

# Approximation Algorithms

## CSCI 432

$P$  {  $P$  is the set of problems that are solvable by an algorithm whose running time is polynomial time.

$P$  {  $P$  is the set of problems that are solvable by an algorithm whose running time is polynomial time.

How do you show a problem is in the set  $P$ ?

$P$  {  $P$  is the set of problems that are solvable by an algorithm whose running time is polynomial time.

How do you show a problem is in the set  $P$ ?  
Solve it in polynomial time.

$P$  {  $P$  is the set of problems that are solvable by an algorithm whose running time is polynomial time.

$NP$  { Set of problems that are verifiable in polynomial time.

# *SUBSET – SUM*

Claim:  $SUBSET - SUM = \{\langle S, t \rangle : S = \{x_1, \dots, x_n\}, \text{ and there exists some } \{y_1, \dots, y_m\} \subseteq S \text{ such that } \sum y_i = t\} \in NP$ .

# *SUBSET – SUM*

Claim:  $SUBSET - SUM = \{\langle S, t \rangle : S = \{x_1, \dots, x_n\}, \text{ and there exists some } \{y_1, \dots, y_m\} \subseteq S \text{ such that } \sum y_i = t\} \in NP$ .

Example:

$\langle \{4, 11, 16, 21, 27\}, 25 \rangle$

Is there a subset of  $\{4, 11, 16, 21, 27\}$  that sums to 25?

# *SUBSET – SUM*

Claim:  $SUBSET - SUM = \{\langle S, t \rangle : S = \{x_1, \dots, x_n\}, \text{ and there exists some } \{y_1, \dots, y_m\} \subseteq S \text{ such that } \sum y_i = t\} \in NP$ .

Example:

$\langle \{4, 11, 16, 21, 27\}, 25 \rangle$

Is there a subset of  $\{4, 11, 16, 21, 27\}$  that sums to 25?

Yes,  $4 + 21 = 25$ .



# *SUBSET – SUM*

Claim:  $SUBSET - SUM = \{\langle S, t \rangle : S = \{x_1, \dots, x_n\}, \text{ and there exists some } \{y_1, \dots, y_m\} \subseteq S \text{ such that } \sum y_i = t\} \in NP$ .

Example:

$\langle \{4, 11, 16, 21, 27\}, 25 \rangle$

Is there a subset of  $\{4, 11, 16, 21, 27\}$  that sums to 25?

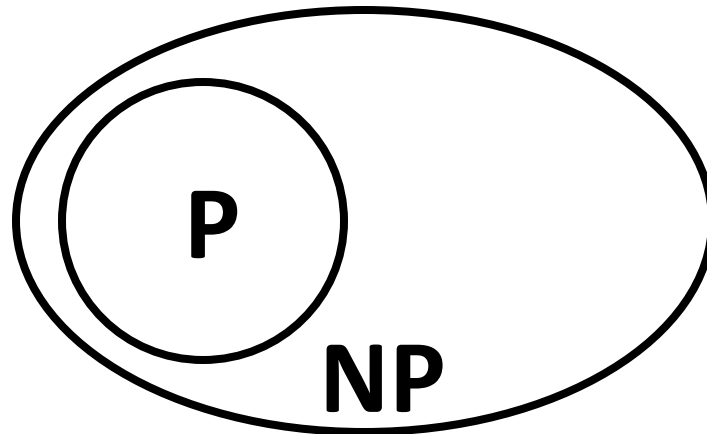
Yes,  $4 + 21 = 25$ .

How do we verify *SUBSET – SUM* answers?

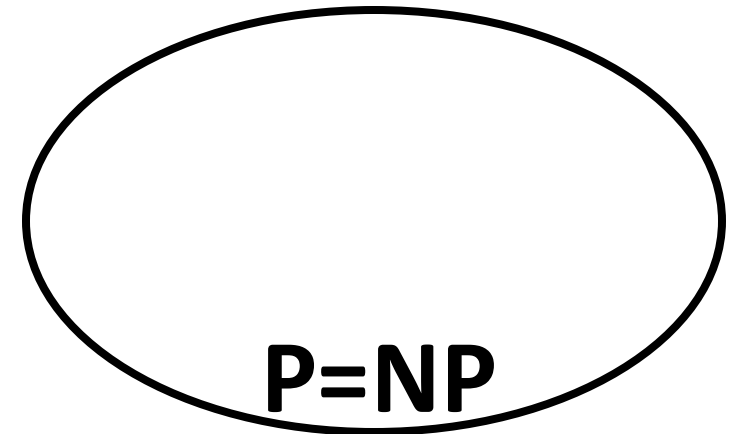
$P$  {  $P$  is the set of problems that are solvable by an algorithm whose running time is polynomial time.

$NP$  { Set of problems that are verifiable in polynomial time.

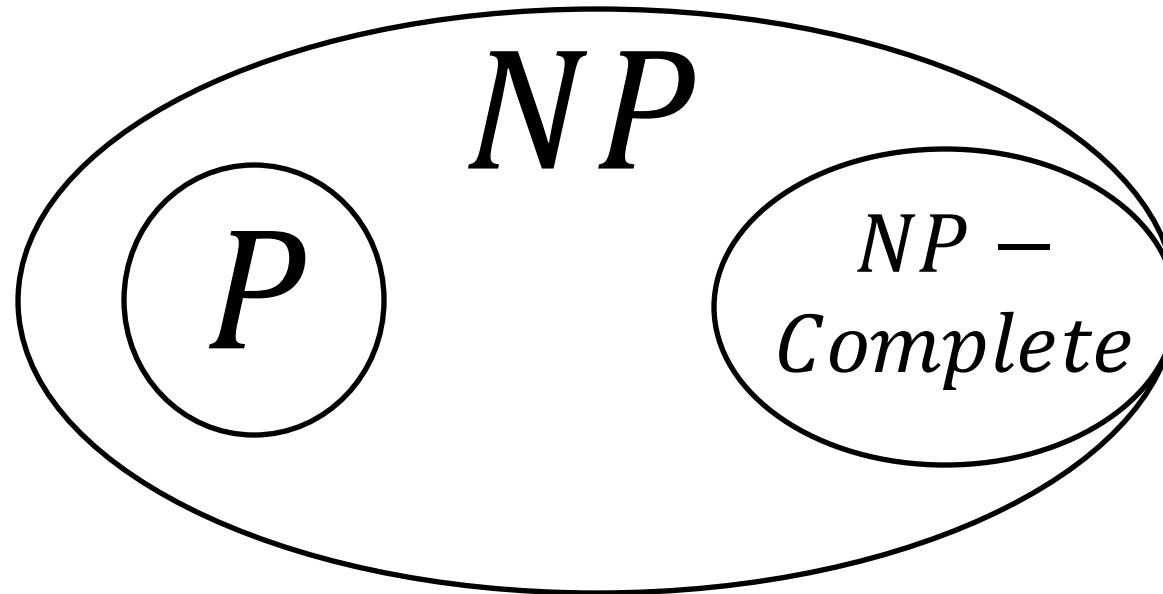
How does  $P$  relate to  $NP$ ?



