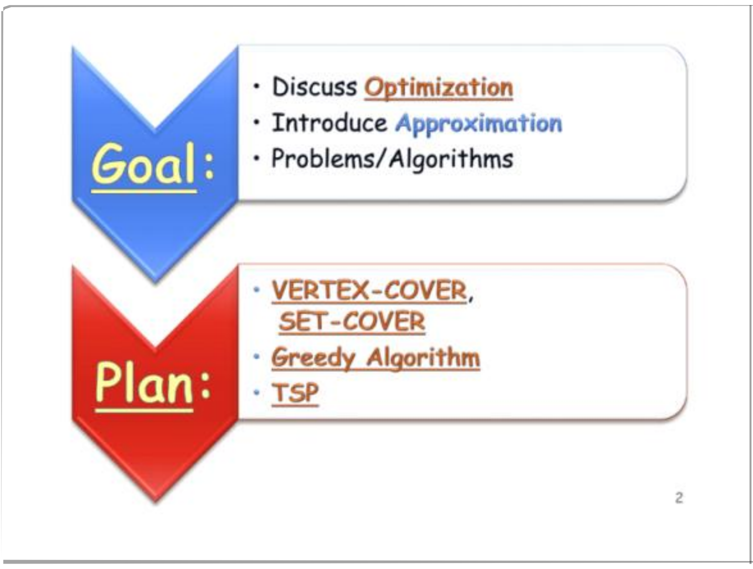
Approximation Problems Notes

Approximation Problems

In this lecture we study *approximation* versions for *optimization* problems whose exact solution is NP hard.


Goal:

- Discuss Optimization
- Introduce Approximation
- Problems/Algorithms

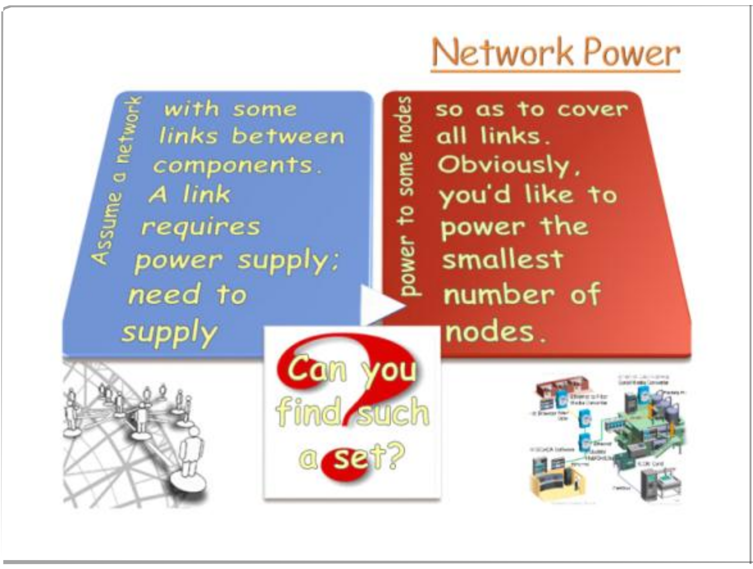
Plan:

- VERTEX-COVER, SET-COVER
- Greedy Algorithm
- TSP

2

We'll define optimization *parameters* and what it means to approximate them, via several concrete problems.

Vertex Cover (1)
Vertex Cover (2)
Set Cover
TSP



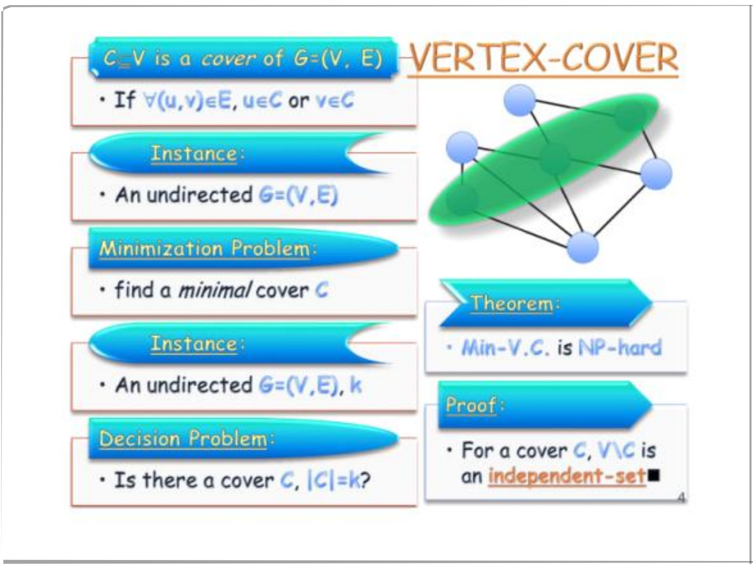
Network Power

Assume a network with some links between components. A link requires power supply; need to supply power to some nodes so as to cover all links. Obviously, you'd like to power the smallest number of nodes.

Can you find such a set?

Consider a wired network, with various locations and some wiring. One needs to find a set of locations so that every wire has one of its ends covered.

Naturally, one looks for the smallest possible set of locations.



VERTEX-COVER

$C \subseteq V$ is a *cover* of $G=(V, E)$

- If $\forall(u,v)\in E$, $u\in C$ or $v\in C$

Instance:

- An undirected $G=(V,E)$

Minimization Problem:

- find a *minimal* cover C

Instance:

- An undirected $G=(V,E)$, k

Decision Problem:

- Is there a cover C, $|C|=k$?

Theorem:

- Min-V.C. is NP-hard

Proof:

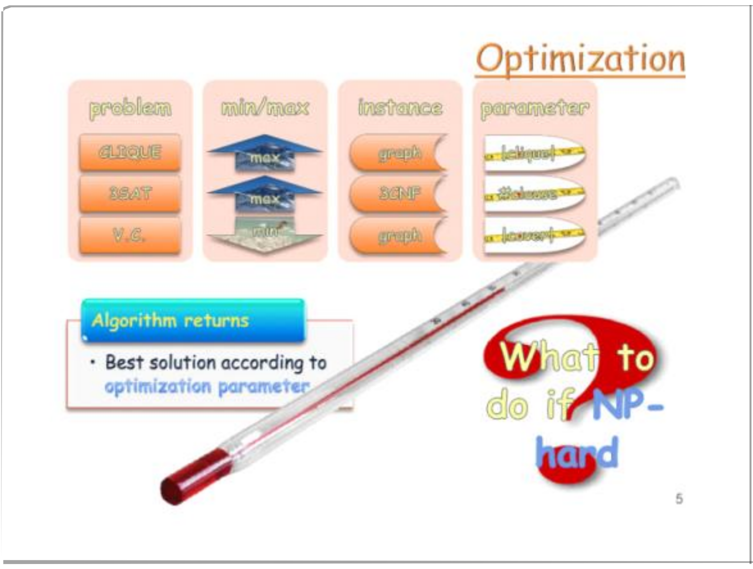
- For a cover C, $V\setminus C$ is an independent-set ■

4

This problem is the *Vertex Cover*: Given a graph, find a smallest set of nodes that covers all edges.

