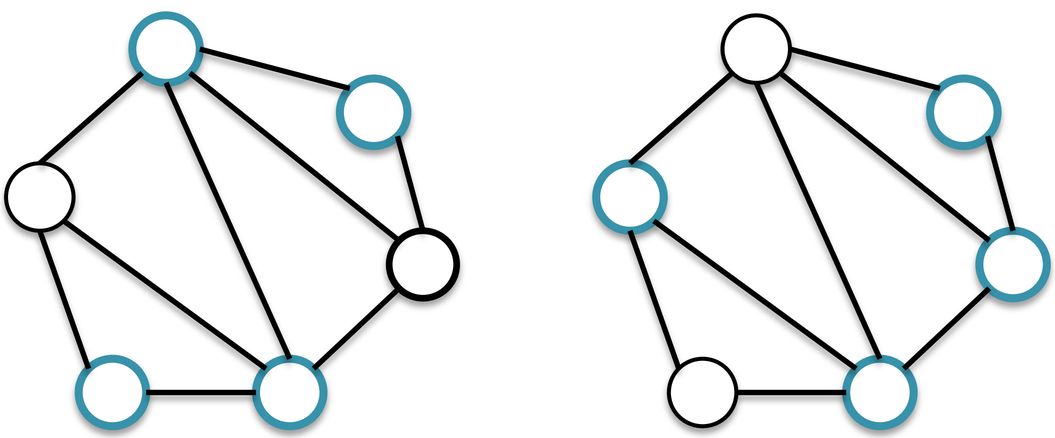
FPT algorithm for Vertex Cover

Review: Vertex Cover (VC)

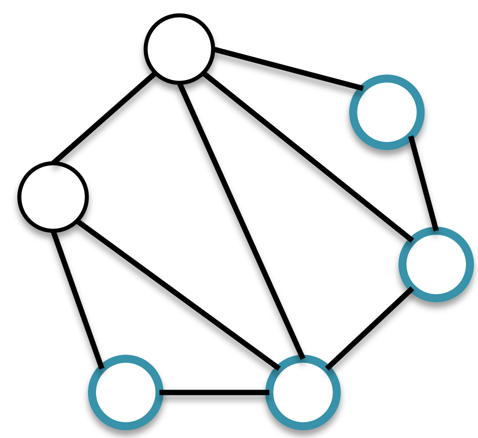
Input : A graph G and a positive integer k

Ask : Is there a vertex cover of size k in G ?

Vertex Cover : A set of vertices C such that for every edge e, at least one endpoint of e in C.



Blue circles are vertex cover



Not vertex cover

$O(2^k n)$ FPT algorithm for VC

G - v is a graph G' obtained by removing the vertex v and all edges connected to v from G.

BS(G, k) : G has n vertices.

1. If there is no edge in G, then return 0
2. If $k = 0$, then return $n + 1$
3. Pick up an edge (u, v) in G at random

   return min(BS$(G - u, k - 1)$, BS$(G - v, k - 1)$)+1

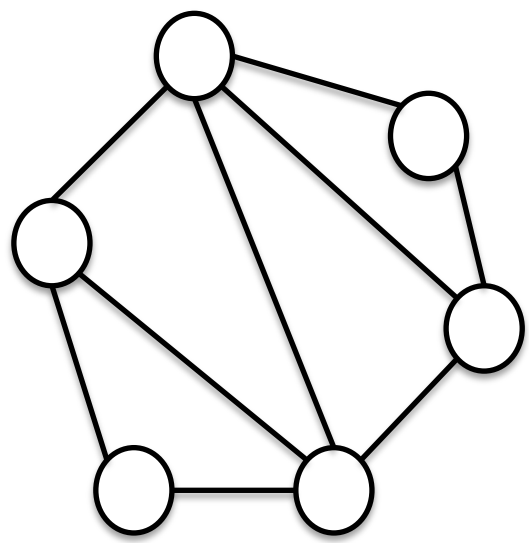
The value of BS(G, k) is less than or equal k, outputs Yes.

Otherwise, outputs No.

An example of algorithm's procedure

Now, an input graph G is the following and $k = 4$ for VC.

If there exists a vertex cover of size at most 4, then algorithms outputs Yes. Otherwise, it outputs No.