Or



$NP$   
*Complete* { Set of problems in NP whose algorithms  
can solve any other problem in NP with  
polynomial extra time.



$NP$   
*Complete* { Set of problems in NP whose algorithms  
can solve any other problem in NP with  
polynomial extra time.

Interesting Properties:

- A polynomial time algorithm for **any**  $NP - Complete$  problem gives a polynomial time algorithm for **every** problem in NP (i.e.,  $P = NP$ )...

$NP$   
*Complete* { Set of problems in NP whose algorithms  
can solve any other problem in NP with  
polynomial extra time.

Interesting Properties:

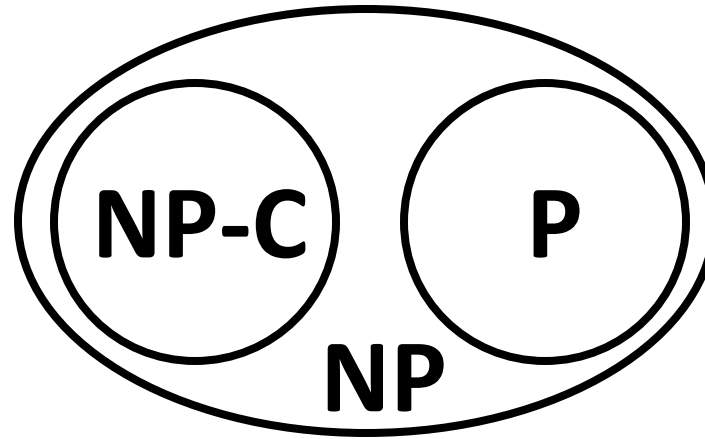
- A polynomial time algorithm for **any**  $NP - Complete$  problem gives a polynomial time algorithm for **every** problem in NP (i.e.,  $P = NP$ )...
- ... including all the other  $NP - Complete$  problems.

$NP$   
*Complete* { Set of problems in NP whose algorithms  
can solve any other problem in NP with  
polynomial extra time.

Interesting Properties:

- A polynomial time algorithm for **any**  $NP - Complete$  problem gives a polynomial time algorithm for **every** problem in NP (i.e.,  $P = NP$ )...
- ... including all the other  $NP - Complete$  problems.
- The thing that makes one NP-C problem (possibly) unsolvable in polynomial time is the exact same thing that makes every other NP-C problem (possibly) unsolvable in polynomial time.

# Handling NP-Completeness



Techniques to handle NP-Complete problems:

1. Brute Force (i.e. Exponential Time).
2. Heuristics.
3. Approximation Algorithms.
4. Fixed-parameter Tractable Algorithms.

# Approximation Algorithms

**For a minimization problem.**

The diagram illustrates the approximation ratio formula  $ALG \leq \alpha OPT$ . Three arrows point from descriptive text to the components of the formula: a curved arrow from the left points to  $ALG$ , a straight arrow from the bottom points to  $\alpha$ , and a curved arrow from the right points to  $OPT$ .

$$ALG \leq \alpha OPT$$

Cost (size) of algorithm's solution.

Approximation Ratio

Cost (size) of optimal solution.



# Approximation Algorithms

For a minimization problem.

$$\text{ALG} \leq \alpha \text{ OPT}$$

Cost (size) of algorithm's solution.

Approximation Ratio

Cost (size) of optimal solution.

Example: If my CheapestPizzaInBozeman algorithm is a 1.25-approximation algorithm, the cost of the pizza it finds is at most 1.25 times the optimal cost.

# Approximation Algorithms

For a minimization problem.

The diagram illustrates the approximation ratio formula  $ALG \leq \alpha OPT$ . Three arrows point from the text labels below to the components of the formula: one from 'Cost (size) of algorithm's solution.' to 'ALG', one from 'Approximation Ratio' to ' $\alpha$ ', and one from 'Cost (size) of optimal solution.' to 'OPT'.

$$ALG \leq \alpha OPT$$

Cost (size) of algorithm's solution.      Approximation Ratio      Cost (size) of optimal solution.

Example: If my CheapestPizzaInBozeman algorithm is a 1.25-approximation algorithm, the cost of the pizza it finds is at most 1.25 times the optimal cost.

I.e. If the cheapest pizza in Bozeman is \$2.00/slice, CheapestPizzaInBozeman will find pizza that is at most \$2.50/slice.

# Approximation Algorithms

For a minimization problem.

Cost (size) of algorithm's solution.

Approximation Ratio

Cost (size) of optimal solution.

$$\text{ALG} \leq \alpha \text{ OPT}$$

Example: If my CheapestPizzaInBozeman algorithm is a 1.25-approximation algorithm, the cost of the pizza it finds is at most 1.25 times the optimal cost.

I.e. If the cheapest pizza in Bozeman is \$2.00/slice, CheapestPizzaInBozeman will find pizza that is at most \$2.50/slice.

Note: if problem is a maximization problem,  $\text{ALG} \geq \frac{1}{\alpha} \text{ OPT}$

# Approximation Algorithms

**For a minimization problem.**

The diagram shows the inequality  $ALG \leq \alpha OPT$  in the center. Three arrows point towards it from below: one from the left pointing to 'ALG', one from the bottom pointing to ' $\alpha$ ', and one from the right pointing to 'OPT'.

Cost (size) of algorithm's solution.      Approximation Ratio      Cost (size) of optimal solution.

Example:

# Approximation Algorithms

**For a minimization problem.**

The diagram illustrates the inequality  $ALG \leq \alpha OPT$ . It features three labels with arrows pointing to the corresponding parts of the equation: "Cost (size) of algorithm's solution." points to "ALG", "Approximation Ratio" points to " $\alpha$ ", and "Cost (size) of optimal solution." points to "OPT".

$$ALG \leq \alpha OPT$$

Cost (size) of algorithm's solution.      Approximation Ratio      Cost (size) of optimal solution.

Example:

- Suppose I know my algorithm is a 1.12-approximation algorithm.