We can define the corresponding decision problem, where the question is whether there is such a cover whose size does not exceed some given threshold k.

This problem is clearly NP-hard: the complement of the Vertex Cover is an Independent Set, hence an exact solution to one implies an exact solution to the other.



Optimization

problem	min/max	instance	parameter
CLIQUE	max	graph	|clique|
3SAT	max	3CNF	#clauses
V.C.	min	graph	|cover|

Algorithm returns

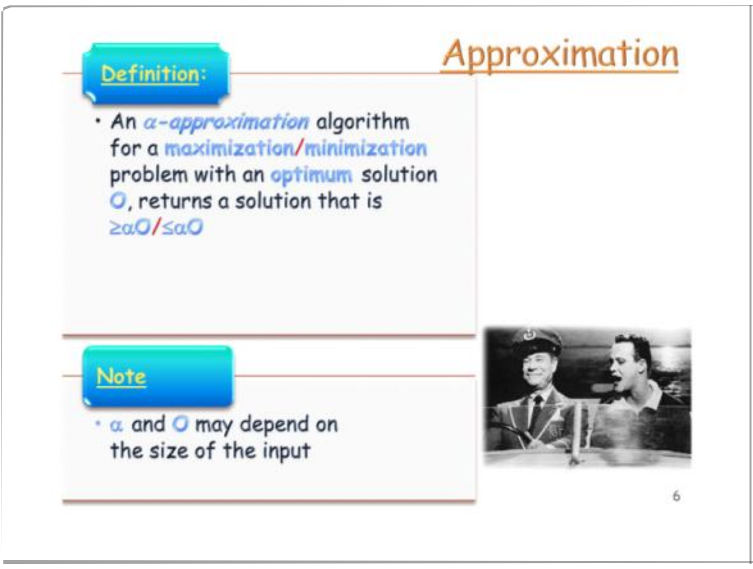
- Best solution according to optimization parameter

What to do if NP-hard

5

Let us recall some of the optimization problems we've discussed, both minimization and maximization, and identify the parameter optimized.

Now, what do we do if it turns out that the problem is NP-hard?



Approximation

Definition:

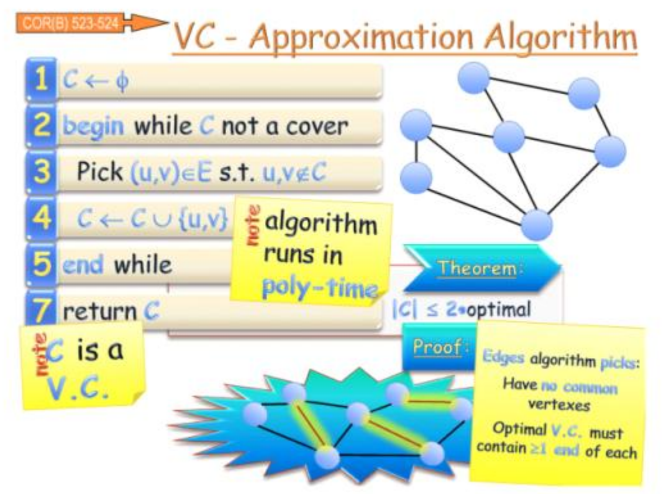
- An *α-approximation* algorithm for a maximization/minimization problem with an optimum solution O, returns a solution that is ≥αO/≤αO

Note

- α and O may depend on the size of the input

6

Many times it suffices to come up with an *approximated* solution, namely one which is not optimal, however deviates from optimal by some known factor. This factor is called the *approximation factor*; it may be a constant or depend on the input size.



COR(B) 523-524

VC - Approximation Algorithm

```
1 $C \leftarrow \phi$
2 begin while $C$ not a cover
3   Pick $(u,v) \in E$ s.t. $u,v \notin C$
4   $C \leftarrow C \cup \{u,v\}$
5 end while
7 return $C$
```

note: algorithm runs in poly-time

note: C is a V.C.

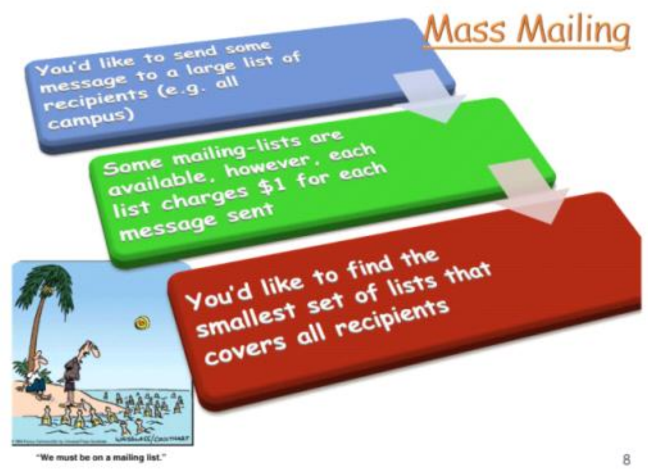
Theorem: $|C| \le 2 \cdot$optimal

Proof: Edges algorithm picks: Have no common vertexes. Optimal V.C. must contain ≥1 end of each

Let us now describe a very simple *approximation algorithm* for Vertex Cover:

at every stage, pick an edge not yet covered and add *both* its ends to the cover.

The cover returned by this algorithm is at most twice as large as the optimal.
To see that, note that the edges picked by the algorithm have no common vertexes. The optimal solution must contain one vertex for each of these edges, while we added both.