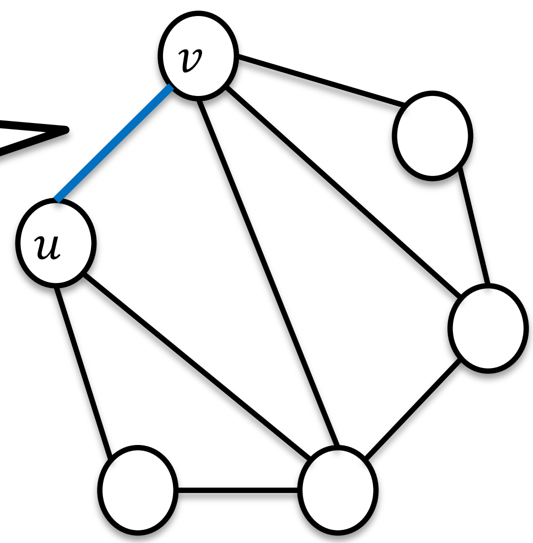


An example of algorithm's procedure

Since G has 9 egdes and $k = 4$, then algorithm first pick up an edge (u, v) at random.

Now, we assume that it choose a blue edge.

Choose this edge



An example of algorithm's procedure

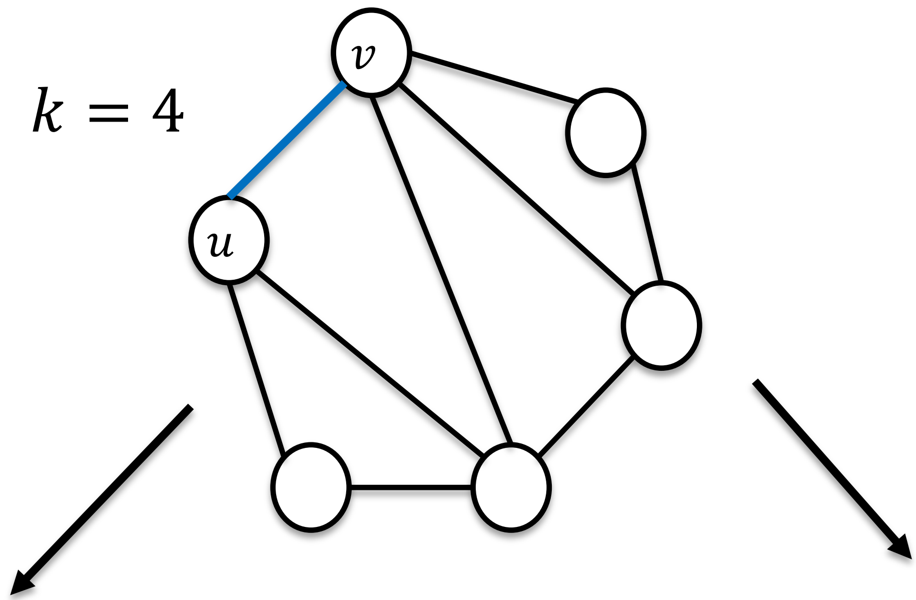
BS$(G-u, k-1)$ and BS$(G-v, k-1)$ means either u or v is included in vertex cover, then algorithm branches to two cases.

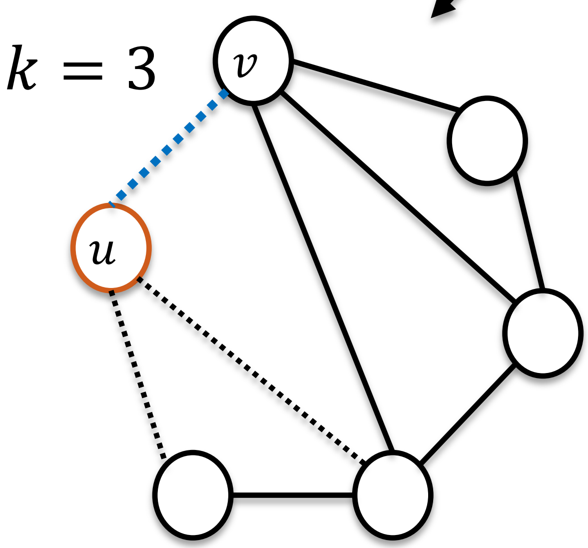
a. Vertex Cover Problem with a graph $G-u$ and $k=3$

b. Vertex Cover Problem with a graph $G-v$ and $k=3$

See the next slide.

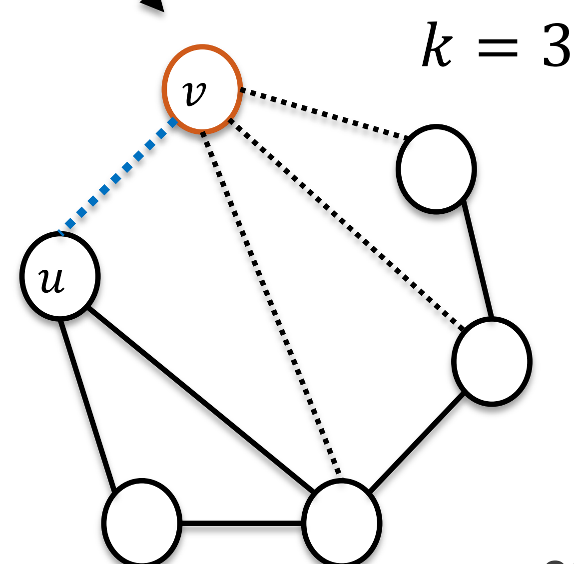
An example of algorithm’s procedure





Dot line edges
are removed from G

An orange vertex is
in a vertex cover.