# Approximation Algorithms

For a minimization problem.

The diagram shows the inequality  $ALG \leq \alpha OPT$ . An arrow points from the text "Cost (size) of algorithm's solution." to the term  $ALG$ . Another arrow points from the text "Cost (size) of optimal solution." to the term  $OPT$ . A third arrow points from the text "Approximation Ratio" to the symbol  $\alpha$ .

$$ALG \leq \alpha OPT$$

Cost (size) of algorithm's solution.      Approximation Ratio      Cost (size) of optimal solution.

Example:

- Suppose I know my algorithm is a 1.12-approximation algorithm.
- Suppose my algorithm returns a solution of cost (size) 746.125.

# Approximation Algorithms

For a minimization problem.

The diagram shows the inequality  $ALG \leq \alpha OPT$ . An arrow points from the text "Cost (size) of algorithm's solution." to "ALG". Another arrow points from the text "Cost (size) of optimal solution." to "OPT". A third arrow points from the text "Approximation Ratio" to the symbol  $\alpha$ .

$$ALG \leq \alpha OPT$$

Cost (size) of algorithm's solution.      Approximation Ratio      Cost (size) of optimal solution.

Example:

- Suppose I know my algorithm is a 1.12-approximation algorithm.
- Suppose my algorithm returns a solution of cost (size) 746.125.

Then, I know that  $746.125 \leq 1.12 OPT$

# Approximation Algorithms

For a minimization problem.

Cost (size) of algorithm's solution.      Approximation Ratio      Cost (size) of optimal solution.

$$\text{ALG} \leq \alpha \text{ OPT}$$

Example:

- Suppose I know my algorithm is a 1.12-approximation algorithm.
- Suppose my algorithm returns a solution of cost (size) 746.125.

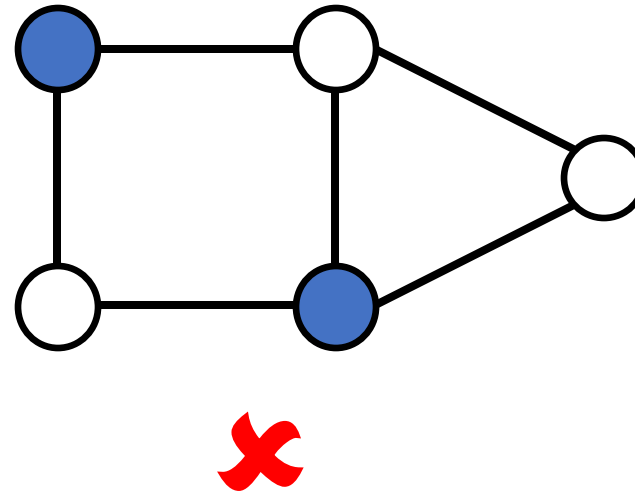
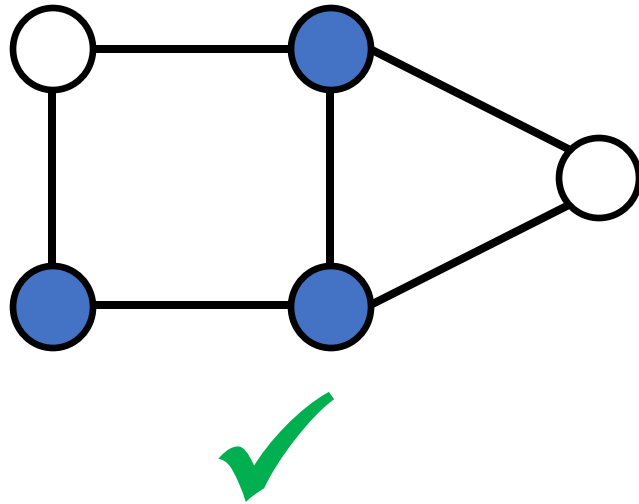
Then, I know that  $746.125 \leq 1.12 \text{ OPT}$

$$\Rightarrow \frac{746.125}{1.12} = 666.183 \leq \text{OPT} \leq 746.125$$



# Vertex Cover

Vertex Cover: Given graph, find the smallest subset of vertices such that every edge in the graph has at least one vertex in the subset.



# Vertex Cover

Vertex Cover: Given graph, find the smallest subset of vertices such that every edge in the graph has at least one vertex in the subset.

Algorithm:

?

# Vertex Cover

Vertex Cover: Given graph, find the smallest subset of vertices such that every edge in the graph has at least one vertex in the subset.

Algorithm:

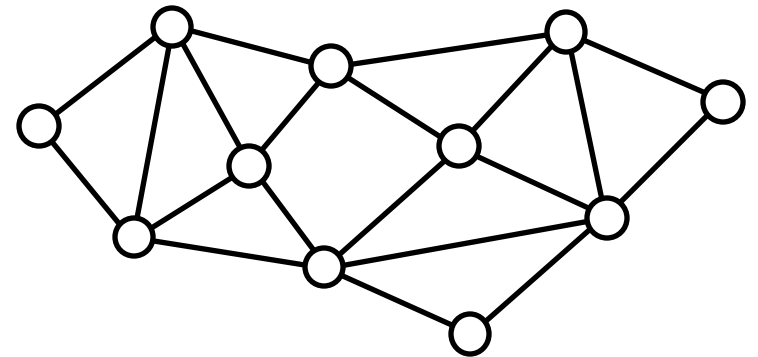
```
while uncovered edge exists
    select both vertices from uncovered edge
```

# Vertex Cover

Vertex Cover: Given graph, find the smallest subset of vertices such that every edge in the graph has at least one vertex in the subset.

Algorithm:

```
while uncovered edge exists  
    select both vertices from uncovered edge
```

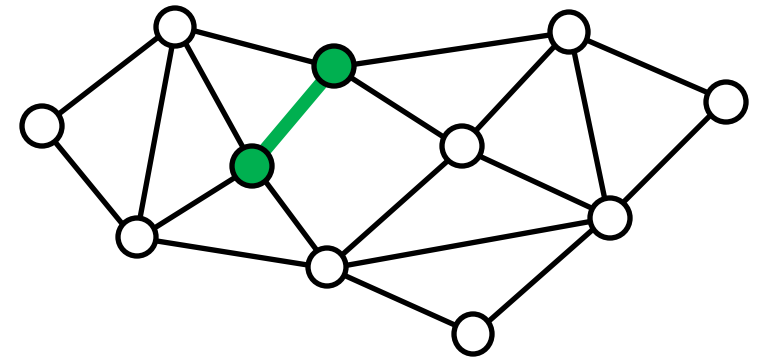


# Vertex Cover

Vertex Cover: Given graph, find the smallest subset of vertices such that every edge in the graph has at least one vertex in the subset.