Mass Mailing

You'd like to send some message to a large list of recipients (e.g. all campus)

Some mailing-lists are available, however, each list charges $1 for each message sent

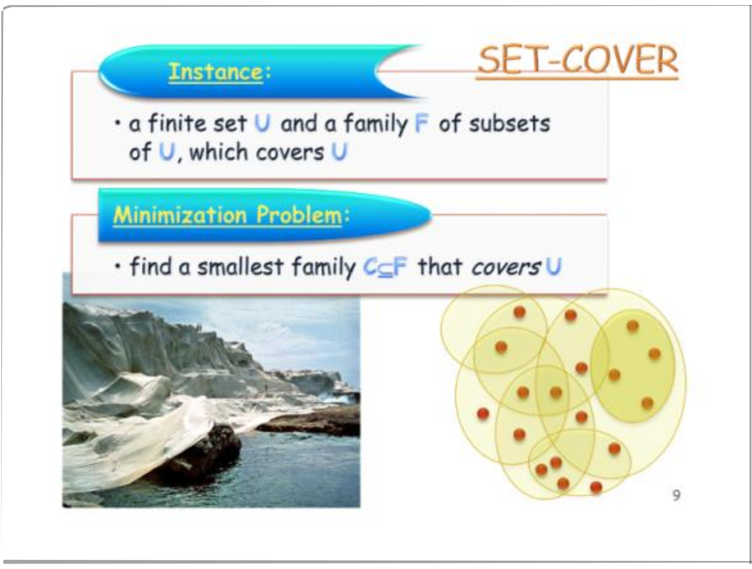
You'd like to find the smallest set of lists that covers all recipients

"We must be on a mailing list."

8

Now consider a different problem:

You have a set of mail recipients, and some mailing lists.
These mailing lists charge $1 for each use.
Hence, when sending a message, one tries to find the minimal number of mailing lists that cover all recipients.



SET-COVER

Instance:

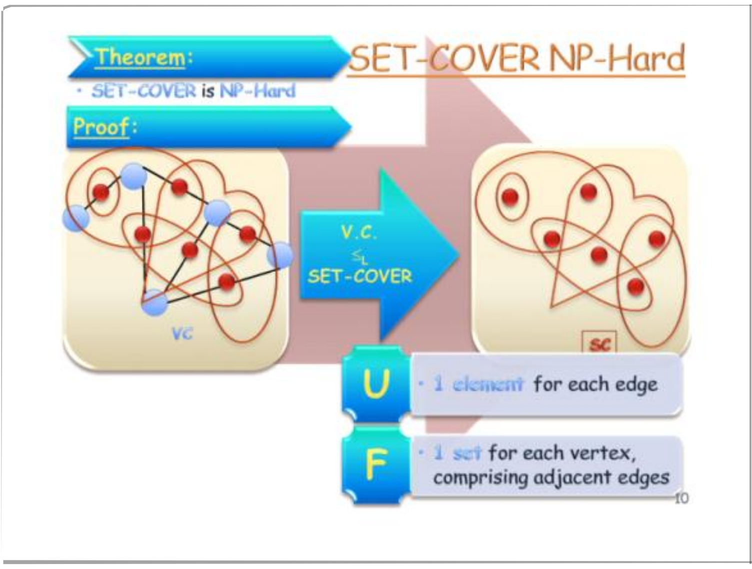
- a finite set U and a family F of subsets of U, which covers U

Minimization Problem:

- find a smallest family C⊆F that *covers* U

9

Formally, in a *Set Cover* instance, we have the universe U and a family F of subsets of U.
The goal is to find the smallest number of subsets in the family that cover the entire universe.



SET-COVER NP-Hard

Theorem:

- SET-COVER is NP-Hard

Proof:

VC
V.C. $\leq_L$ SET-COVER
SC

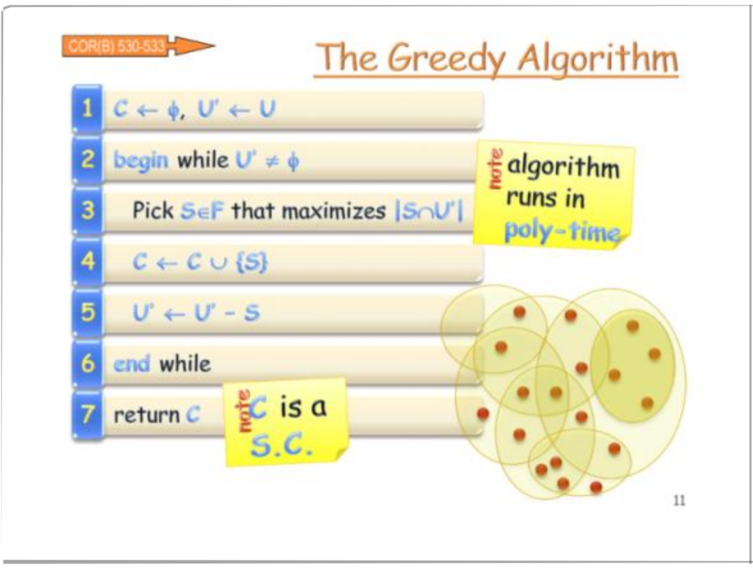
U · 1 element for each edge

F · 1 set for each vertex, comprising adjacent edges

10

Set Cover is NP-hard.

We prove this via direct reduction from Vertex Cover:
Each edge becomes an element of the universe, while each vertex becomes a subset comprising all of the edges that touch it.



COR(B) 530-533

The Greedy Algorithm

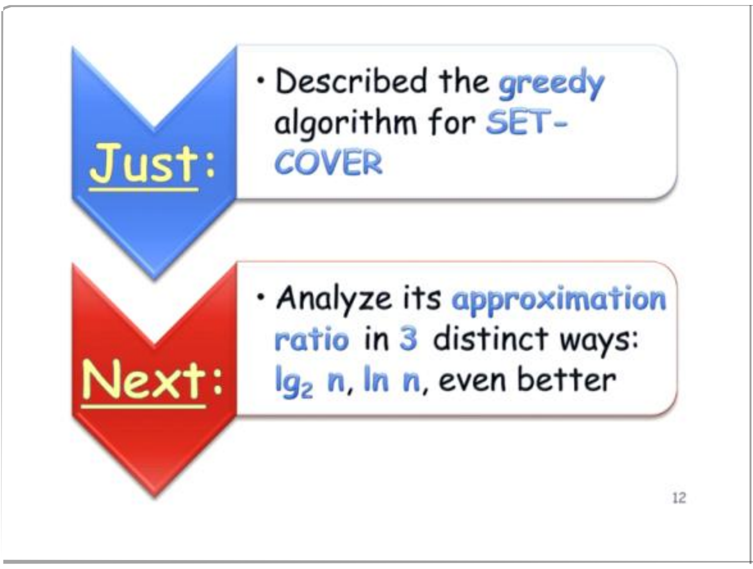
```
1 $C \leftarrow \phi$, $U' \leftarrow U$
2 begin while $U' \neq \phi$
3     Pick $S \in F$ that maximizes $|S \cap U'|$
4     $C \leftarrow C \cup \{S\}$
5     $U' \leftarrow U' - S$
6 end while
7 return $C$
```

note algorithm runs in poly-time

note C is a S.C.

