A Short Tour of Tractability Hunting

With a special focus on Parameterized Complexity

Martin Berglund

Umeå University

September 6, 2014

On Guest Lectures

This is a slightly badly timed *guest lecture*

Course only starting: lot of fundamentals remain. Yet I want to say something aspirational

On Guest Lectures

This is a slightly badly timed *guest lecture*

Course only starting: lot of fundamentals remain. Yet I want to say something aspirational

As such I will give a *short tour* of some complexity theory concepts ahead of their time

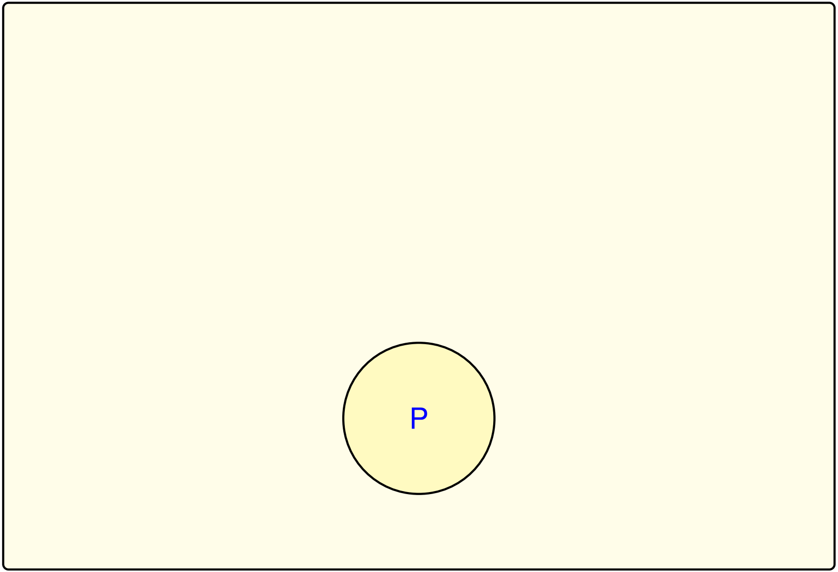
I will assume:

- Propositional logic
- A basic idea of what complexity, P and NP, *means*
- Some minor math and $\mathcal{O}$-notation

Will hopefully explain the rest

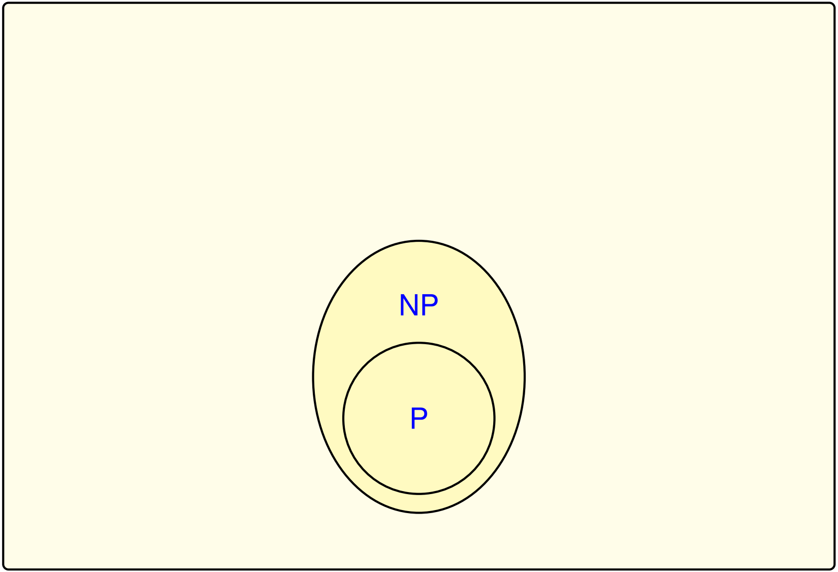
Part I: Complexity Background and Tractability

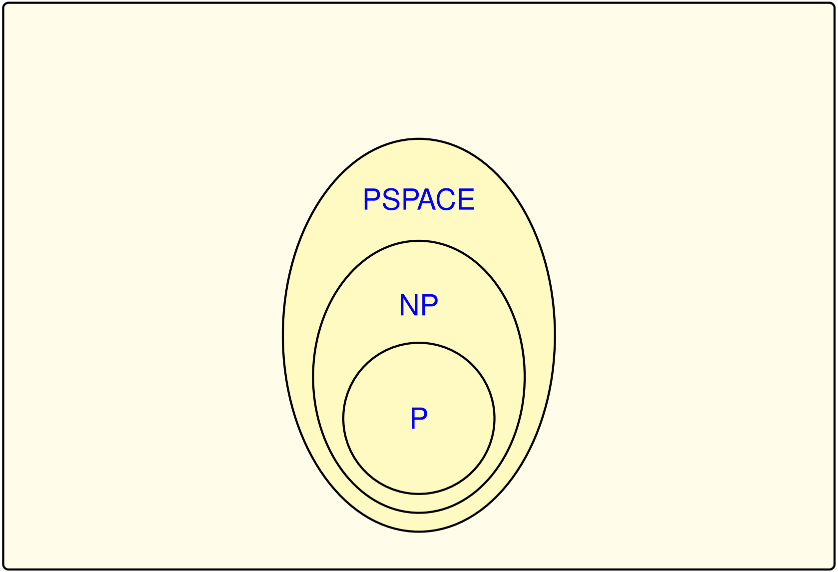
Classical complexity

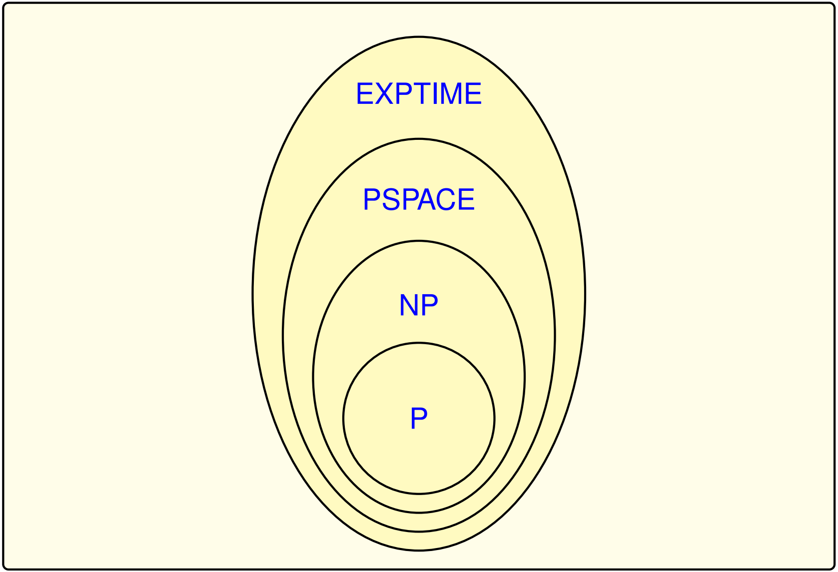


big-oh, poly, circuit eval, moore's

Classical complexity




PSPACE
NP
P


EXPTIME
PSPACE
NP
P

Tractability = P?

Objection 1: A nicely behaved exponential is sometimes better than an ugly polynomial.

$$\begin{array}{ccc} 2^n & \textit{vs.} & n^{10} \\ 2^{50} & < & 50^{10} \end{array}$$

Tractability = P?

Objection 1: A nicely behaved exponential is sometimes better than an ugly polynomial.

$$2^n \quad vs. \quad n^{10}$$
$$2^{50} \quad < \quad 50^{10}$$

Objection 2: We simply cannot afford to consider all NP-hard problems intractable.

Tractability = P?

Objection 1: A nicely behaved exponential is sometimes better than an ugly polynomial.

$$\begin{array}{ccc} 2^n & vs. & n^{10} \\ 2^{50} & < & 50^{10} \end{array}$$

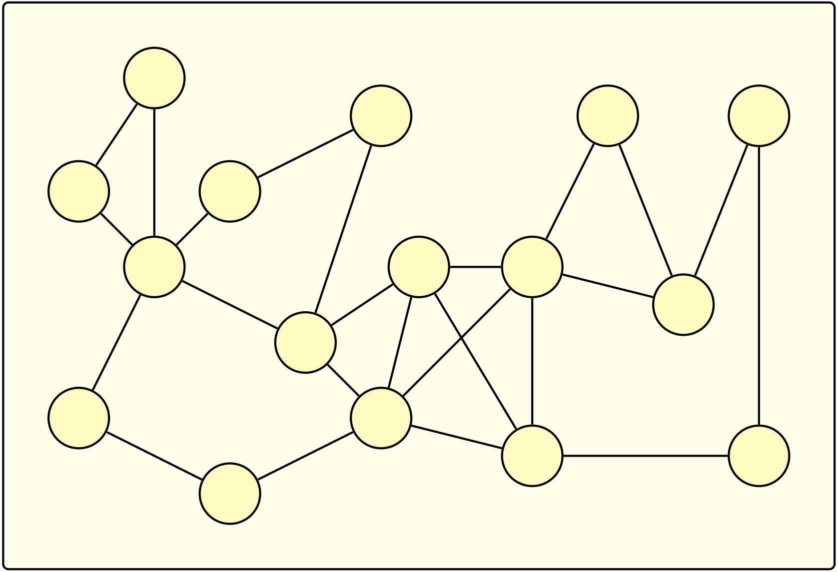
Objection 2: We simply cannot afford to consider all NP-hard problems intractable.

Still, we cannot just consider NP *tractable*: Cook was right!

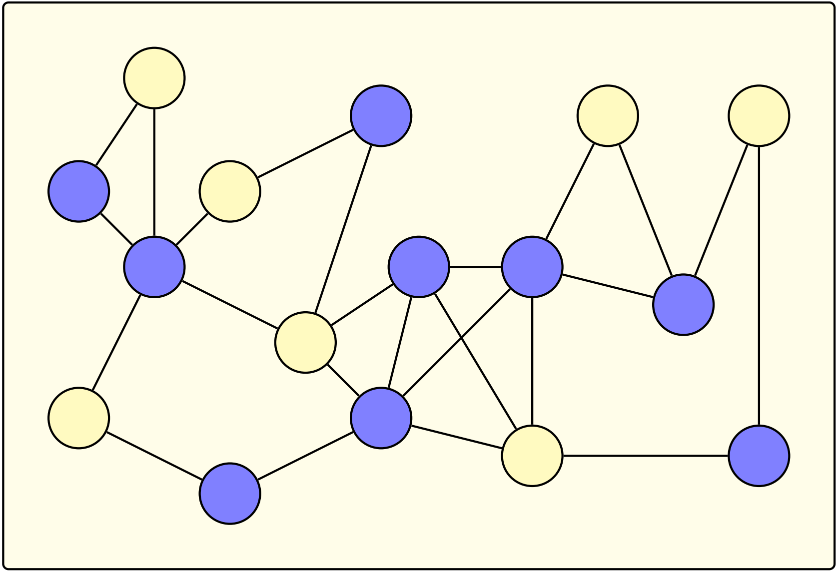
VERTEX COVER

Input: A graph $G = (V, E)$, a constant $k \in N$.
Output: "Yes" if and only if there exists some $V' \subseteq V$, $|V'| \leq k$ such that every edge in E touches a vertex in V'.