Algorithm:

```
while uncovered edge exists  
    select both vertices from uncovered edge
```

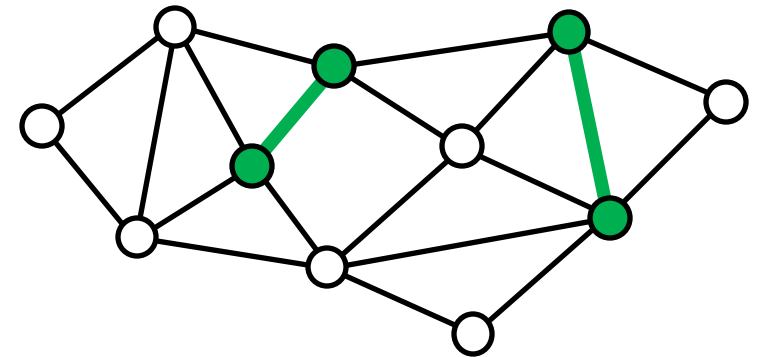


# Vertex Cover

Vertex Cover: Given graph, find the smallest subset of vertices such that every edge in the graph has at least one vertex in the subset.

Algorithm:

```
while uncovered edge exists  
    select both vertices from uncovered edge
```

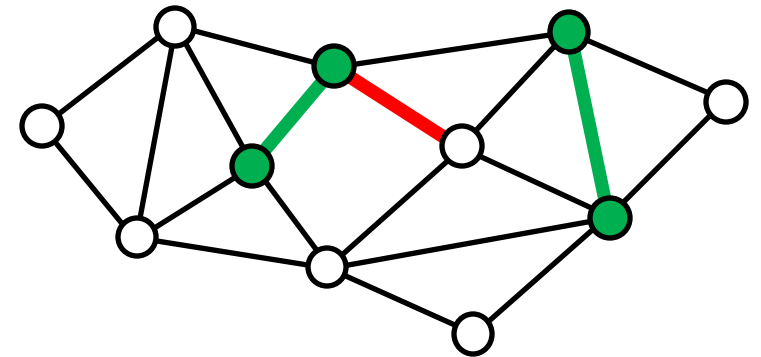


# Vertex Cover

Vertex Cover: Given graph, find the smallest subset of vertices such that every edge in the graph has at least one vertex in the subset.

Algorithm:

```
while uncovered edge exists  
    select both vertices from uncovered edge
```

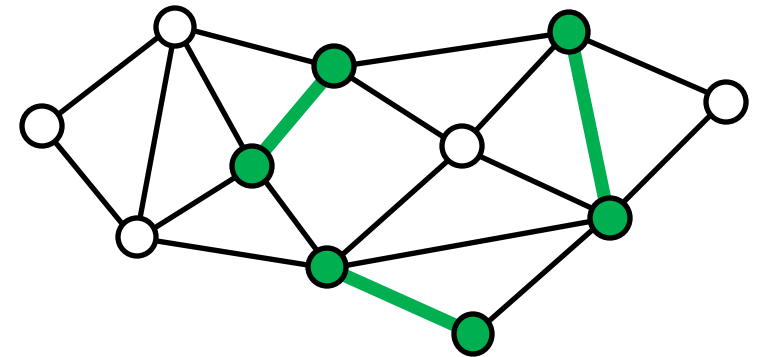


# Vertex Cover

Vertex Cover: Given graph, find the smallest subset of vertices such that every edge in the graph has at least one vertex in the subset.

Algorithm:

```
while uncovered edge exists  
    select both vertices from uncovered edge
```



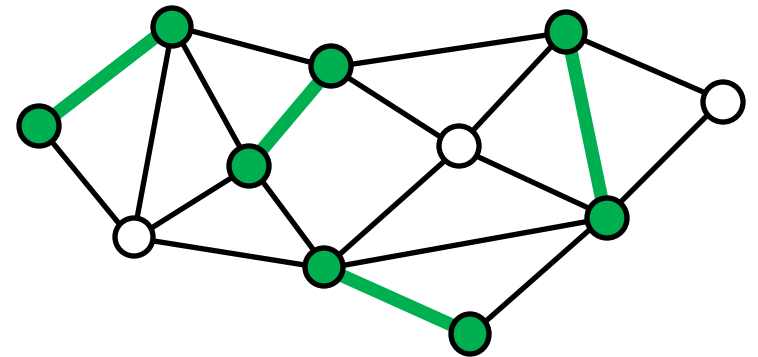


# Vertex Cover

Vertex Cover: Given graph, find the smallest subset of vertices such that every edge in the graph has at least one vertex in the subset.

Algorithm:

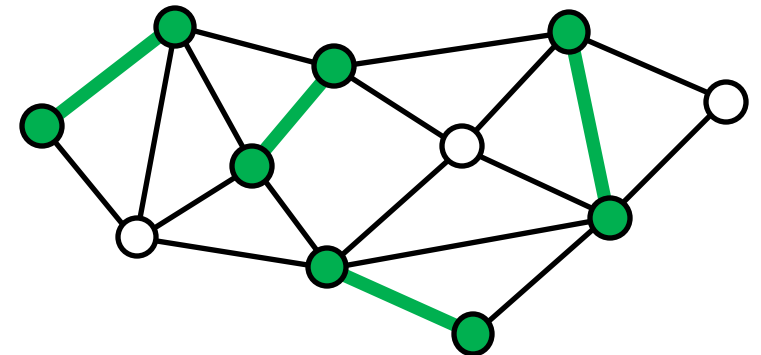
```
while uncovered edge exists  
    select both vertices from uncovered edge
```



# Vertex Cover

```
while uncovered edge exists  
    select both vertices from uncovered edge
```

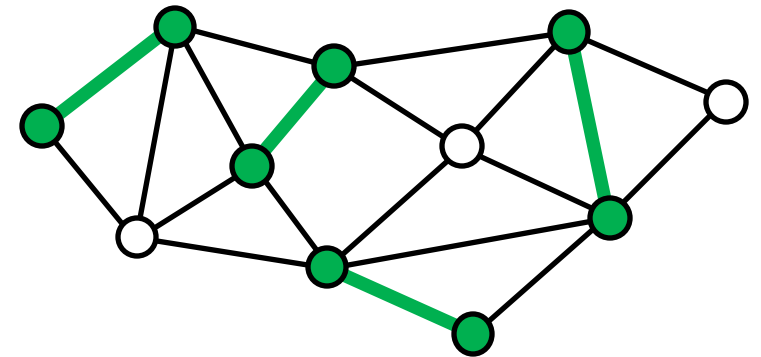
Consider a set of edges,  $E' \subset E$ , that do not share vertices.