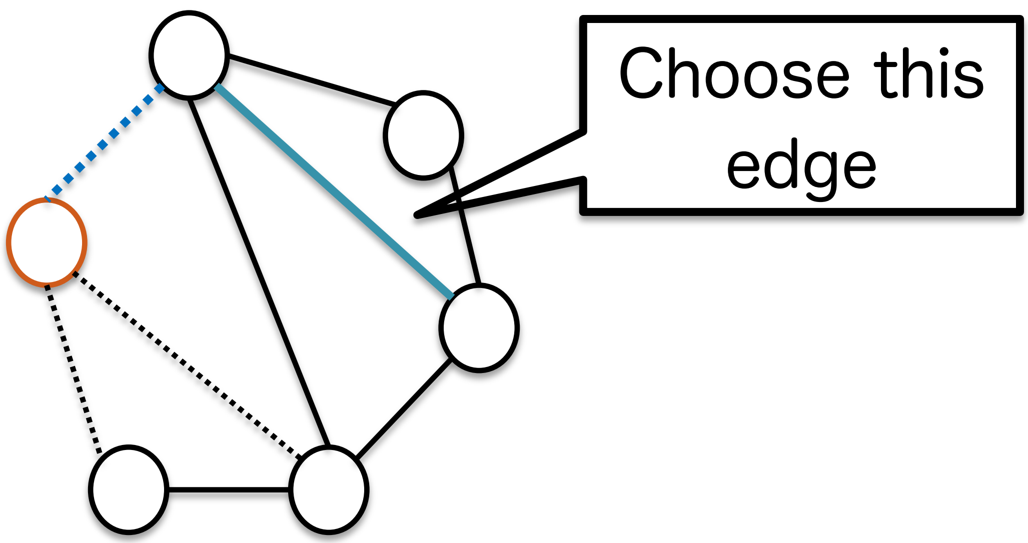


An example of algorithm's procedure

Next, the graph $G - u$ has 6 edges and $k = 3$, then algorithm pick up an edge at random.

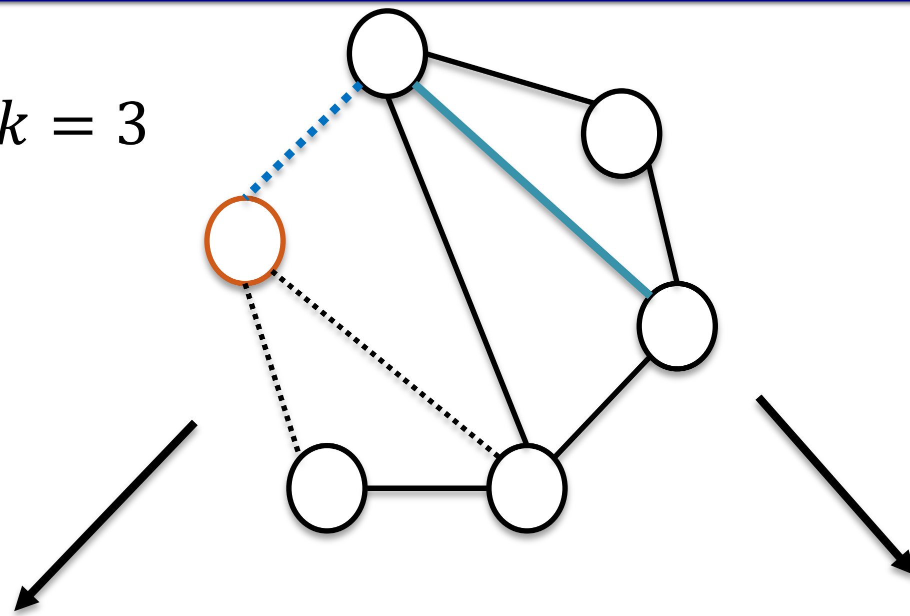
Now, we assume that it choose a blue edge.

Branches to two cases in the same way of the previous slide.