11

To approximate Set Cover, let us introduce a general scheme for simple approximation algorithms: the *greedy strategy*, which at every stage picks locally the most advantageous option. For Set Cover, this means choosing a subset that covers the most out of the yet uncovered elements.



Just:

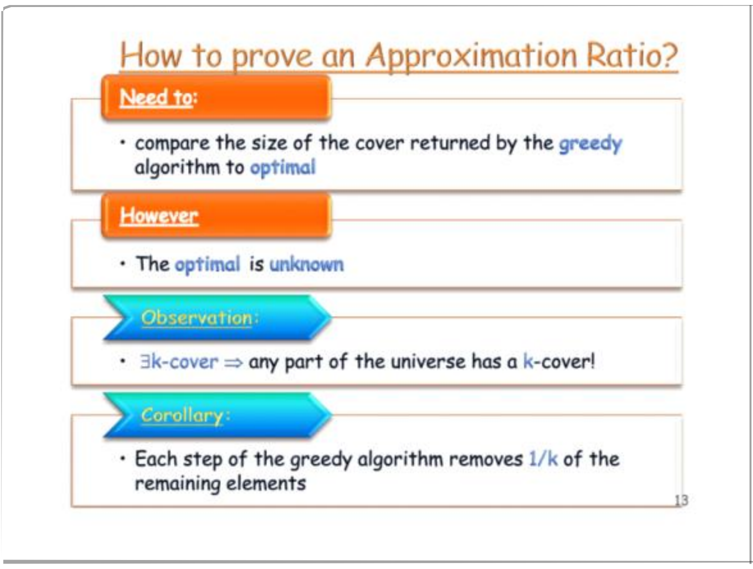
- Described the greedy algorithm for SET-COVER

Next:

- Analyze its approximation ratio in 3 distinct ways: $\lg_2 n$, $\ln n$, even better

12

In the next few slides we look at the greedy algorithm, and analyze its performance in three different ways.



How to prove an Approximation Ratio?

Need to:

- compare the size of the cover returned by the greedy algorithm to optimal

However

- The optimal is unknown

Observation:

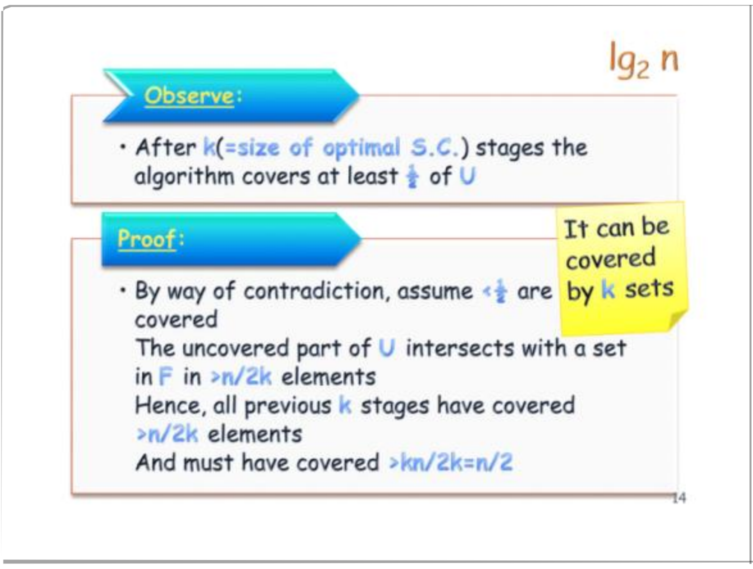
- $\exists$k-cover $\Rightarrow$ any part of the universe has a k-cover!

Corollary:

- Each step of the greedy algorithm removes 1/k of the remaining elements

13

The first point we need to realize is that we are going to compare the solution given by the Greedy Algorithm to the optimal solution, however, without any information about the optimal solution.
We will use the fact that if the optimal solution contains k subsets, any part of the universe ---in particular, the set of yet uncovered elements--- can be covered by k subsets.
Hence, at each stage, the algorithm covers 1/k fraction of the remaining elements.



$\lg_2 n$

Observe:

- After k(=size of optimal S.C.) stages the algorithm covers at least $\frac{1}{2}$ of U

Proof:

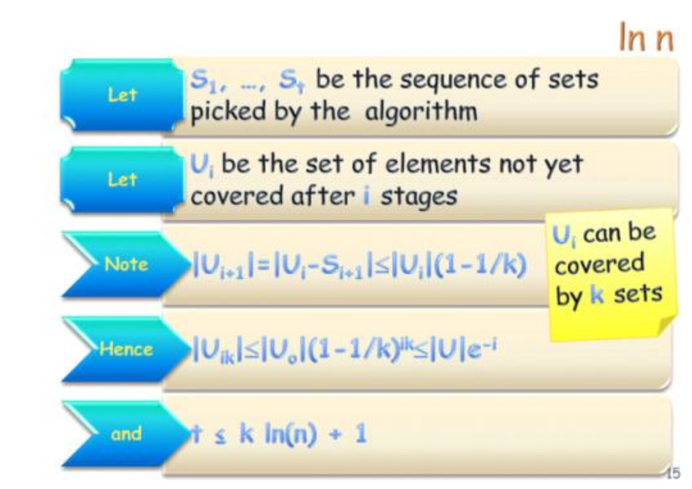
- By way of contradiction, assume $<\frac{1}{2}$ are covered
  The uncovered part of U intersects with a set in F in >n/2k elements
  Hence, all previous k stages have covered >n/2k elements
  And must have covered >kn/2k=n/2

It can be covered by k sets

14

To prove the Greedy Algorithm approximates the optimal Set Cover to within a *logarithmic factor* (log base 2), we show that every k stages cover half of the remaining elements.