Vertex Cover: Example



Vertex Cover: Example

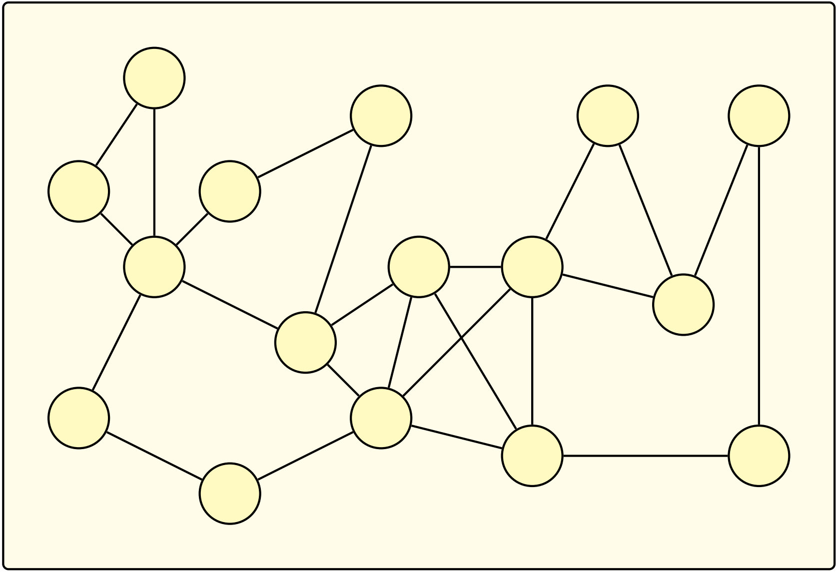


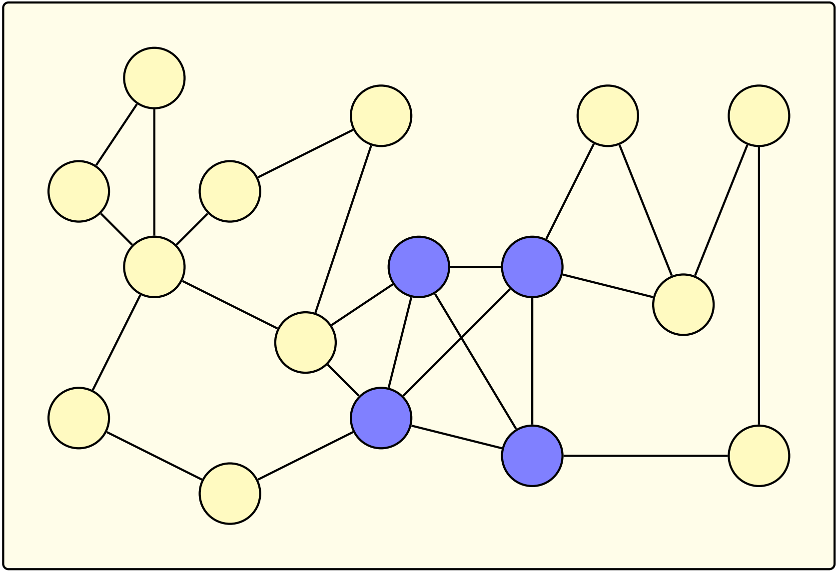
Vertex Cover: NP-complete

Vertex Cover is known to be *NP-complete*.

It is one of Karp's original 21 problems. Reduction from Chromatic Number (graph coloring).

It is easy to bring back to Satisfiability however, or at least 3-Satisfiability.





Practical solvability

CLIQUE is *also* NP-complete (reduction from SATISFIABILITY by Karp).

CLIQUE is *also* NP-complete (reduction from SATISFIABILITY by Karp).

However, while VERTEX COVER is *NP*-complete, but it is solvable in practice for rather large instances (comp. bio.)

Practical solvability

CLIQUE is *also* NP-complete (reduction from SATISFIABILITY by Karp).

However, while VERTEX COVER is *NP*-complete, but it is solvable in practice for rather large instances (comp. bio.)

However, for CLIQUE we have no practical algorithms for large instances

Practical solvability

CLIQUE is *also* NP-complete (reduction from SATISFIABILITY by Karp).

However, while VERTEX COVER is *NP*-complete, but it is solvable in practice for rather large instances (comp. bio.)

However, for CLIQUE we have no practical algorithms for large instances

LTL MODEL CHECKING is *PSPACE*-complete, but used in practice to verify hardware design

1. *Approximation*
2. *Randomization*
3. *“Islands of tractability”*
4. *Parallelization*
5. *Parameterization*

Approximation means accepting less than “perfect” solutions to reach a tractable algorithm.

- Only well defined for *optimization problems*: e.g. SATISFIABILITY has no “near” solution.

Approximation means accepting less than “perfect” solutions to reach a tractable algorithm.

- Only well defined for *optimization problems*: e.g. SATISFIABILITY has no “near” solution.

A algorithm is a c-approximation for a problem P if it computes an answer that is at most c times worse than the optimal.

VERTEX COVER has a 2-approximation in polynomial time.

CLIQUE cannot easily be approximated (this is a complex matter however).

Approximation means accepting less than “perfect” solutions to reach a tractable algorithm.

- Only well defined for *optimization problems*: e.g. SATISFIABILITY has no “near” solution.

A algorithm is a c-approximation for a problem P if it computes an answer that is at most c times worse than the optimal.

VERTEX COVER has a 2-approximation in polynomial time.

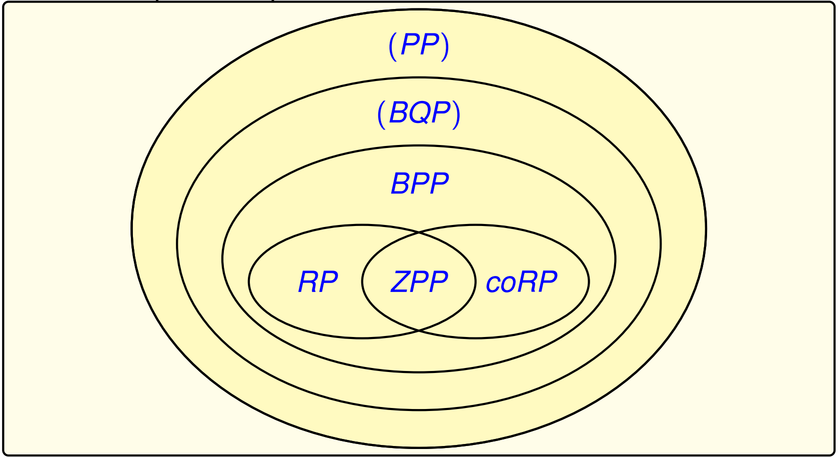
CLIQUE cannot easily be approximated (this is a complex matter however).

PTAS: Polynomial time approximation schemes are the centerpiece.

Randomized algorithms are algorithms permitted to *flip coins*

Randomized algorithms are algorithms permitted to *flip coins*

The zoo is quite complex:



Randomized is faster for many problems in P, but $P \stackrel{?}{=} BPP$

list, zpp, bqp, pp silly

3. Islands of Tractability

Not strictly defined, just a subset $P' \subset P$ of an NP-complete problem which is in P and has some argument why it is what is of *actual* interest.

Not strictly defined, just a subset $P' \subset P$ of an NP-complete problem which is in P and has some argument why it is what is of *actual* interest.

The most classic way of finding tractable subsets is *slicing* and *fixing constants*

Not strictly defined, just a subset $P' \subset P$ of an NP-complete problem which is in P and has some argument why it is what is of *actual* interest.

The most classic way of finding tractable subsets is *slicing* and *fixing constants*

Fixing constants: SATISFIABILITY is in P if no clause has more than 2 literals.

Not strictly defined, just a subset $P' \subset P$ of an NP-complete problem which is in P and has some argument why it is what is of *actual* interest.

The most classic way of finding tractable subsets is *slicing* and *fixing constants*

Fixing constants: SATISFIABILITY is in P if no clause has more than 2 literals.

Slicing: For every constant k the k-Clique problem is in P.

3. Islands of Tractability

Not strictly defined, just a subset $P' \subset P$ of an NP-complete problem which is in P and has some argument why it is what is of *actual* interest.

The most classic way of finding tractable subsets is *slicing* and *fixing constants*

Fixing constants: SATISFIABILITY is in P if no clause has more than 2 literals.

Slicing: For every constant k the k-Clique problem is in P.

Easy but clumsy: parameterized complexity picks up from here

The fan favourite *parallelization*: spread work among more machines

The fan favourite *parallelization*: spread work among more machines

The complexity theory is quite interesting, the **NC** (Nick's Class after Nick Pippenger) hierarchy defines how problems may be split

(log to c) n, n to k

The fan favourite *parallelization*: spread work among more machines

The complexity theory is quite interesting, the **NC** (Nick's Class after Nick Pippenger) hierarchy defines how problems may be split, *within P*

4. Parallelization

The fan favourite *parallelization*: spread work among more machines

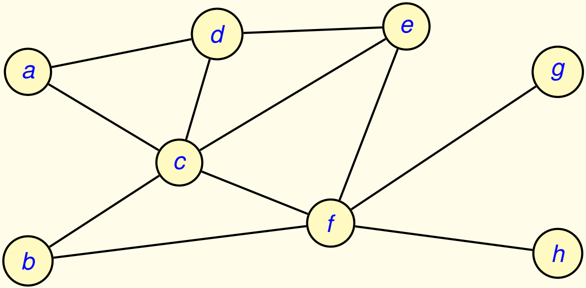
The complexity theory is quite interesting, the **NC** (Nick's Class after Nick Pippenger) hierarchy defines how problems may be split, *within P*

in general parallelization operates within P to an even greater extent than randomization:

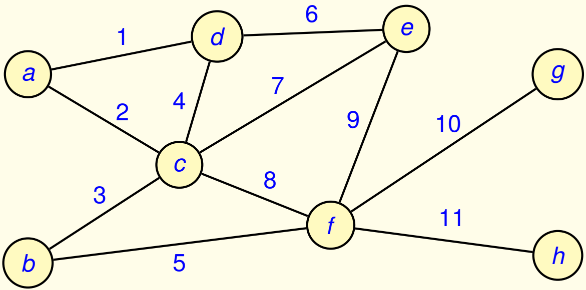
- Most complex problems actually resist parallelization
- Even if an NP-complete problem can be parallelized this entails *increasing the amount of hardware exponentially* (unless P=NP)