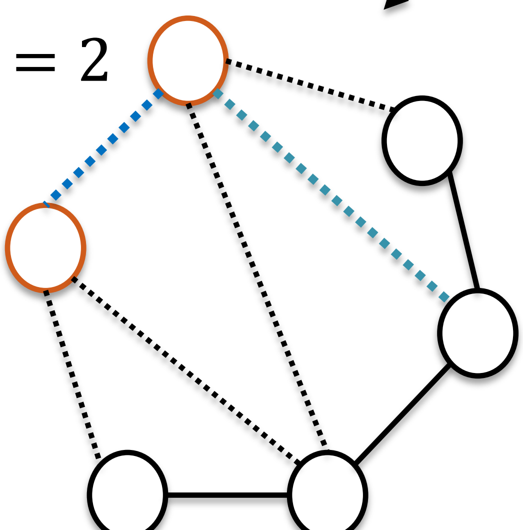


An example of algorithm’s procedure

$k = 3$



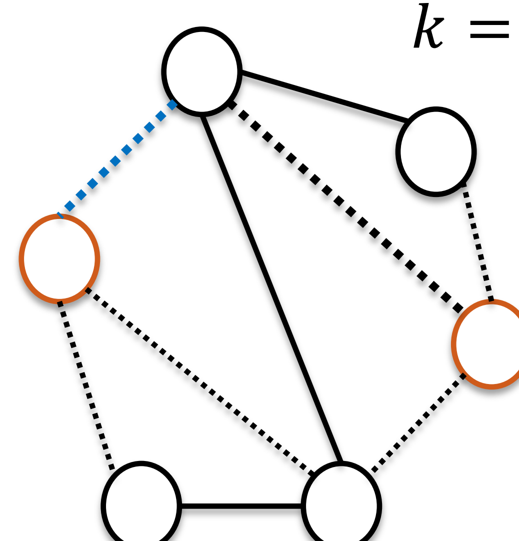
$k = 2$



Dot line edges
are removed from G

An orange vertex is
in a vertex cover.