This is true since, after k stages, a fraction of 1/k of the remaining elements must be covered in the next stage, and thus all previous stages must've covered at least that number of elements.



ln n

Let $S_1, \ldots, S_t$ be the sequence of sets picked by the algorithm

Let U_i be the set of elements not yet covered after i stages

Note $|U_{i+1}|=|U_i-S_{i+1}|\le|U_i|(1-1/k)$

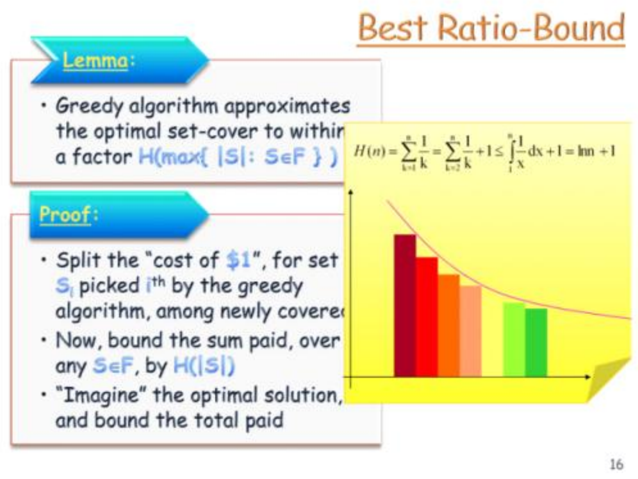
U_i can be covered by k sets

Hence $|U_{ik}|\le|U_0|(1-1/k)^{ik}\le|U|e^{-i}$

and $t \le k\ln(n)+1$

15

To prove a tighter bound, where the log is the natural log, we notice that at every stage a fraction 1/k of the remaining elements is covered. Hence, the number of remaining elements after k * ln |U| stages is 1.



Best Ratio-Bound

Lemma:

- Greedy algorithm approximates the optimal set-cover to within a factor $H(\max\{\ |S|:\ S\in F\ \})$

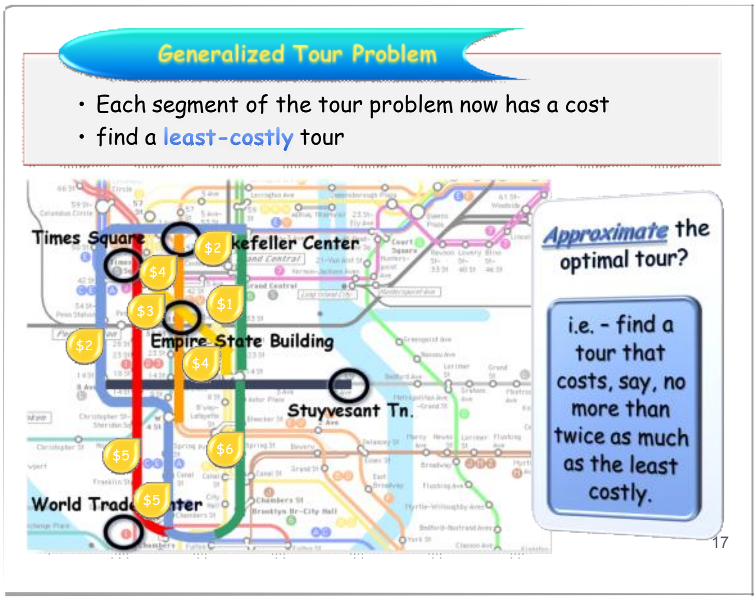
Proof:

- Split the "cost of \$1", for set S_i picked i^{th} by the greedy algorithm, among newly covered
- Now, bound the sum paid, over any $S\in F$, by $H(|S|)$
- "Imagine" the optimal solution, and bound the total paid

$$H(n)=\sum_{k=1}^{n}\frac{1}{k}=\sum_{k=2}^{n}\frac{1}{k}+1\le\int_{1}^{n}\frac{1}{x}dx+1=\ln n+1$$

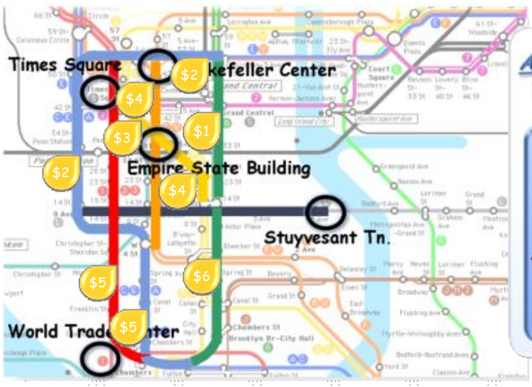
16

The best bound, which is ln of the size of the largest set in F, is obtained as followed:
Let us think of a price of the use of each mailing list being split among the recipients in the following manner. At each stage of the algorithm an additional \$1 has to be paid, and it is split among those recipients just covered.
A bound the total amount paid, will give us a bound on the number of subsets the greedy algorithm outputs.
We now look at any subset S of F, and bound the total amount its members pay by ln |S|.
This will end the proof, as we can consider the optimal cover of k subsets (not really knowing what it is) and see that each of its subsets pay at most ln of its size --- altogether at most k*ln|S| for the largest S in the optimal cover.
Now for an arbitrary set, in whichever stage of the algorithm, if m of its members are not yet covered, the greedy algorithm chooses a subset that covers at least m elements, hence each will pay at most 1/m of \$1.
The members of the set S will pay the most, if they are covered one by one, and in each stage the set chosen covers exactly m elements, which will result with the harmonic series, which sums up to ln of S's size.



Generalized Tour Problem

- Each segment of the tour problem now has a cost
- find a least-costly tour

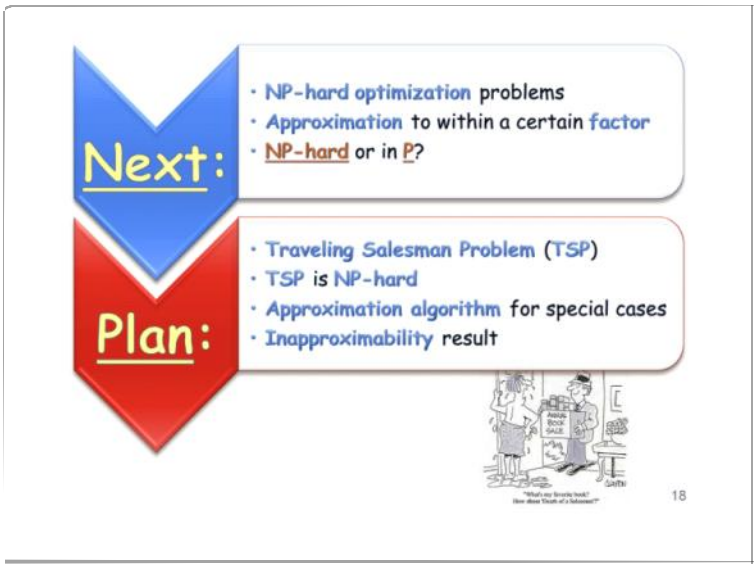


Approximate the optimal tour?

i.e. - find a tour that costs, say, no more than twice as much as the least costly.