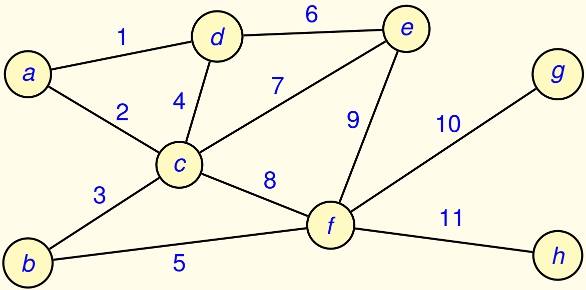
(log to c) n, n to k

Part II: Fixed Parameter Tractability



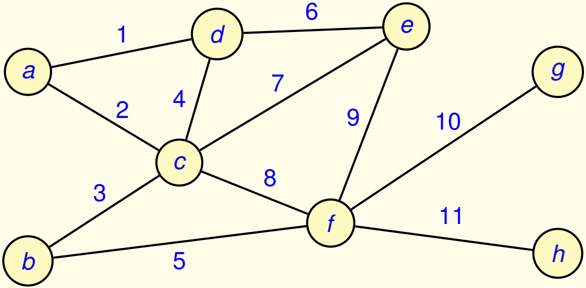
a
d
e
g
c
f
b
h


1
2
3
4
5
6
7
8
9
10
11
a
b
c
d
e
f
g
h


a
b
c
d
e
f
g
h
1
2
3
4
5
6
7
8
9
10
11

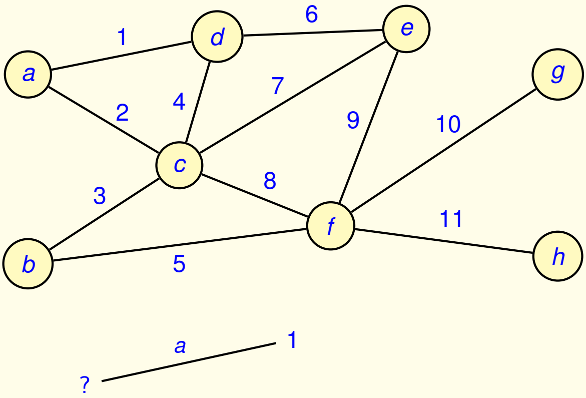
?

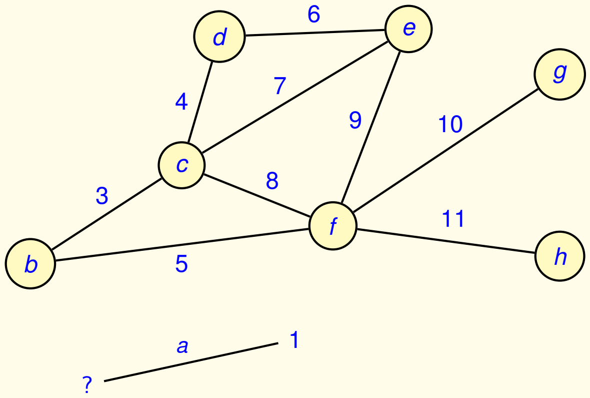
Solving 3-Vertex Cover



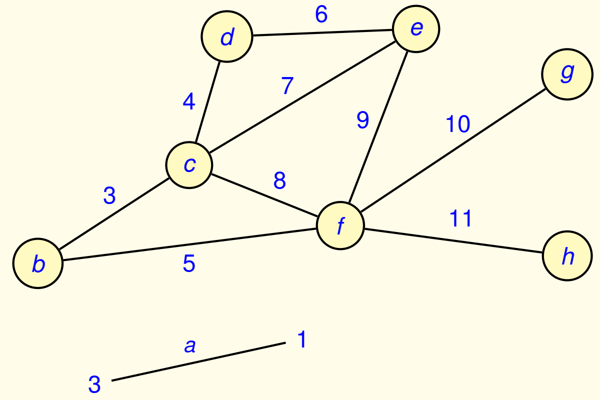
1

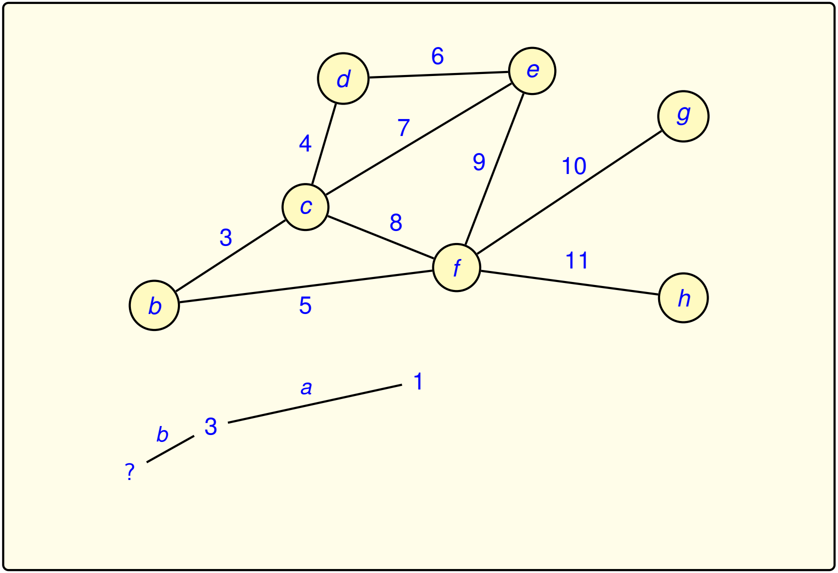
Solving 3-Vertex Cover




d
e
g
c
f
b
h
6
4
7
9
10
3
8
11
5
a
1
?

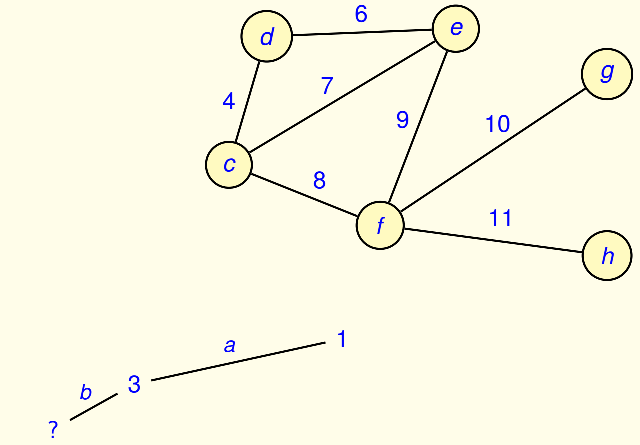
Solving 3-Vertex Cover

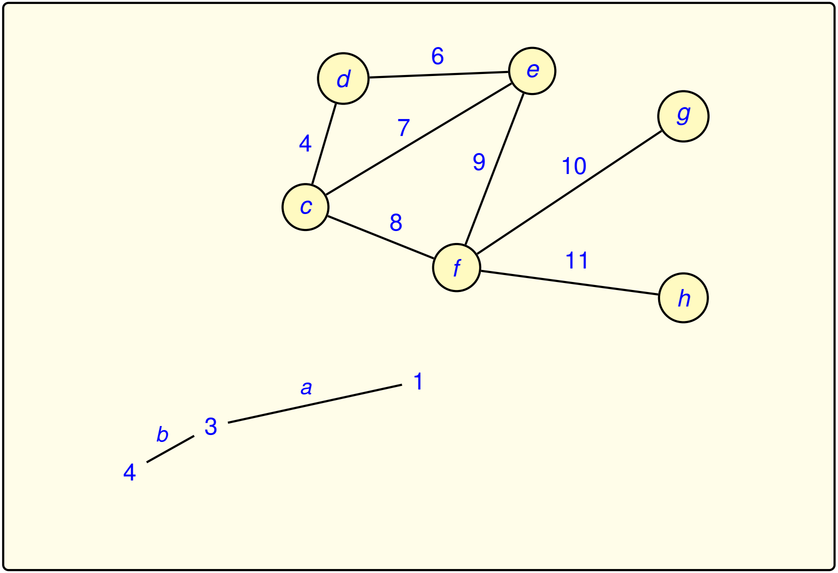




6
d
e
g
4
7
9
10
c
3
8
f
11
b
5
h
a
1
b
3
?