$k = 2$

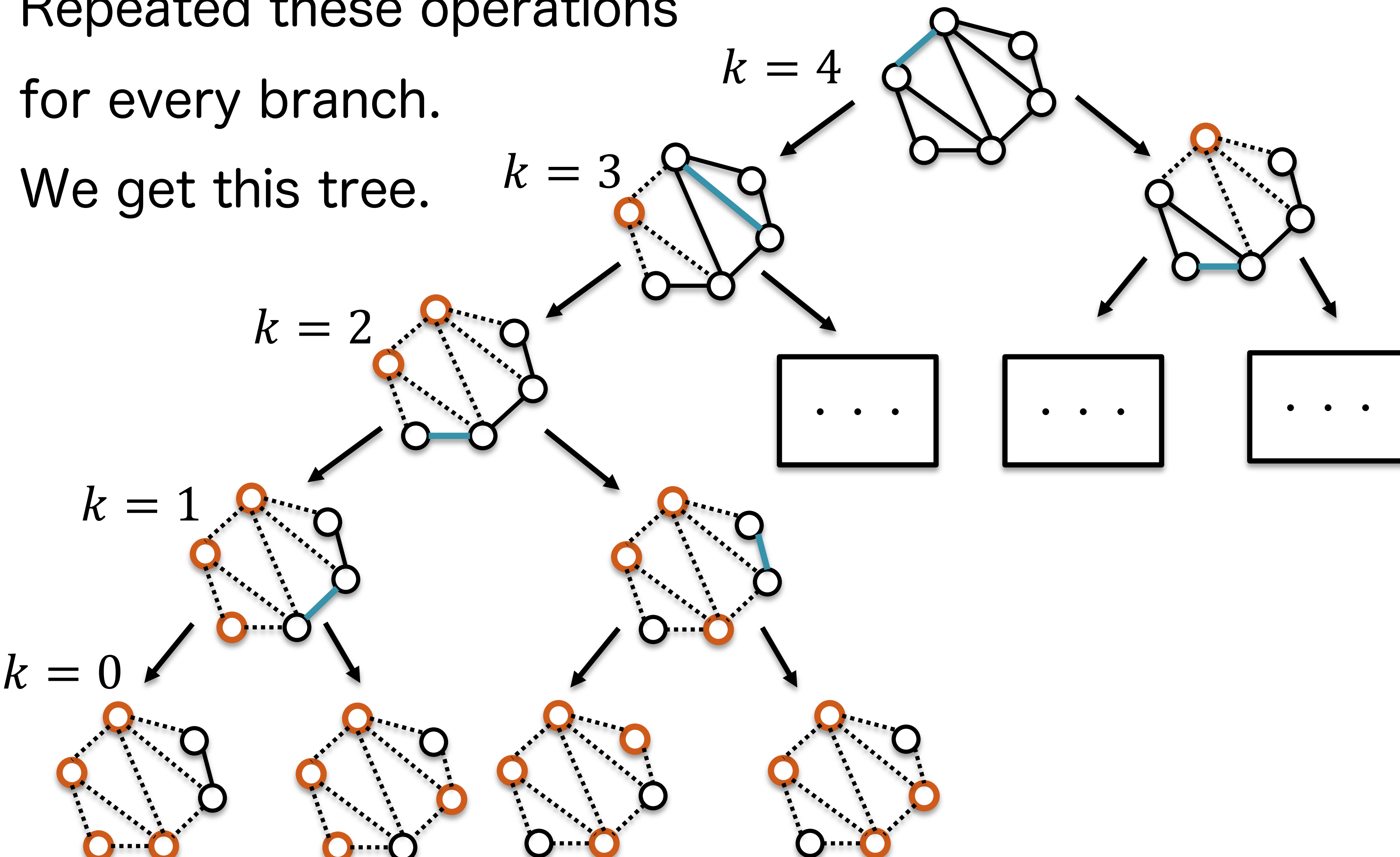


An example of algorithm's procedure

Repeated these operations

for every branch.

We get this tree.



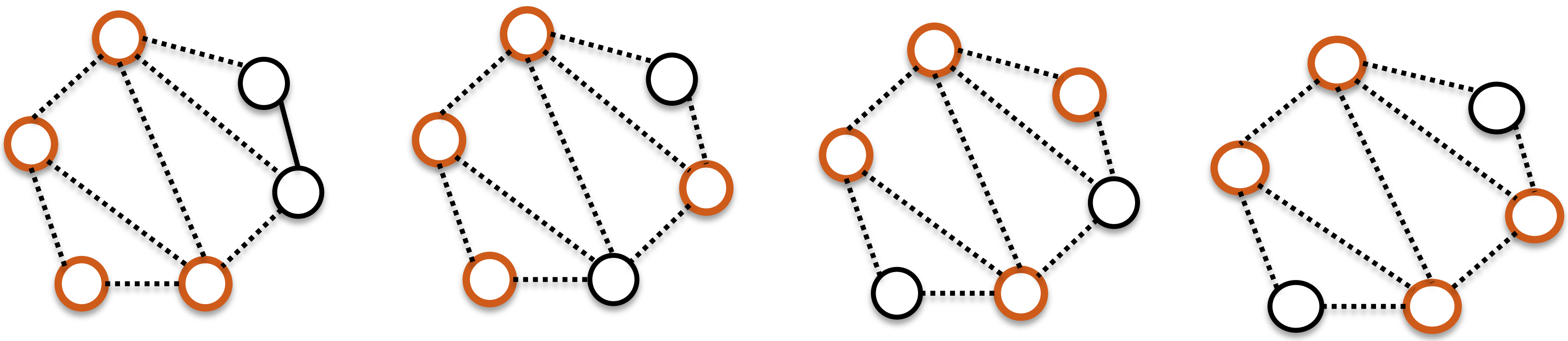
An example of algorithm's procedure

Now, we focus on graphs with $k = 0$.

See the following figures.

There exists at least one vertex cover, then algorithm output Yes for this input.

If there exists no vertex cover, the algorithm outputs No.



Not VC　　VC　　VC　　VC

An analysis of the running time

For every branch, we branches two cases and reduce k to $k-1$. Thus, the maximum number of branches is 2^k.

In addition, we remove edges for each branch, and it takes $O(n)$ time.

Hence, the total running time is $2^k \times O(n) = O(2^k n)$

The algorithm can finds a minimum vertex cover

We consider how to find a minimum vertex cover of a graph G.