17

Now we go back to a problem we mentioned early in the course, namely, the tour problem where segments have a price that may vary.



Next:

- NP-hard optimization problems
- Approximation to within a certain factor
- NP-hard or in P?

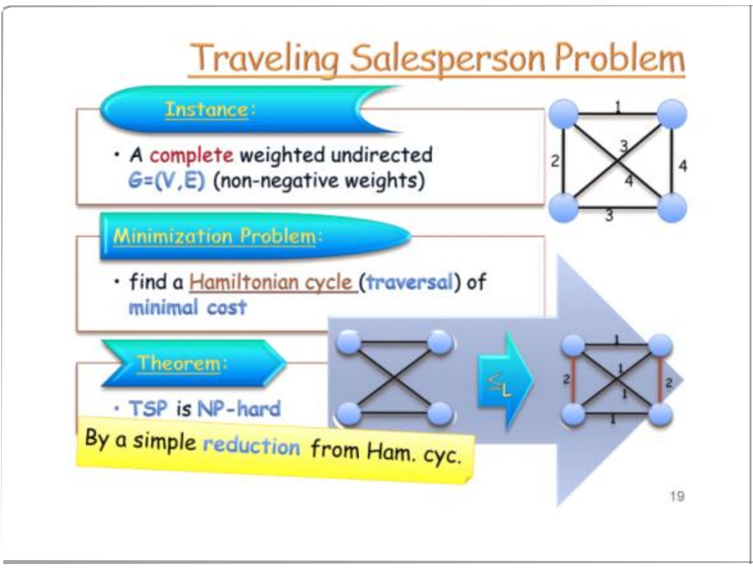
Plan:

- Traveling Salesman Problem (TSP)
- TSP is NP-hard
- Approximation algorithm for special cases
- Inapproximability result

18

This is the Traveling Salesperson Problem (TSP): we will analyze the complexity of the general variant as well as a more restricted variant.

We will show an approximation algorithm in the special case, and prove an NP-hardness result, introducing a technique that will help us prove such results for approximation problems later on.



Traveling Salesperson Problem

Instance:

- A complete weighted undirected $G=(V,E)$ (non-negative weights)

Minimization Problem:

- find a Hamiltonian cycle (traversal) of minimal cost

Theorem:

- TSP is NP-hard

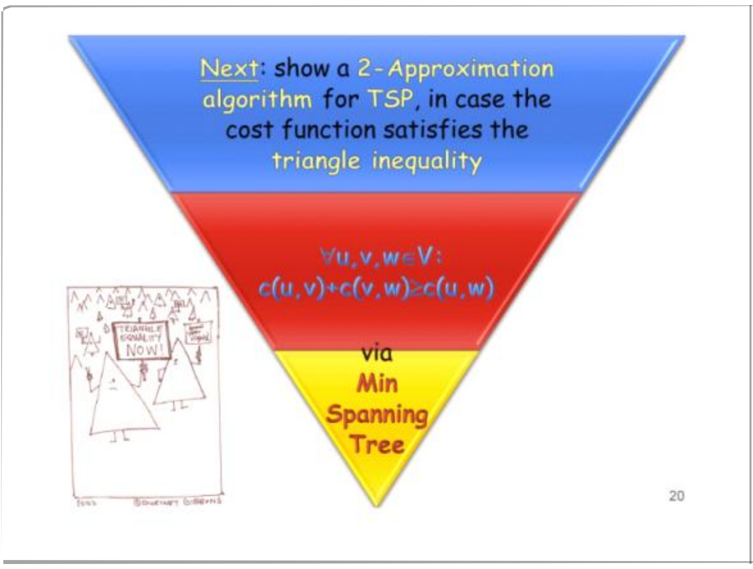
By a simple reduction from Ham. cyc.

1
2
3
4
4
3
$\le_L$
1
2
1
1
2
1

19

Formally, a TSP instance is a graph with non-negative weights, and the goal is to find a traversal (a Hamiltonian cycle) that is of the smallest sum of weights.

We immediately see that the optimization exact solution is NP-hard, via a simple reduction from Hamiltonian cycle (edges become edges of weight 1; non-edges become edges of larger weight).



Next: show a 2-Approximation algorithm for TSP, in case the cost function satisfies the triangle inequality

$\forall u,v,w \in V$:
$c(u,v)+c(v,w) \ge c(u,w)$

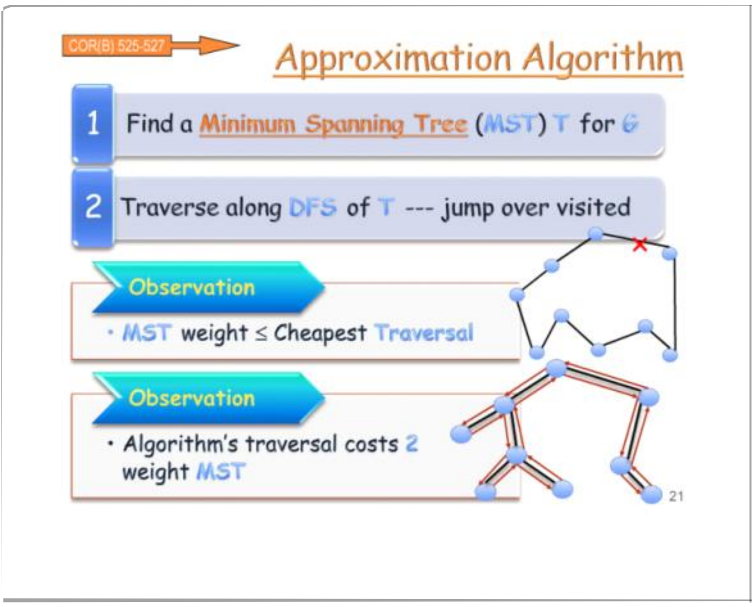
via Min Spanning Tree

TRIANGLE EQUALITY NOW!