# Vertex Cover

```
while uncovered edge exists  
    select both vertices from uncovered edge
```

Consider a set of edges,  $E' \subset E$ , that do not share vertices. Is there a relationship between the minimum vertex cover and  $|E'|$ ?

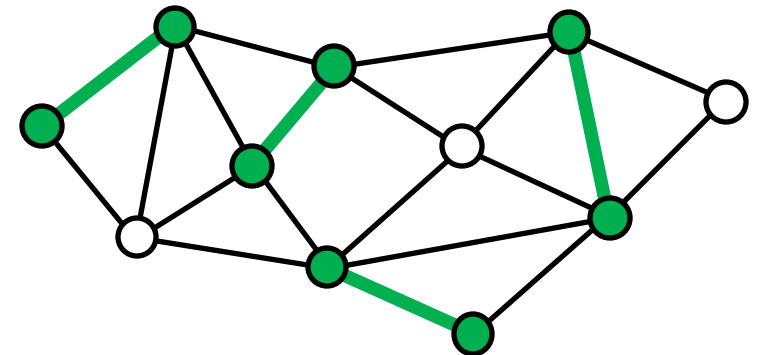


# Vertex Cover

```
while uncovered edge exists  
    select both vertices from uncovered edge
```

Consider a set of edges,  $E' \subset E$ , that do not share vertices. Is there a relationship between the minimum vertex cover and  $|E'|$ ?

$|E'| \leq \text{OPT}$  ← Size of actual smallest vertex cover.



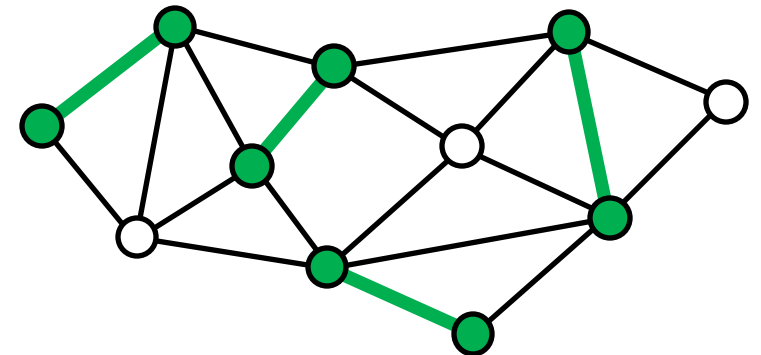
# Vertex Cover

```
while uncovered edge exists  
    select both vertices from uncovered edge
```

Consider a set of edges,  $E' \subset E$ , that do not share vertices. Is there a relationship between the minimum vertex cover and  $|E'|$ ?

$|E'| \leq \text{OPT}$  ← Size of actual smallest vertex cover.

If we selected fewer than one vertex per edge, we would not have a vertex cover, because that edge would not be covered!



# Vertex Cover

```
while uncovered edge exists  
    select both vertices from uncovered edge
```

Consider a set of edges,  $E' \subset E$ , that do not share vertices. Is there a relationship between the minimum vertex cover and  $|E'|$ ?

$$|E'| \leq \text{OPT}$$

Does the size of the algorithm's output relate to a set of edges that do not share vertices?

# Vertex Cover

```
while uncovered edge exists  
    select both vertices from uncovered edge
```

Consider a set of edges,  $E' \subset E$ , that do not share vertices. Is there a relationship between the minimum vertex cover and  $|E'|$ ?

$$|E'| \leq \text{OPT}$$

Does the size of the algorithm's output relate to a set of edges that do not share vertices?

$$\text{ALG} = 2 |E'|$$

# Vertex Cover

```
while uncovered edge exists  
    select both vertices from uncovered edge
```

Consider a set of edges,  $E' \subset E$ , that do not share vertices. Is there a relationship between the minimum vertex cover and  $|E'|$ ?

$$|E'| \leq \text{OPT}$$

Does the size of the algorithm's output relate to a set of edges that do not share vertices?

$$\text{ALG} = 2 |E'|$$

$$\Rightarrow \text{ALG} = 2 |E'|$$



# Vertex Cover

```
while uncovered edge exists  
    select both vertices from uncovered edge
```

Consider a set of edges,  $E' \subset E$ , that do not share vertices. Is there a relationship between the minimum vertex cover and  $|E'|$ ?

$$|E'| \leq \text{OPT}$$

Does the size of the algorithm's output relate to a set of edges that do not share vertices?

$$\text{ALG} = 2 |E'|$$

$$\Rightarrow \text{ALG} = 2 |E'| \leq 2 \text{ OPT}$$

# Vertex Cover

```
while uncovered edge exists  
    select both vertices from uncovered edge
```

Consider a set of edges,  $E' \subset E$ , that do not share vertices. Is there a relationship between the minimum vertex cover and  $|E'|$ ?

$$|E'| \leq \text{OPT}$$

Does the size of the algorithm's output relate to a set of edges that do not share vertices?

$$\text{ALG} = 2 |E'|$$

$$\Rightarrow \text{ALG} = 2 |E'| \leq 2 \text{ OPT} \Rightarrow \text{ALG} \leq 2 \text{ OPT}$$

# Vertex Cover

```
while uncovered edge exists
    select both vertices from uncovered edge
```

Consider a set of edges  $E' \subseteq E$  that do not share vertices. Is there a relation between  $|E'|$  and the size of an optimal vertex cover?

Does the greedy algorithm find an optimal vertex cover?  
do not share vertices?

We cannot find optimal vertex covers in poly time unless  $P = NP$ , but this algorithm is at worst 2-times optimal.

$$\text{ALG} = 2 |E'|$$

$$\Rightarrow \text{ALG} = 2 |E'| \leq 2 \text{ OPT} \Rightarrow \text{ALG} \leq 2 \text{ OPT}$$

# Vertex Cover

```
while uncovered edge exists  
    select both vertices from uncovered edge
```

$$\Rightarrow \text{ALG} \leq 2 \text{ OPT}$$

Is this the best this algorithm can do?

# Vertex Cover

```
while uncovered edge exists  
    select both vertices from uncovered edge
```

$$\Rightarrow \text{ALG} \leq 2 \text{ OPT}$$

Is this the best this algorithm can do?

I.e. Can we guarantee this algorithm does better than 2 OPT?