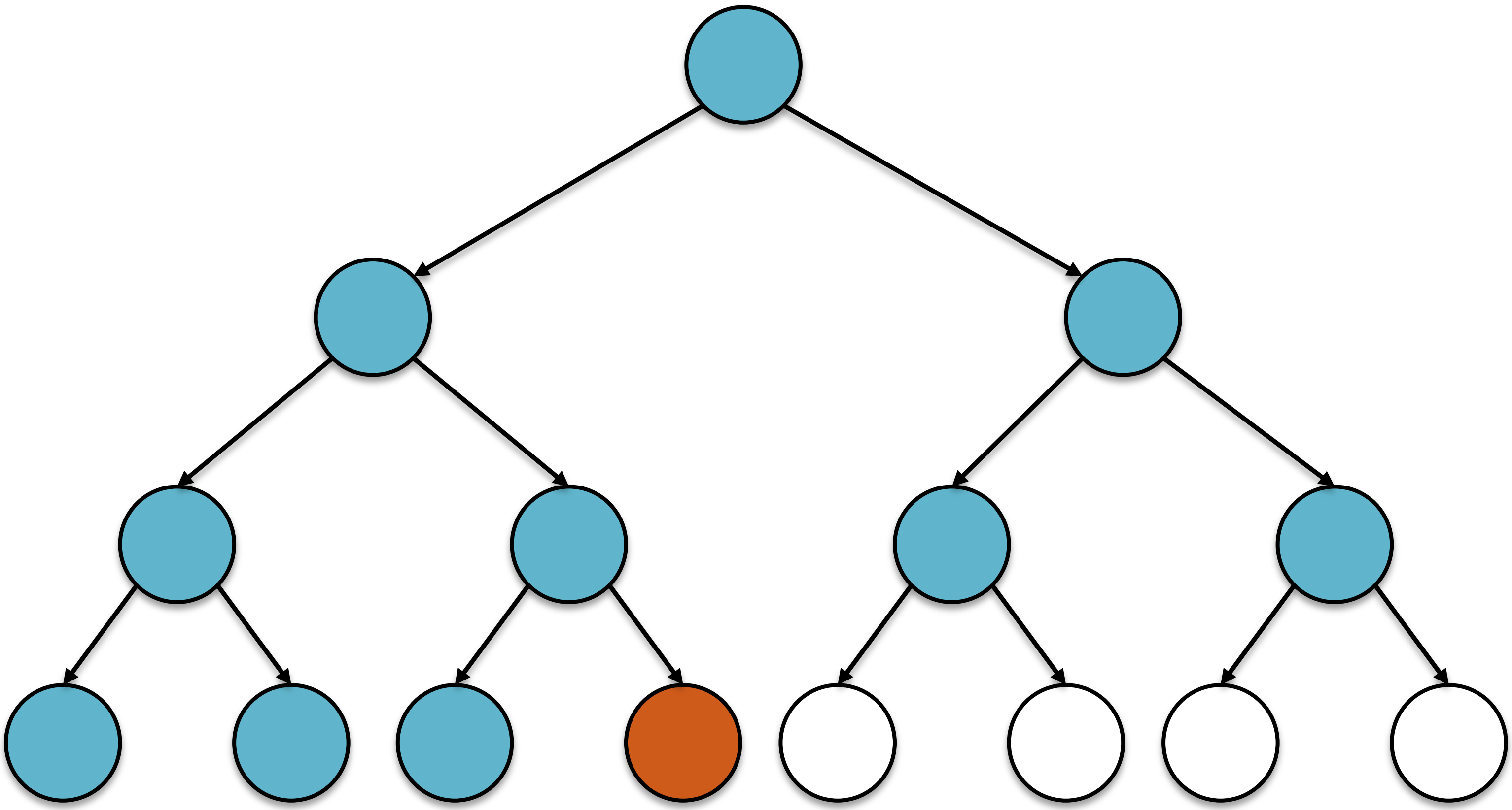
We can find it by using the previous algorithm.

For each level, the number of vertices included in a vertex cover is same.

Thus, if there exists a vertex cover of G at level ℓ, then the size of vertex cover is ℓ.

The way to find a minimum vertex cover is to find the minimum ℓ, that is, the level such that a vertex cover of G first appears. See the next slide.