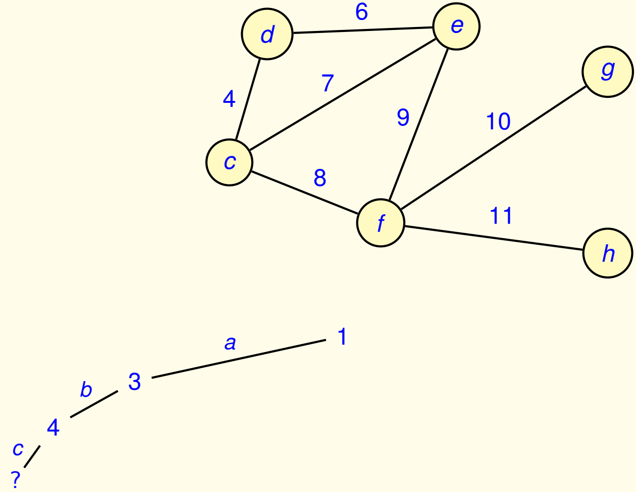
Solving 3-VERTEX COVER

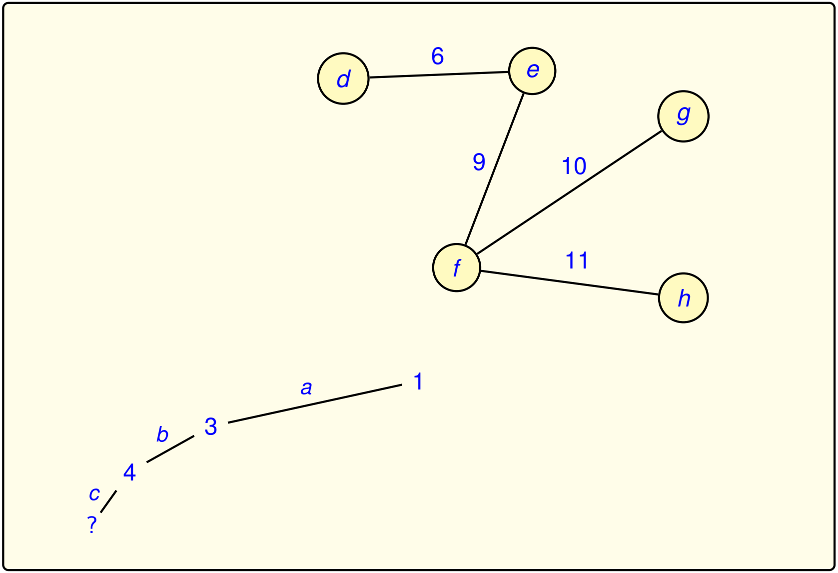


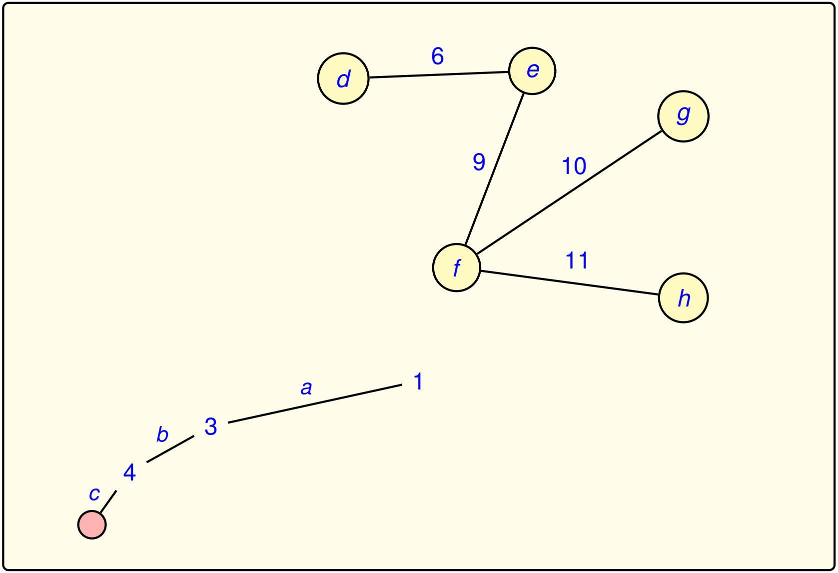


d
e
g
c
f
h
6
4
7
9
10
8
11
1
a
3
b
4

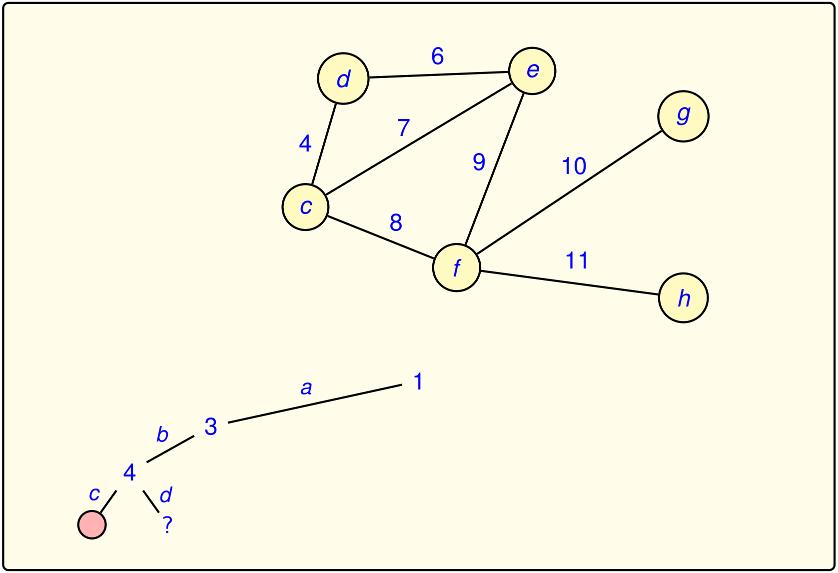
Solving 3-VERTEX COVER

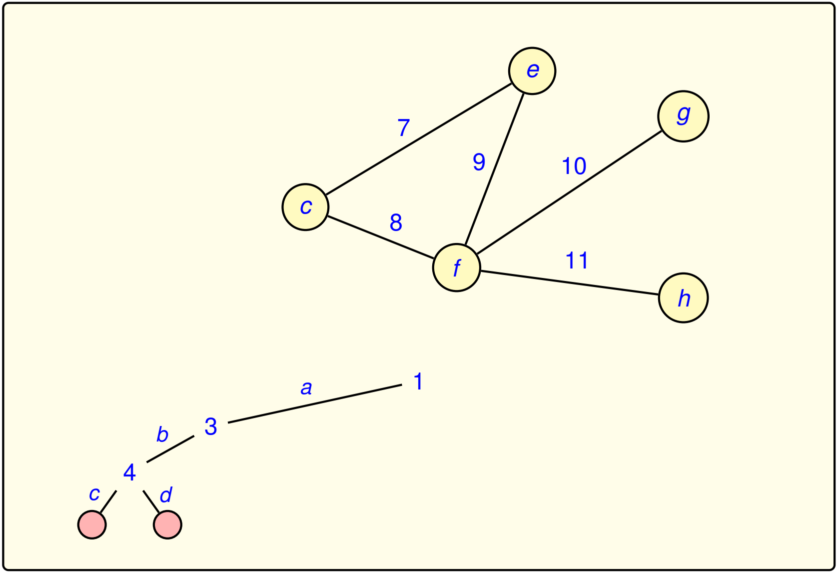


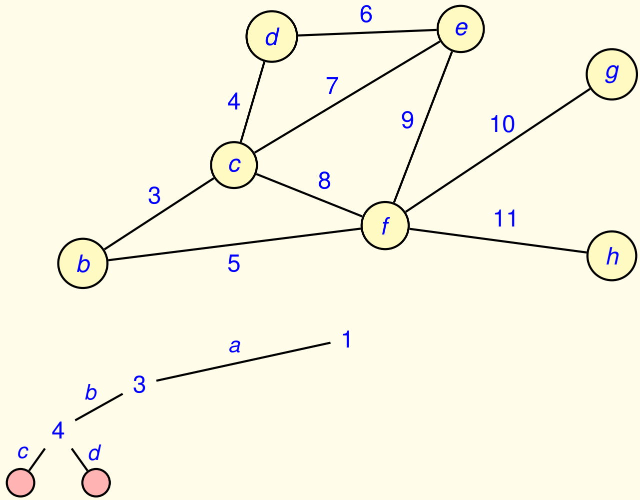

6
d
e
g
9
10
f
11
h
1
a
3
b
4
c
?