20

Now, we move to a special ---more interesting--- case of TSP, where the weights satisfy the *triangle inequality*.

In this case we are able to come up with an Approximation Algorithm, utilizing a procedure for finding the Minimum Spanning Tree (MST).



COR(B) 525-527

Approximation Algorithm

1 Find a Minimum Spanning Tree (MST) T for G

2 Traverse along DFS of T --- jump over visited

Observation

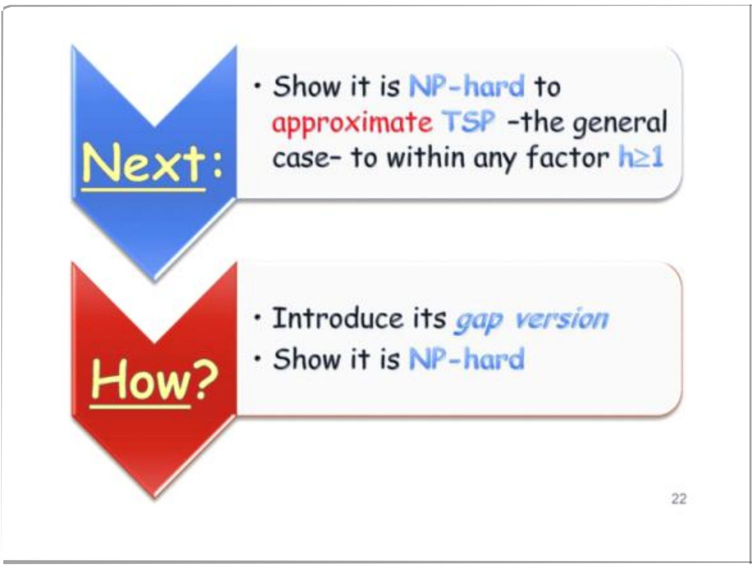
- MST weight ≤ Cheapest Traversal

Observation

- Algorithm's traversal costs 2 weight MST

21

The algorithm starts by finding a MST of the graph, then follows a DSP to traverse the tree. While doing so some vertexes have already been visited, but due to the triangle inequality the traverse can simply jump to the next yet unvisited vertex, paying at most the sum of edges along the way.
To see this approximates TSP to within a factor of 2, note first that the minimal traversal bounds from above the weight of the MST, while the algorithm returns a traversal which is at most twice the MST weight.



Next:

- Show it is NP-hard to approximate TSP -the general case- to within any factor h≥1

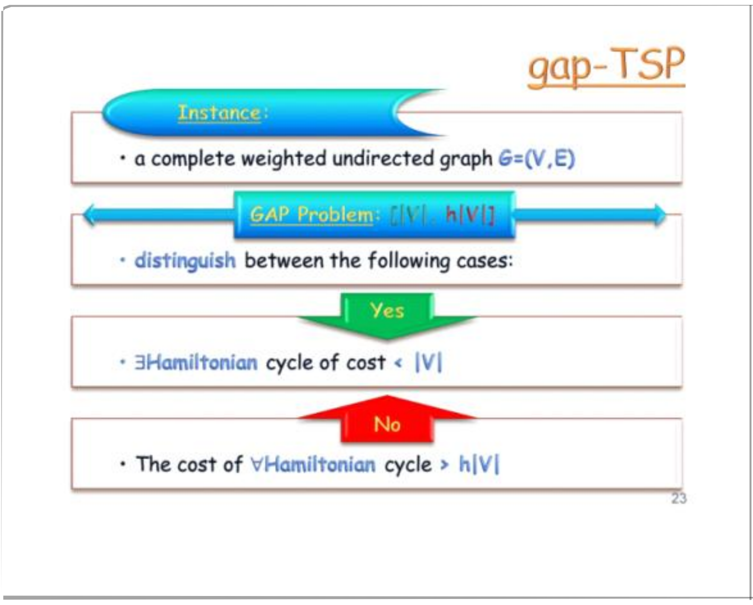
How?

- Introduce its *gap version*
- Show it is NP-hard

22

We are now prepared to show our first *NP-hardness* result for an *approximation* problem.

We will define for that purpose Gap Problems (which give us the closest version to a decision problem) and see how to prove NP-hardness of a Gap Problem, and how to obtain from that an NP-hardness result for the corresponding approximation problem.



gap-TSP

Instance:

- a complete weighted undirected graph $G=(V,E)$

GAP Problem: $[|V|, h|V|]$

- distinguish between the following cases:

Yes

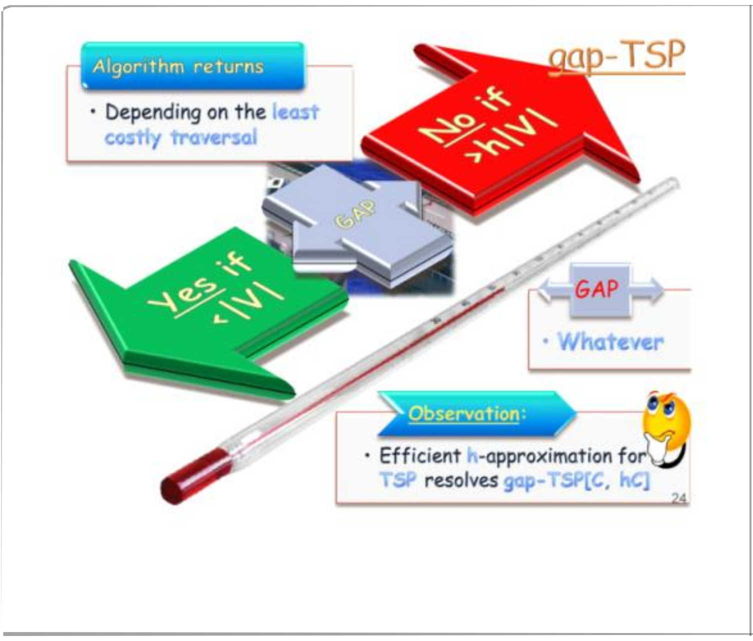
- $\exists$Hamiltonian cycle of cost $<$ $|V|$

No

- The cost of $\forall$Hamiltonian cycle $>$ $h|V|$

23

A Gap Problem takes an optimization problem and defines two thresholds. For a minimization problem, the good instances are those whose optimization parameter is below the low threshold, while the bad instances are those for which this parameter turns out to be higher than the high threshold. The instances whose parameter is in between are the "don't care" part.
In TSP case, we define the Gap Problem to have the number of vertexes as the low threshold, while h times that to be the high threshold --- this for an arbitrary h.