The algorithm finds a minimum vertex cover



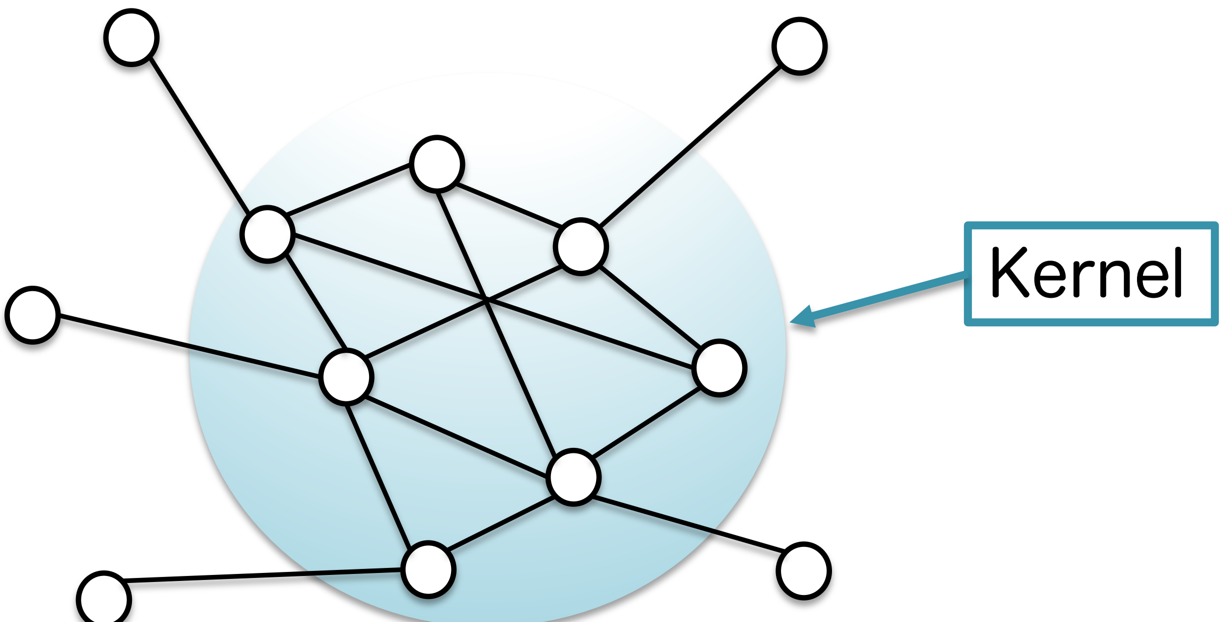
This is a branching tree.
Blued nodes are not vertex covers and an orange node is a vertex cover of G.
White nodes are unsearched.
In this case, level 3 is the first level that a vertex cover are found.
Thus, the size of a minimum vertex cover is 3.

Kernelization

Kernel

Intuitively speaking, "Intrinsically difficult points to solve"

- If we can solve kernel part of an input, its solution is to be almost solution of the input.



The definition of Kernelization

A kernelizaition is a kind of self-reduction

- Input (x, k)： x and k are an original instance and parameter, respectively
- Output (x', k') ：a kernelized instance
- Constraints
  - ✓ $|x'| = f(k)$, where $f(k)$ is some function of k
  - ✓ $k' = g(k)$, where g(k) is some function of k
  - ✓ A reduction is done in polynomial time of $|x| + k$

There exists a FPT algorithm for a problem P
if and only if P has a kernel. [Cai, Chen, Downey, and Fellows 1997]

Kernelization for Vertex Cover

A degree of v is the number of edges connected to the vertex v

Kernelization:

We do the following operations until there exist no more vertices with degree greater than k.

- We include v in a vertex cover U and remove v and the edges connected to v from G

Algorithm for VC with Kernelizaiton

Let G' be a graph that removed all vertices with degree greater than k from G.

Let k' be k – (the number of vertices included in U by kernelization)

BS'(G, k)

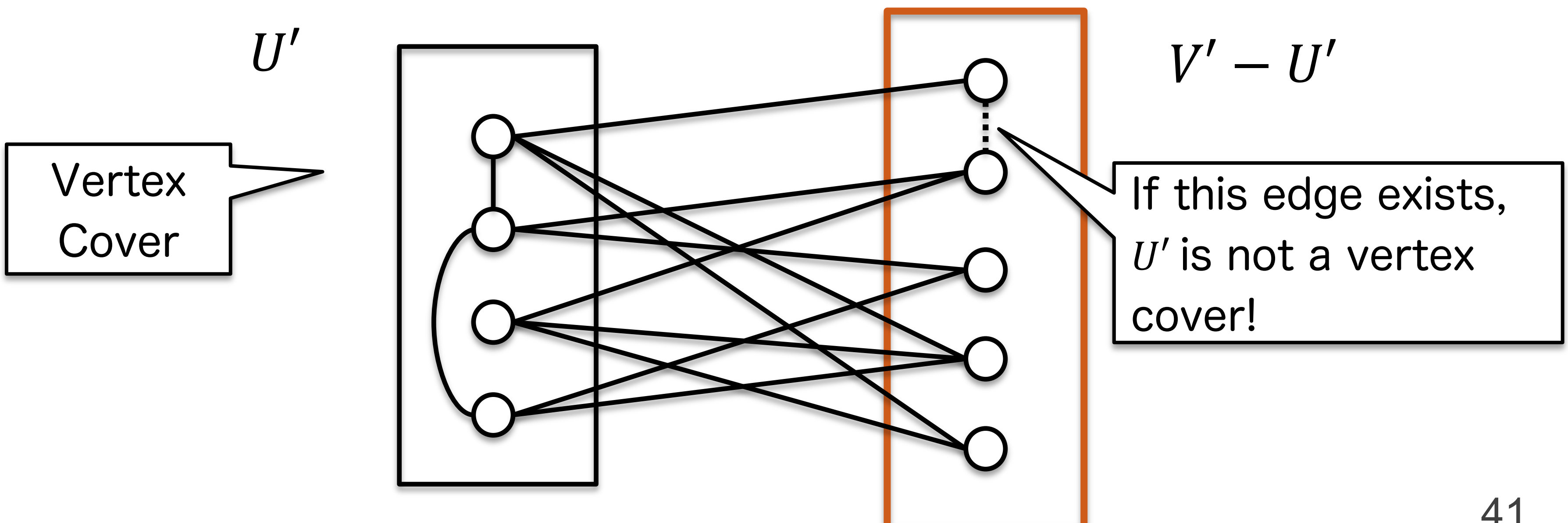
1. By kernelization, we get G' and k'
2. We run BS$(G'\ k')$, where BS is the FPT algorithm for VC explained former.