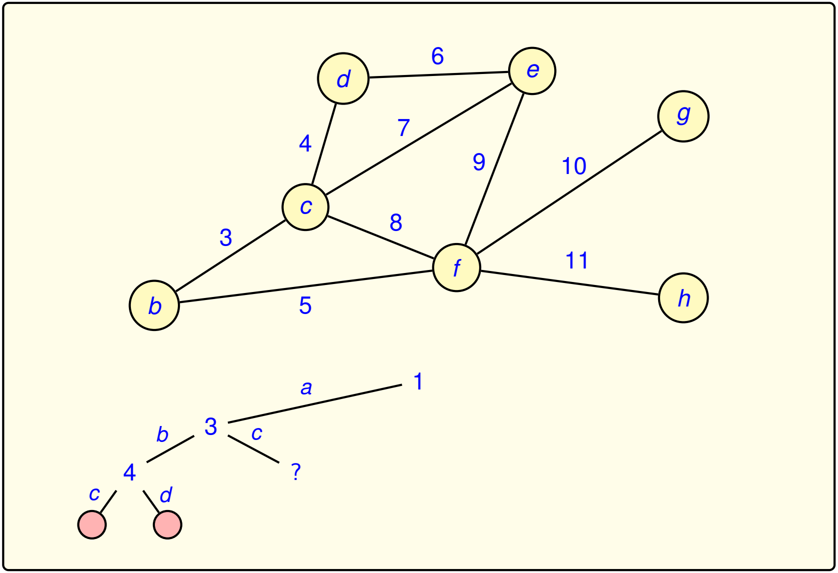

6
d
e
g
9
10
f
11
h
a
1
b
3
4
c

Solving 3-Vertex Cover

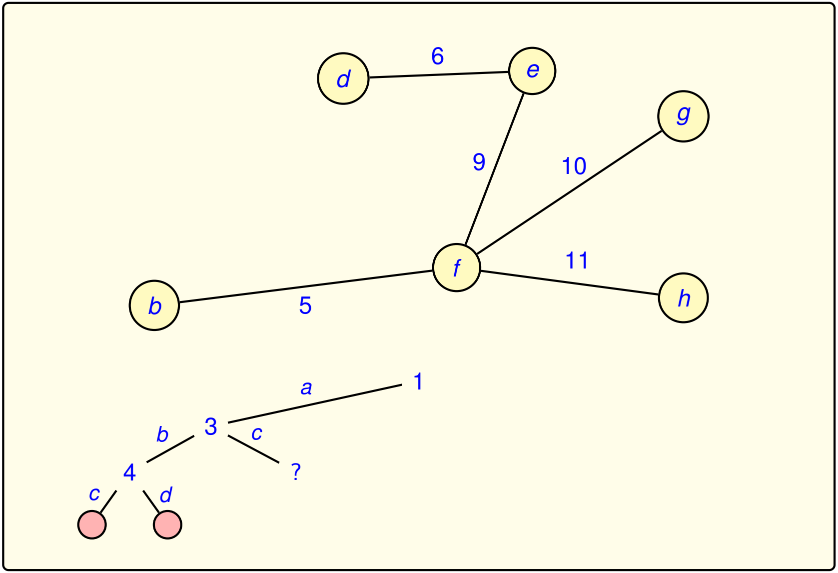


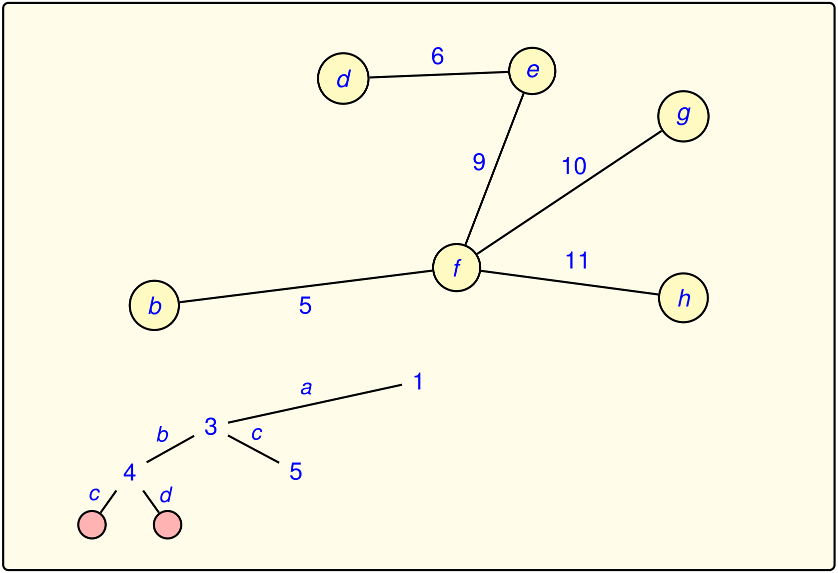

e
g
7
9
10
c
8
f
11
h
a
1
b
3
4
c
d


6
d
e
g
4
7
9
10
c
3
8
f
11
h
b
5
a
1
3
b
4
c
d

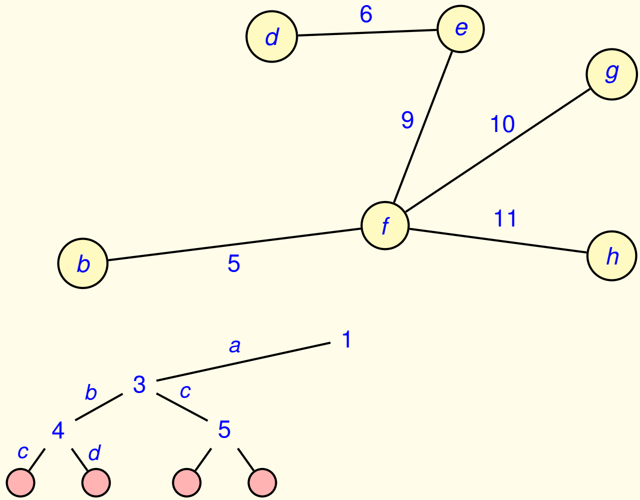

d
e
g
c
f
b
h
6
4
7
9
10
3
8
11
5
a
1
b
3
c
4
?
c
d

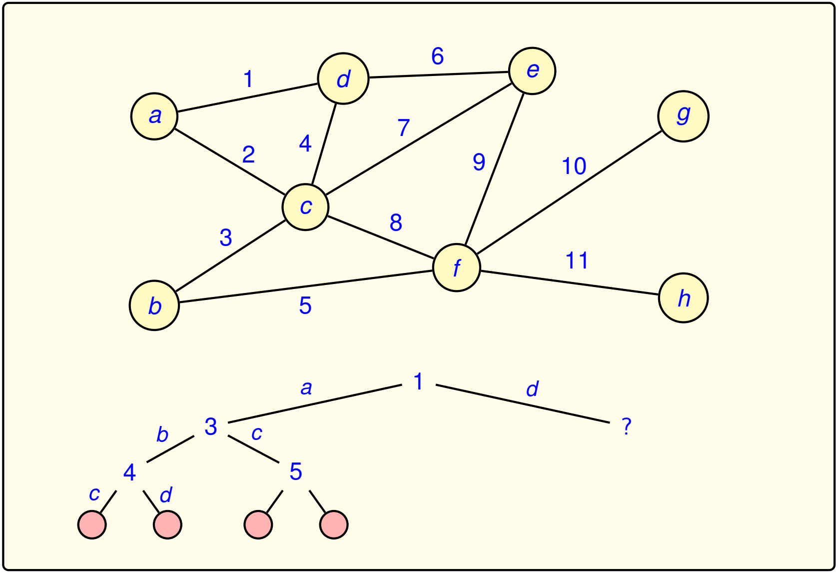
Solving 3-VERTEX COVER



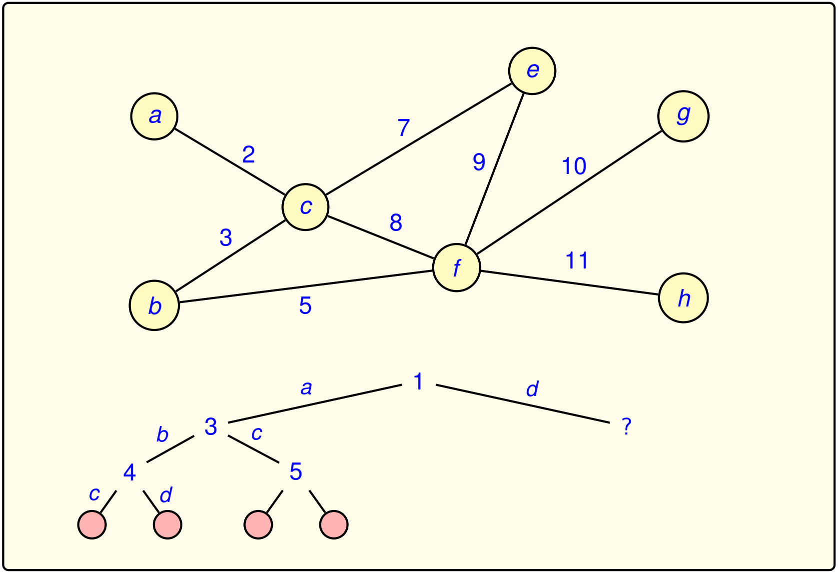


d
e
6
g
9
10
f
11
b
5
h
a
1
3
b
c
4
5
c
d

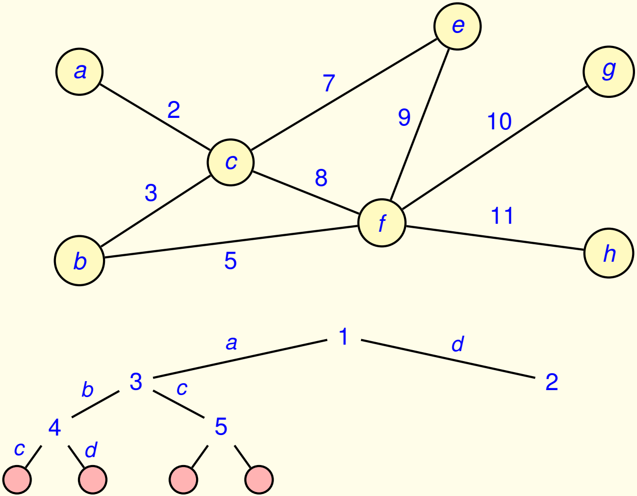

d
e
g
f
b
h
6
9
10
11
5
1
a
3
b
c
4
5
c
d



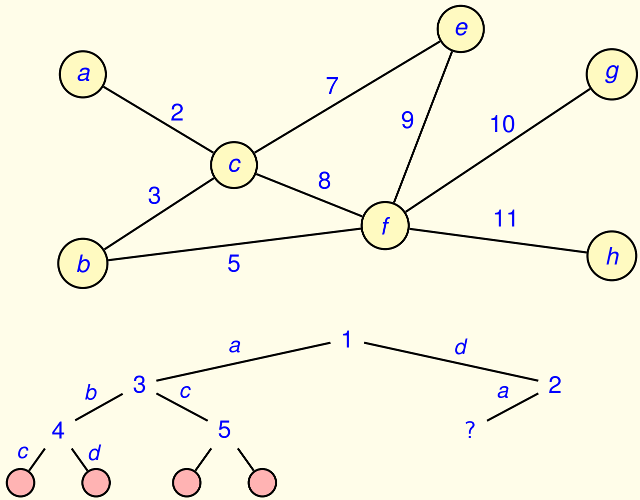
a
b
c
d
e
f
g
h
1
2
3
4
5
6
7
8
9
10
11
1
3
4
5
?
a
d
b
c
c
d