# Vertex Cover

```
while uncovered edge exists  
    select both vertices from uncovered edge
```

$$\Rightarrow \text{ALG} \leq 2 \text{ OPT}$$

Is this the best this algorithm can do?

I.e. Can we guarantee this algorithm does better than 2 OPT?

Is there a graph where this algorithm does exactly 2 OPT?

# Vertex Cover

```
while uncovered edge exists  
    select both vertices from uncovered edge
```

$$\Rightarrow \text{ALG} \leq 2 \text{ OPT}$$

Is this the best this algorithm can do?

I.e. Can we guarantee this algorithm does better than 2 OPT?

Is there a graph where this algorithm does exactly 2 OPT?



ALG



OPT

# Vertex Cover

```
while uncovered edge exists  
    select both vertices from uncovered edge
```

$$\Rightarrow \text{ALG} \leq 2 \text{ OPT}$$

Is this the best this algorithm can do?

I.e. Can we guarantee this algorithm does better than  $2 \text{ OPT}$ ?

Is there a graph where this algorithm does exactly  $2 \text{ OPT}$ ?

of arbitrary size



ALG



OPT



# Vertex Cover

**while** uncovered edge exists  
    select both vertices from uncovered edge

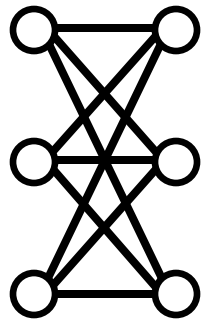
$$\Rightarrow \text{ALG} \leq 2 \text{ OPT}$$

Is this the best this algorithm can do?

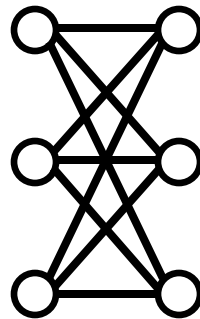
I.e. Can we guarantee this algorithm does better than 2 OPT?

Is there a graph where this algorithm does exactly 2 OPT?

Complete  
Bipartite Graph



ALG



OPT

ALG?

OPT?

# Vertex Cover

**while** uncovered edge exists  
    select both vertices from uncovered edge

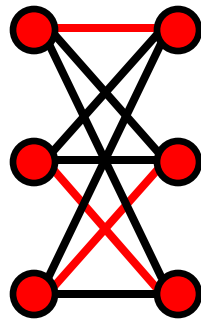
$$\Rightarrow \text{ALG} \leq 2 \text{ OPT}$$

Is this the best this algorithm can do?

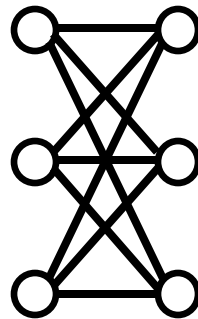
I.e. Can we guarantee this algorithm does better than 2 OPT?

Is there a graph where this algorithm does exactly 2 OPT?

Complete  
Bipartite Graph



ALG



OPT

$|\text{ALG}| = n$ : If  $v$  is not selected, all neighbors are  $\Rightarrow \frac{n}{2}$  edges are selected  $\Rightarrow$  all  $n$  vertices are selected.

# Vertex Cover

**while** uncovered edge exists  
    select both vertices from uncovered edge

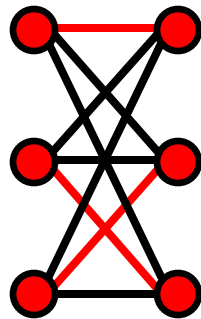
$$\Rightarrow \text{ALG} \leq 2 \text{ OPT}$$

Is this the best this algorithm can do?

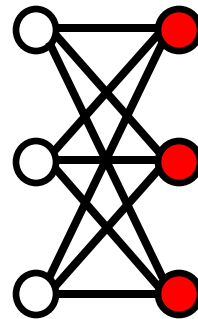
I.e. Can we guarantee this algorithm does better than  $2 \text{ OPT}$ ?

Is there a graph where this algorithm does exactly  $2 \text{ OPT}$ ?

Complete  
Bipartite Graph



ALG



OPT

$|\text{ALG}| = n$ : If  $v$  is not selected, all neighbors are  $\Rightarrow \frac{n}{2}$  edges are selected  $\Rightarrow$  all  $n$  vertices are selected.

$|\text{OPT}| = \frac{n}{2}$ : Fewer than  $\frac{n}{2}$  nodes selected  $\Rightarrow$   $\exists$  unselected edge.

# Vertex Cover

```
while uncovered edge exists  
    select both vertices from uncovered edge
```

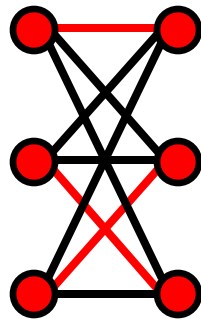
$$\Rightarrow \text{ALG} \leq 2 \text{ OPT}$$

Is this the best this algorithm can do?

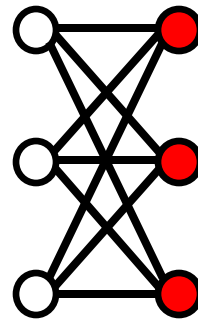
I.e. Can we guarantee this algorithm does better than 2 OPT?

Is there a graph where this algorithm does exactly 2 OPT?

Complete  
Bipartite Graph



ALG



OPT

$\therefore$  The best Vertex Cover  
can be approximated is  
within a factor of 2

# Vertex Cover

**while** uncovered edge exists  
    select both vertices from uncovered edge

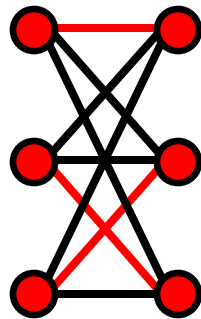
$$\Rightarrow \text{ALG} \leq 2 \text{ OPT}$$

Is this the best this algorithm can do?

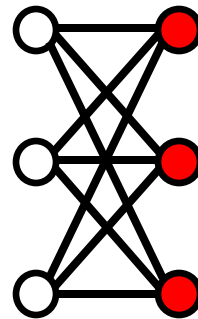
I.e. Can we guarantee this algorithm does better than 2 OPT?

Is there a graph where this algorithm does exactly 2 OPT?