gap-TSP

Algorithm returns

- Depending on the least costly traversal

No if
>h|V|
GAP
Yes if
<|V|

GAP

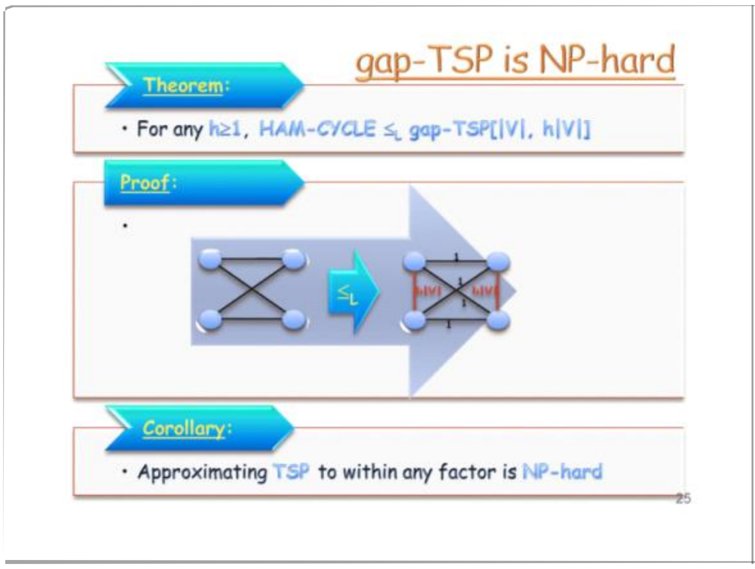
- Whatever

Observation:

- Efficient h-approximation for TSP resolves gap-TSP[C, hC]

24

An algorithm for a Gap Problem must *accept* all *good* instances, must *reject* all *bad* instances, and can *accept/reject* all the *"don't care"* instances.
An efficient algorithm that approximates the problem to within a factor h, can be used to efficiently solve the Gap Problems where the thresholds are C and hC, for any C: simply apply it and accept all instances for which the solution returned lets the optimization parameter be smaller than hC.
No bad instances can be accepted, while no good instance will be rejected as the approximation factor is h.



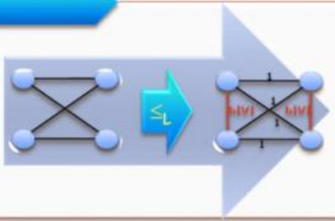
gap-TSP is NP-hard

Theorem:

- For any $h \geq 1$, HAM-CYCLE $\leq_L$ gap-TSP[$|V|$, $h|V|$]

Proof:

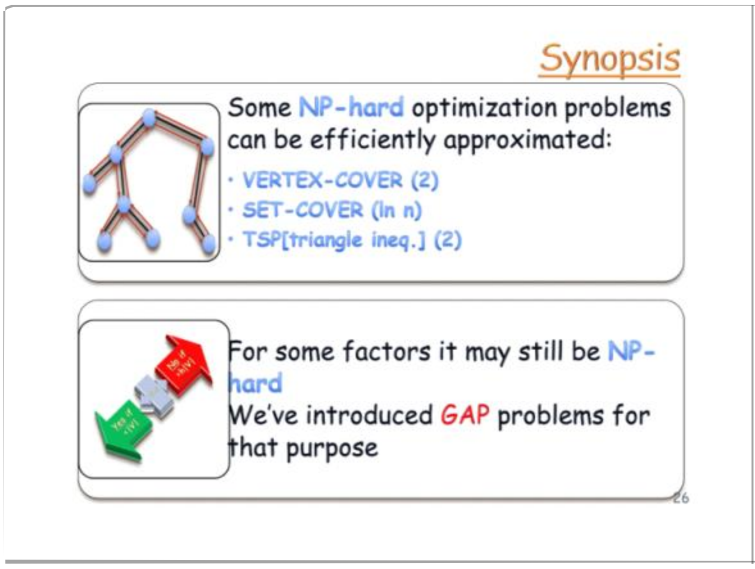
-



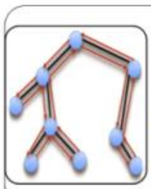
Corollary:

- Approximating TSP to within any factor is NP-hard

25



Synopsis



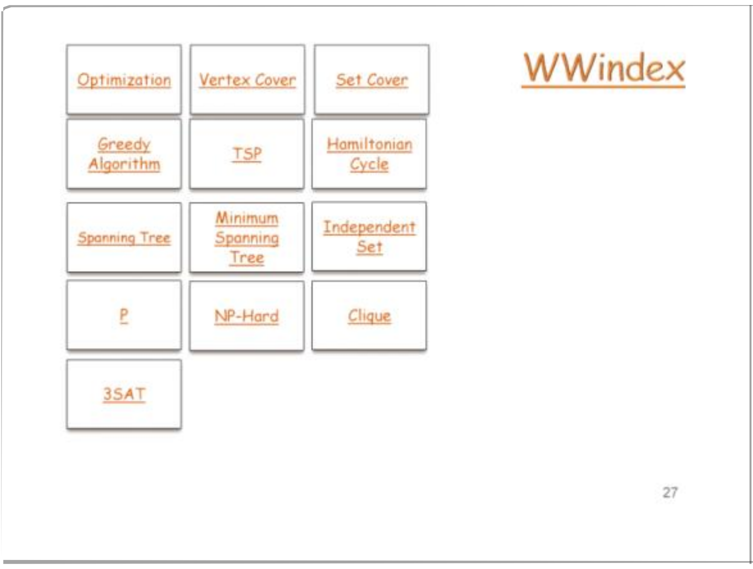
Some NP-hard optimization problems can be efficiently approximated:

- VERTEX-COVER (2)
- SET-COVER (ln n)
- TSP[triangle ineq.] (2)

For some factors it may still be NP-hard

We've introduced GAP problems for that purpose

26



WWindex

Optimization	Vertex Cover	Set Cover
Greedy Algorithm	TSP	Hamiltonian Cycle
Spanning Tree	Minimum Spanning Tree	Independent Set
P	NP-Hard	Clique
3SAT		

27

- Optimization
- Vertex Cover
- Greedy Algorithm
- Spanning Tree
- Set Cover
- Minimum Spanning Tree
- Independent Set
- P
- NP-Hard
- Hamiltonian Cycle
- TSP
- Clique
- 3SAT