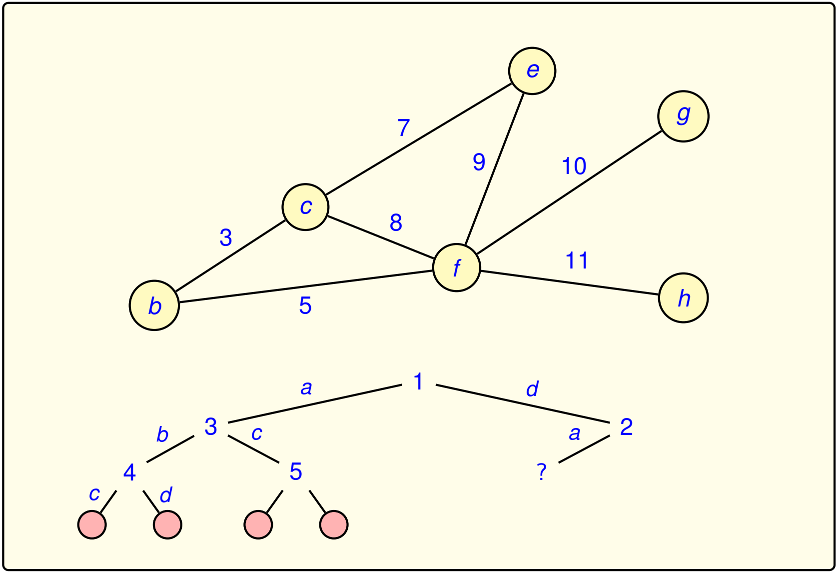

a
b
c
e
f
g
h
2
3
5
7
8
9
10
11
1
a
d
3
?
b
c
4
5
c
d

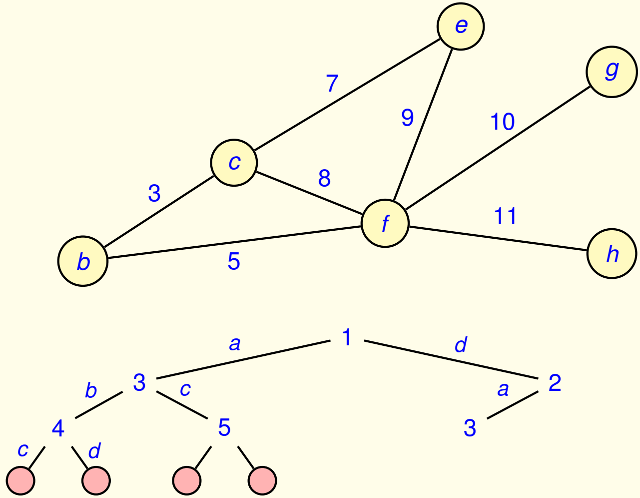
Solving 3-Vertex Cover

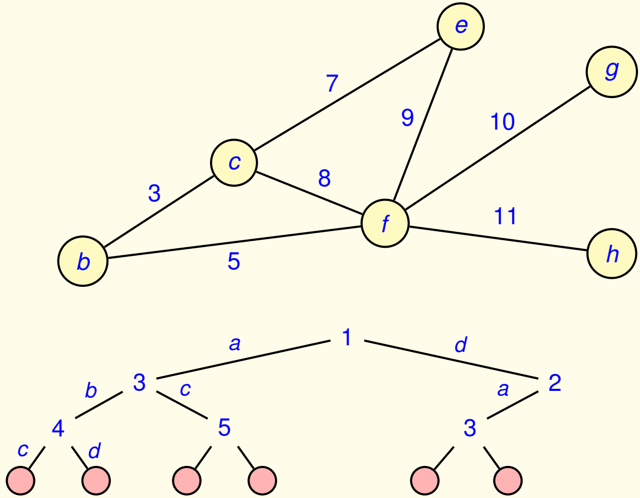


Solving 3-Vertex Cover

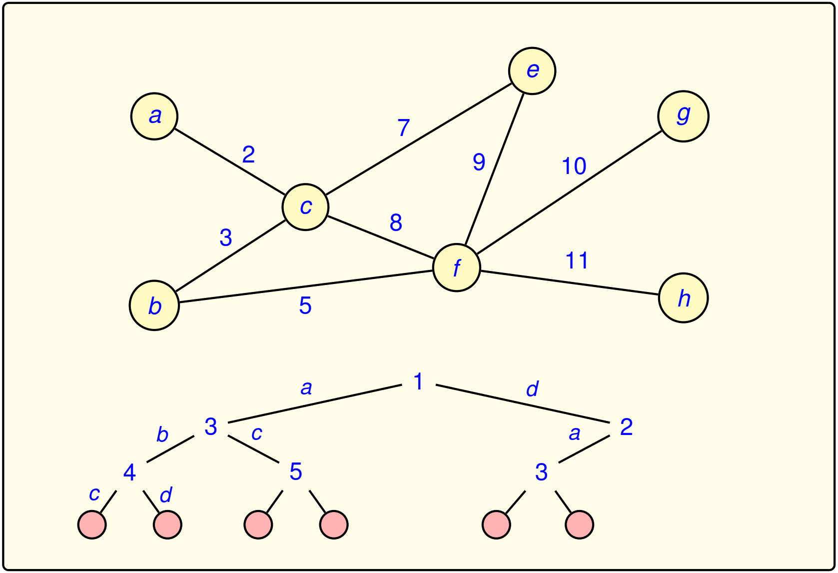



e
g
c
f
b
h
7
9
10
8
3
11
5
1
a
d
3
2
b
c
a
4
5
?
c
d

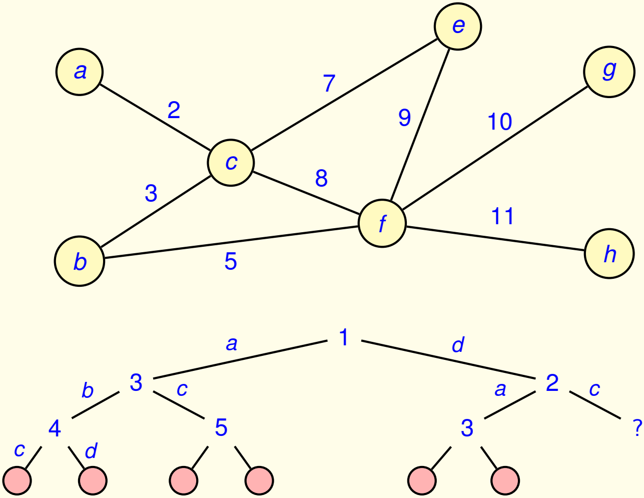

e
g
7
9
10
c
3
8
f
11
b
5
h
1
a
d
3
2
b
c
a
4
5
3
c
d


e
g
7
9
10
c
3
8
f
11
b
5
h
1
a
d
3
2
b
c
a
4
5
3
c
d

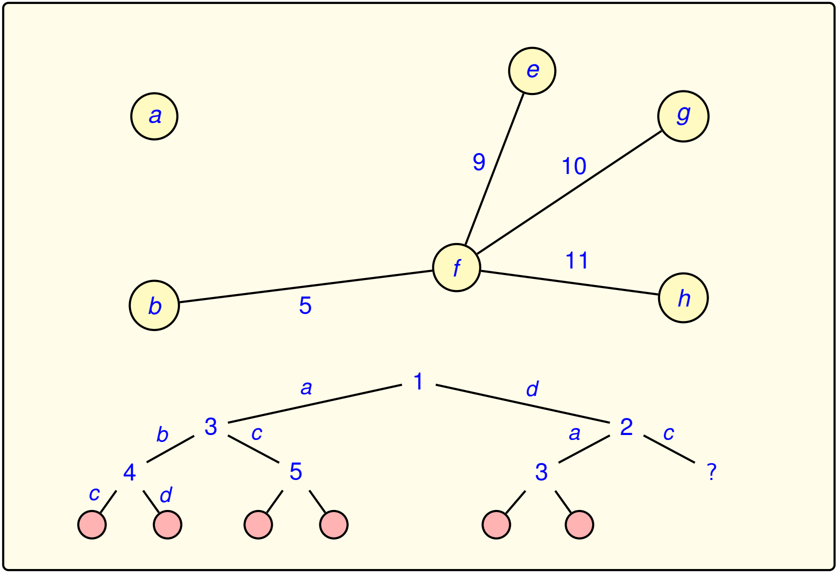
Solving 3-Vertex Cover

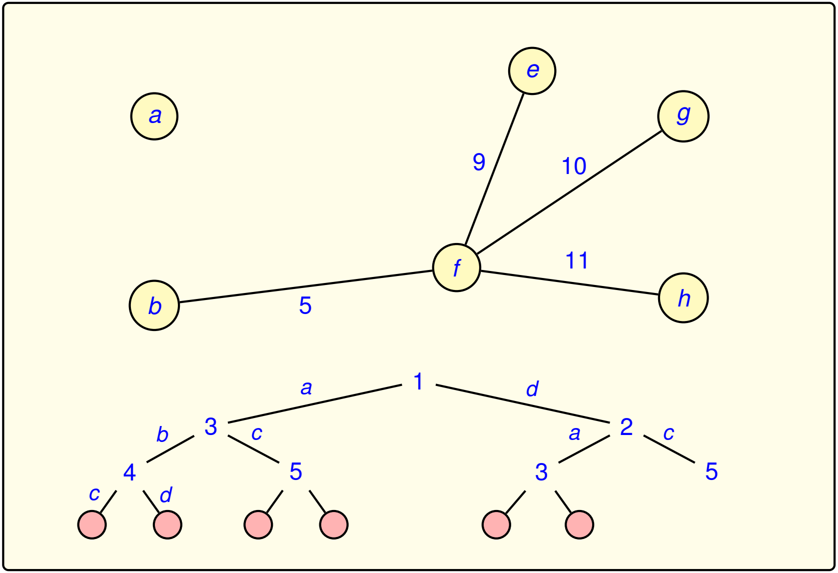


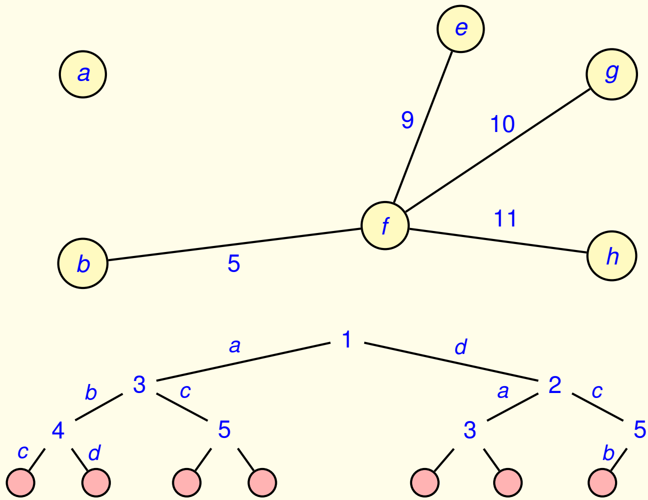
Solving 3-Vertex Cover



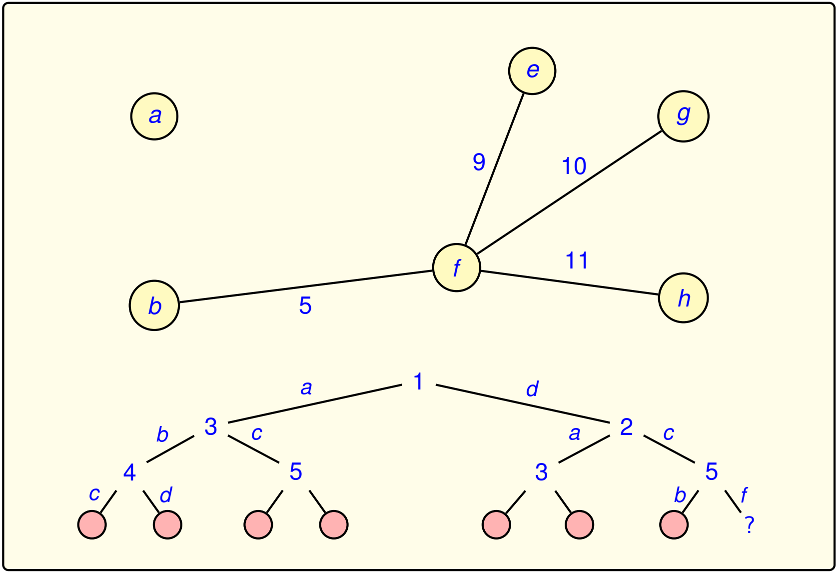
Solving 3-Vertex Cover



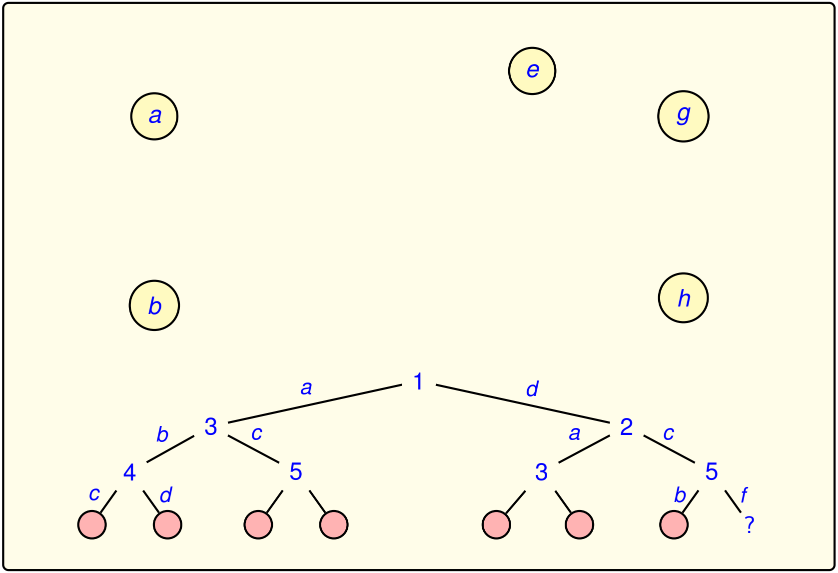

a
e
g
9
10
f
11
b
5
h
1
a
d
3
2
b
c
a
c
4
5
3
5
c
d


a
e
g
9
10
f
11
b
h
5
1
a
d
3
2
b
c
a
c
4
5
3
5
c
d
b

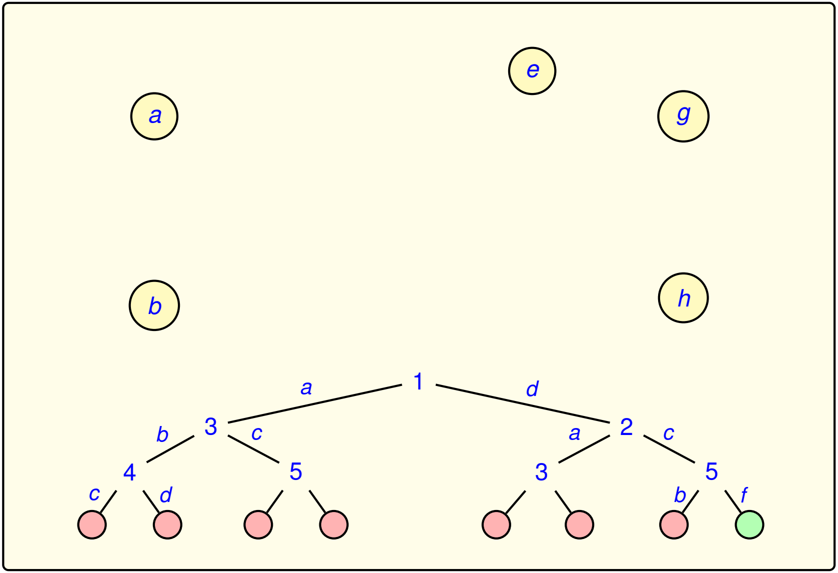
Solving 3-Vertex Cover

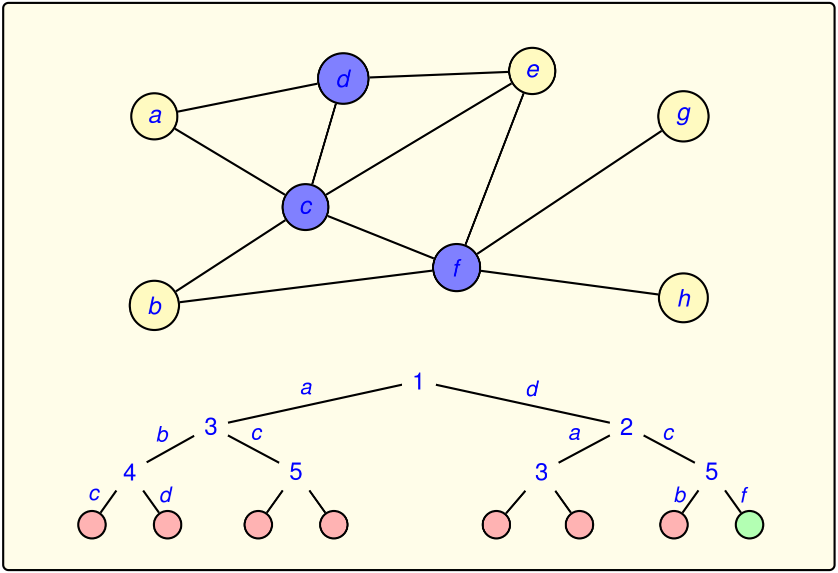


Solving 3-Vertex Cover



Solving 3-Vertex Cover




a
b
c
d
e
f
g
h
1
a
d
3
2
b
c
a
c
4
5
3
5
c
d
b
f

Given an instance (G, k) of VERTEX COVER, this bounded branching algorithm can solve it in time

$$2^k \cdot \mathcal{O}(|G|).$$

Given an instance (G, k) of VERTEX COVER, this bounded branching algorithm can solve it in time

$$2^k \cdot \mathcal{O}(|G|).$$

This should be compared to brute force algorithms for, e.g., CLIQUE that run in time

$$\Omega(|G|^k).$$

Given an instance (G, k) of VERTEX COVER, this bounded branching algorithm can solve it in time

$$2^k \cdot \mathcal{O}(|G|).$$

This should be compared to brute force algorithms for, e.g., CLIQUE that run in time

$$\Omega(|G|^k).$$

We say that VERTEX COVER *parameterized by k* is *fixed-parameter tractable*.

Definition. Let Σ be a finite alphabet. A *parameterization* of Σ^* is a mapping $\kappa : \Sigma^* \to \mathbb{N}$ that is computable in polynomial time.

Definition. Let Σ be a finite alphabet. A *parameterization* of Σ^* is a mapping $\kappa : \Sigma^* \to \mathbb{N}$ that is computable in polynomial time.

Example. We can parameterize VERTEX COVER by setting

$$\kappa(G, k) = k.$$

Definition. Let Σ be a finite alphabet. A *parameterized problem* over Σ is a pair (Q, κ) consisting of