The correctness of algorithm

Let V' be the vertex set of G' and U' be a vertex cover of G.

Then, there exists no edge in any pair of vertices in $V' - U'$.

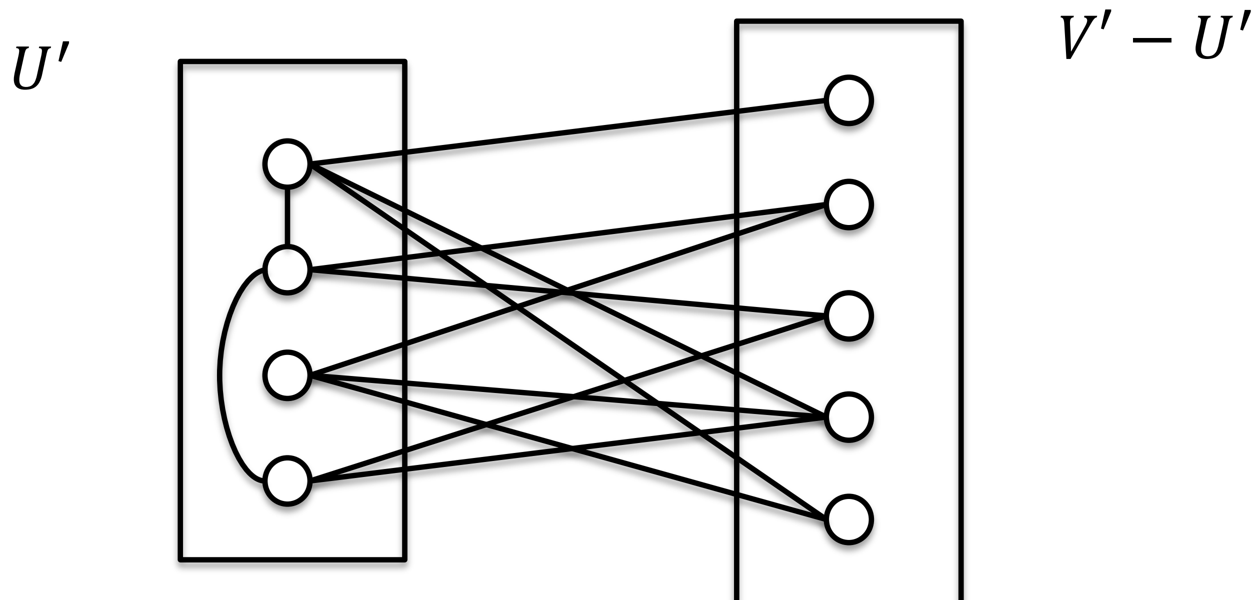
If there exists some edge, it violates U' is a vertex cover.



Estimation of the size of G'

For each vertex u in U', u' is connected to at most k vertices. Thus, $|V' - U'| \leq k|U|$ holds.

Hence, $|V'| = |U'| + |V' -' '| \leq (k+1)|U'| \leq k(k+1)$ holds.



Analysis of the running time

Kernelization is done in $O(n^2)$ time because the number of vertices is n and removing edges is done in $O(n)$ time.

In addition, the followings hold.

- $|V'| \leq k(k+1)$
- $k' \leq k$

As the running time of BS(n, k) is $O(2^k n)$, then the running time of BS$(|V'|, k')$ is $O\left(2^k k(k+1)\right) = O(k^2 2^k)$.

Thus, the overall running time is $O(k^2 2^k + n^2)$.

Summary

We introduce algorithms for the (Minimum) Vertex Cover

- 2-approximation algorithm for the Minimum Vetex Cover
- A simple FPT algorithm for the Vertex Cover
  - It can find minimum vertex covers.
- FPT algorithm for the Vertex Cover via Kernelization.

There exist a various kind of algorithms; if you are interested, we recommend you survey them.