Complete  
Bipartite Graph



ALG



OPT

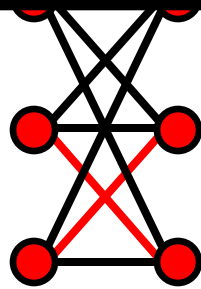
~~$\therefore$  The best Vertex Cover  
can be approximated is  
within a factor of 2~~

# Vertex Cover

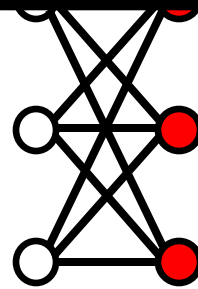
while uncovered edge exists  
select both vertices from uncovered edge

Vertex Cover is approximable within the  
bound  $2 - \frac{\log \log |V|}{2 \log |V|}$  and inapproximable  
within the bound 1.3606.

Complete  
Bipartite Graph

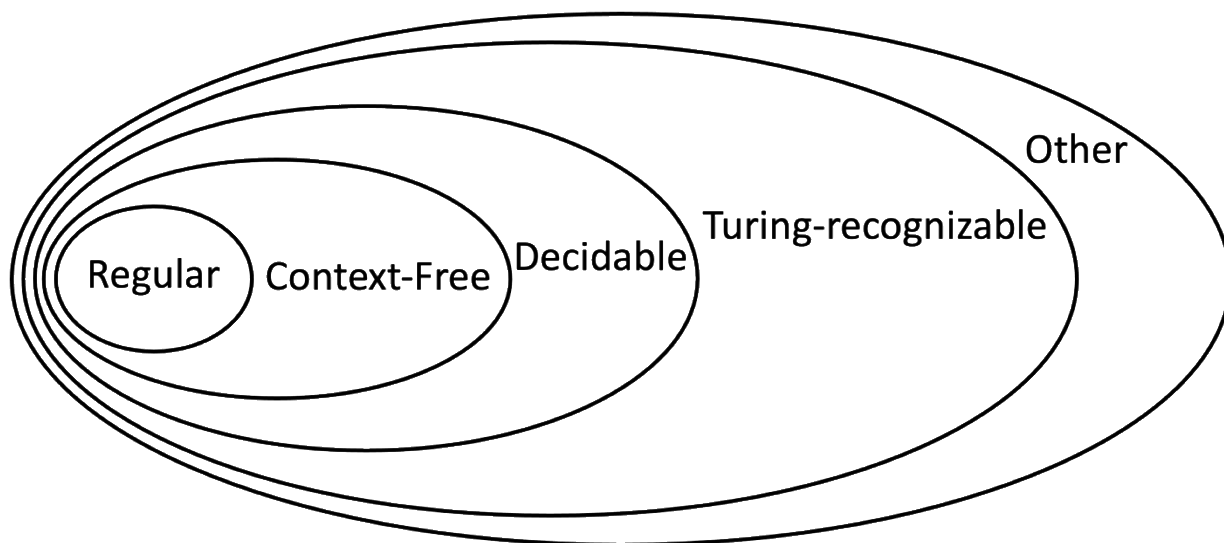


ALG

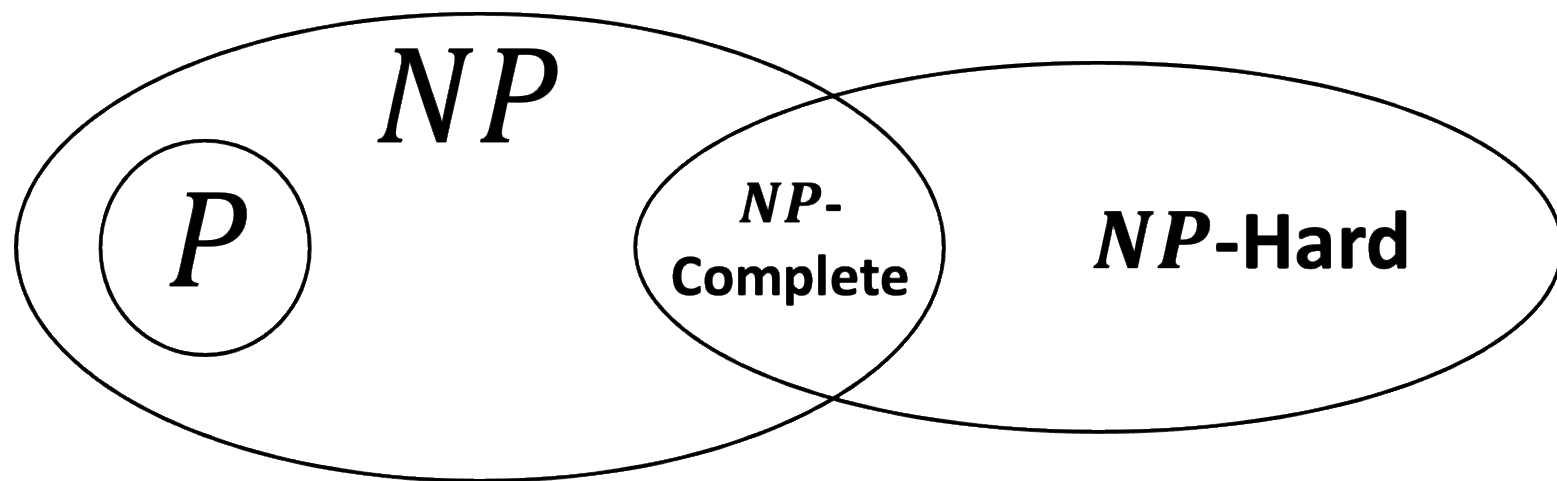


OPT

~~$\therefore$  The best Vertex Cover  
can be approximated is  
within a factor of 2~~