- a set $Q \subseteq \Sigma^*$ of strings over Σ, and
- a parameterization κ of Σ^*.

Definition. Let Σ be a finite alphabet and κ a parameterization of Σ^*.

- An algorithm is *FPT* w.r.t. κ if there is a computable function $f : \mathbb{N} \to \mathbb{N}$ and a polynomial p such that for every $x \in \Sigma^*$, the algorithm, when given x, has running time at most
$$f(\kappa(x)) \cdot p(|x|).$$

Definition. Let Σ be a finite alphabet and κ a parameterization of Σ.

A parameterized problem (Q, κ) over Σ is *fixed-parameter tractable* if there is an FPT-algorithm w.r.t. κ that decides Q.

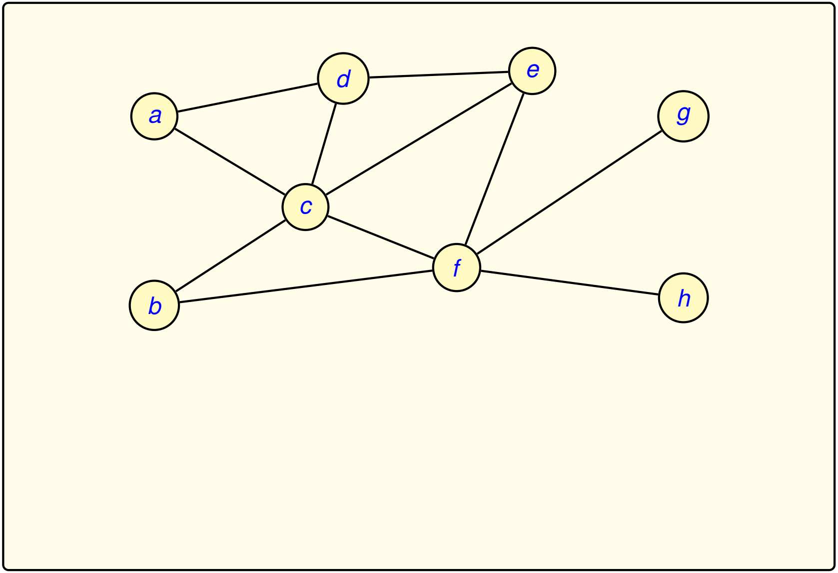
Definition. Let Σ be a finite alphabet and κ a parameterization of Σ.

A parameterized problem (Q, κ) over Σ is *fixed-parameter tractable* if there is an FPT-algorithm w.r.t. κ that decides Q.

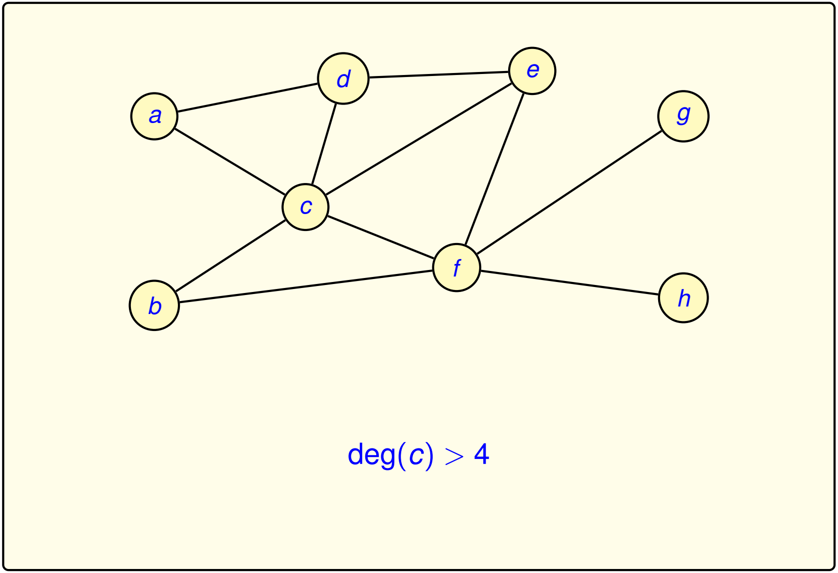
FPT is the class of all fixed-parameter tractable problems.

$\Rightarrow$

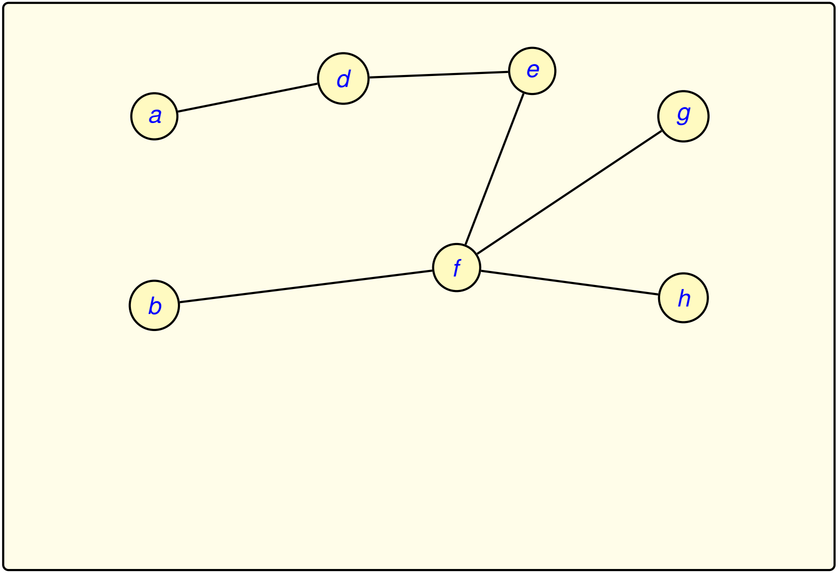
$k = 4$



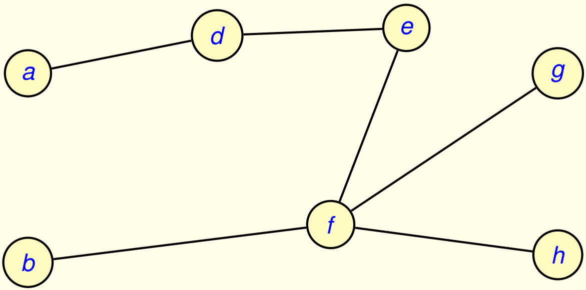
$k = 4$



$k = 3$

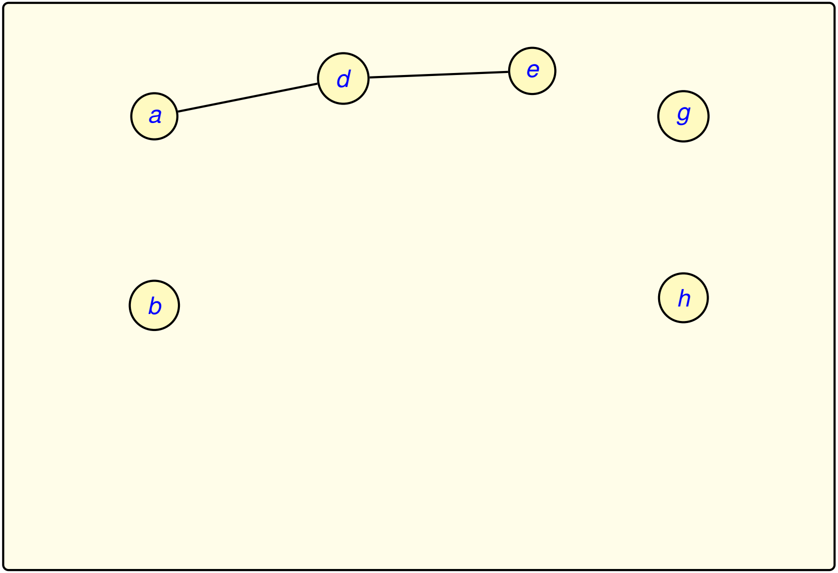


$k = 3$

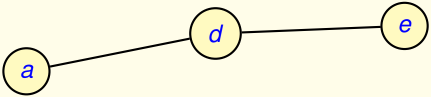


$\deg(f) > 3$

$k = 2$



$k = 2$



$|E| < 2^2$

More generally: *exhaustively applying the 3 reduction rules* transforms any VERTEX COVER instance into one where every vertex has a degree $2 \leq d \leq k$

More generally: *exhaustively applying the 3 reduction rules* transforms any VERTEX COVER instance into one where every vertex has a degree $2 \leq d \leq k$

It does so in *polynomial time in* $|G|$

More generally: *exhaustively applying the 3 reduction rules* transforms any VERTEX COVER instance into one where every vertex has a degree $2 \leq d \leq k$

It does so in *polynomial time in* $|G|$

Combinatorics show that a k-coverable graph where all vertices have degree $2 \leq d \leq k$ cannot have more than k^2 vertices

More generally: *exhaustively applying the 3 reduction rules* transforms any Vertex Cover instance into one where every vertex has a degree $2 \leq d \leq k$

It does so in *polynomial time in* $|G|$

Combinatorics show that a k-coverable graph where all vertices have degree $2 \leq d \leq k$ cannot have more than k^2 vertices

A *polynomial procedure* making *the whole problem* work in terms of k!