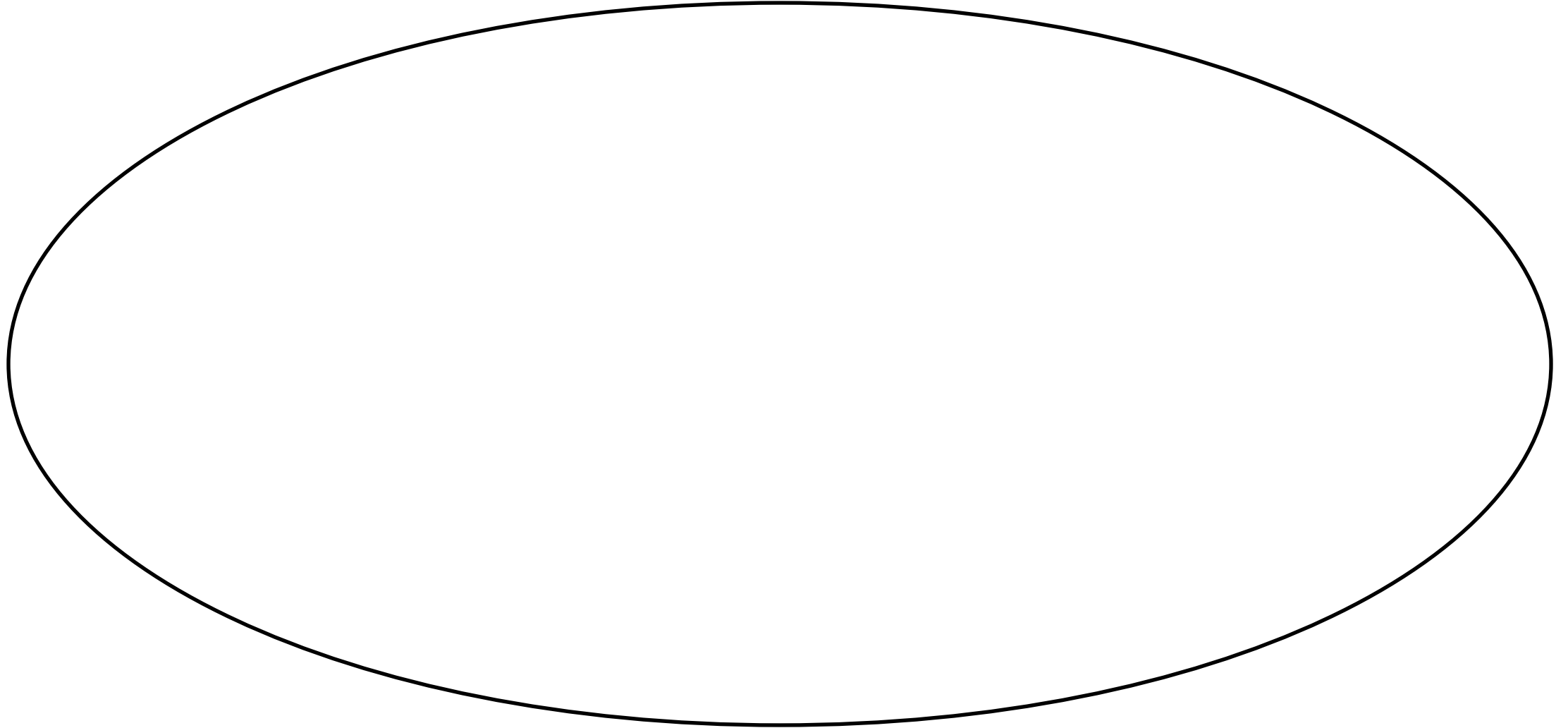


**Computability Hierarchy**



**Complexity Hierarchy**

# Approximability Hierarchy



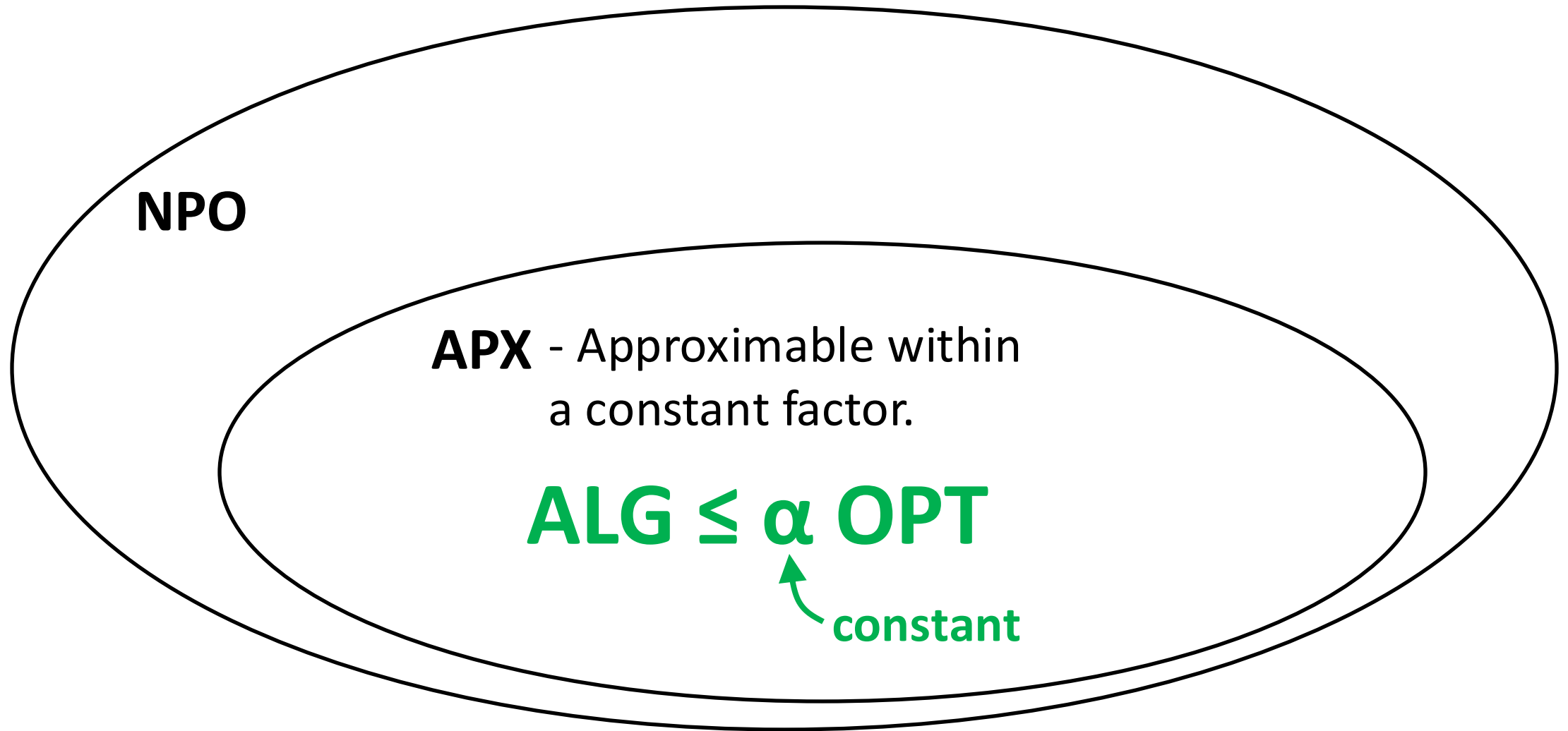


# Approximability Hierarchy



**NPO** - Optimization versions of problems in NP.

# Approximability Hierarchy



# Approximability Hierarchy