Applying the bounded branching algorithm to a reduced graph gives us $\mathcal{O}(1.2738^k)k^2$

Let (Q, κ) be a parameterized problem over Σ.
A *kernelization* of (Q, κ) is a mapping

$$K : \Sigma^* \rightarrow \Sigma^*$$

such that

- $x \in Q \Leftrightarrow K(x) \in Q$,
- there is a computable function g such that $|K(x)| < g(\kappa(x))$.

Let (Q, κ) be a parameterized problem over Σ.
A *kernelization* of (Q, κ) is a mapping

$$K : \Sigma^* \to \Sigma^*$$

such that

- $x \in Q \Leftrightarrow K(x) \in Q$,
- there is a computable function g such that $|K(x)| < g(\kappa(x))$.

Theorem

If (Q, κ) has a polynomial-time computable kernelization, then $(Q, \kappa) \in FPT$.

Let (Q, κ) be a parameterized problem over Σ.
A *kernelization* of (Q, κ) is a mapping

$$K : \Sigma^* \to \Sigma^*$$

such that

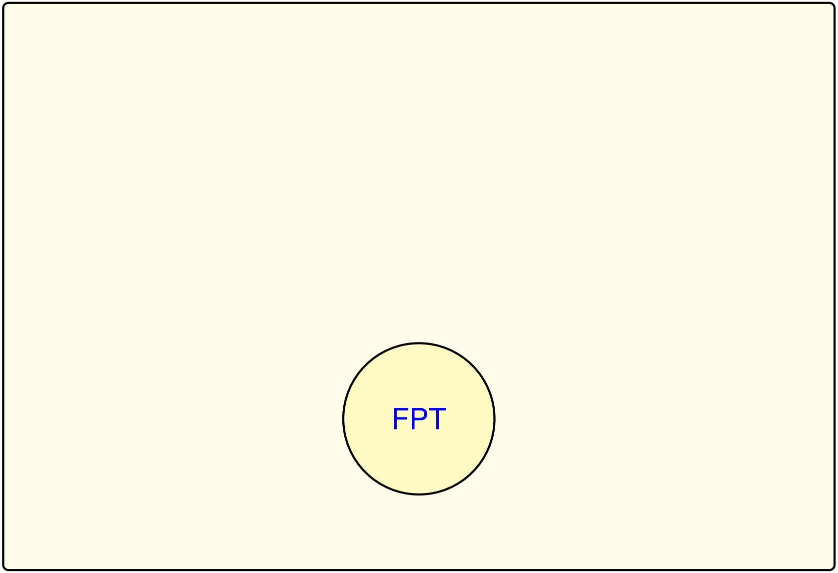
- $x \in Q \Leftrightarrow K(x) \in Q$,
- there is a computable function g such that $|K(x)| < g(\kappa(x))$.

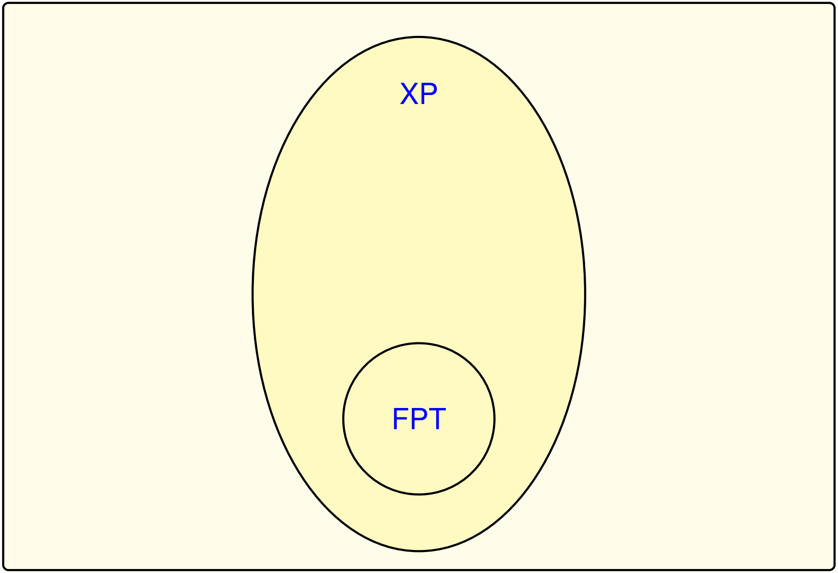
Theorem

If (Q, κ) has a polynomial-time computable kernelization, then $(Q, \kappa) \in FPT$.

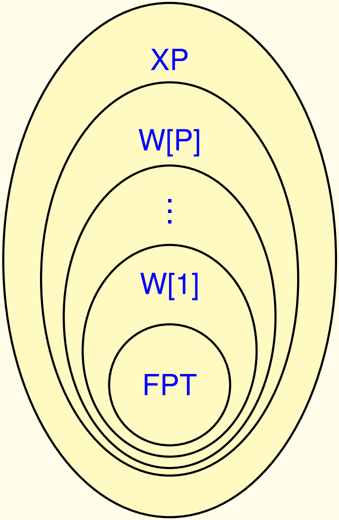
Theorem (!)

If $(Q, \kappa) \in FPT$ then (Q, κ) has a polynomial-time computable kernelization.


FPT


XP
FPT

The parameterized hierarchy



Part III: Proving Limited Parameterizability

To show that a problem P is *hard*:

To show that a problem P is *hard*:

- Pick a problem P' we *already know* is hard

To show that a problem P is *hard*:

- Pick a problem P' we *already know* is hard
- Create a *reduction* which takes any problem $p' \in P'$ and constructs a problem $p \in P$ which has the same answer as p'

To show that a problem P is *hard*:

- Pick a problem P' we *already know* is hard
- Create a *reduction* which takes any problem $p' \in P'$ and constructs a problem $p \in P$ which has the same answer as p'

1. The translation procedure can't be too powerful (or it might just solve p'!)

To show that a problem P is *hard*:

- Pick a problem P' we *already know* is hard
- Create a *reduction* which takes any problem $p' \in P'$ and constructs a problem $p \in P$ which has the same answer as p'

1. The translation procedure can't be too powerful (or it might just solve p'!)
2. p can't be too much *larger* than p' (or p becomes easy in *in terms of its size!*)

Reductions

To show that a problem P is *hard*:

- Pick a problem P' we *already know* is hard
- Create a *reduction* which takes any problem $p' \in P'$ and constructs a problem $p \in P$ which has the same answer as p'

  1. The translation procedure can't be too powerful (or it might just solve p'!)
  2. p can't be too much *larger* than p' (or p becomes easy in *in terms of its size!*)

- For P and NP: any reduction which runs in P solves both

To show that a problem P is *hard*:

- Pick a problem P' we *already know* is hard
- Create a *reduction* which takes any problem $p' \in P'$ and constructs a problem $p \in P$ which has the same answer as p'

  1. The translation procedure can't be too powerful (or it might just solve p'!)
  2. p can't be too much *larger* than p' (or p becomes easy in *in terms of its size!*)

- For P and NP: any reduction which runs in P solves both
- For parameterized complexity more care is needed

Definition. Let (Q, κ) and (Q', κ') be two parameterized problems over Σ and Γ.

An *FPT-reduction* from (Q, κ) to (Q', κ') is a mapping $R : \Sigma^* \to \Gamma^*$ such that

1. $x \in Q \Leftrightarrow R(x) \in Q'$, for all $x \in \Sigma^*$,
2. R is *FPT-computable* w.r.t. κ, and
3. there is a computable function g such that $\kappa'(R(x)) \leq g(\kappa(x))$ for all $x \in \Sigma^*$.

Demonstrating a Reduction

Unfortunately demonstrating a reduction gets complex

Demonstrating a Reduction

Unfortunately demonstrating a reduction gets complex

Suffice to say: k-Clique can be proven to be W[1]-hard

Demonstrating a Reduction

Unfortunately demonstrating a reduction gets complex

Suffice to say: k-Clique can be proven to be W[1]-hard

The top of the hierarchy, XP, is also interesting

The class XP

Definition. Let (Q, κ) be a parameterized problem over Σ. Then (Q, κ) belongs to *XP* if there is a function $f : \mathbb{N} \to \mathbb{N}$ and an algorithm that decides Q and runs on input $x \in \Sigma^*$ in time

$$|x|^{f(\kappa(x))} + f(\kappa(x)).$$

Definition. Let (Q, κ) be a parameterized problem over Σ. Then (Q, κ) belongs to *XP* if there is a function $f : \mathbb{N} \to \mathbb{N}$ and an algorithm that decides Q and runs on input $x \in \Sigma^*$ in time

$$|x|^{f(\kappa(x))} + f(\kappa(x)).$$

In other words, *XP* is the class of all *slice-wise polynomial* problems.

The class XP

Definition. Let (Q, κ) be a parameterized problem over Σ. Then (Q, κ) belongs to *XP* if there is a function $f : \mathbb{N} \to \mathbb{N}$ and an algorithm that decides Q and runs on input $x \in \Sigma^*$ in time

$$|x|^{f(\kappa(x))} + f(\kappa(x)).$$

In other words, *XP* is the class of all *slice-wise polynomial* problems.

Recall *slicing* as an island of tractability technique

The class XP

Definition. Let (Q, κ) be a parameterized problem over Σ. Then (Q, κ) belongs to *XP* if there is a function $f : \mathbb{N} \to \mathbb{N}$ and an algorithm that decides Q and runs on input $x \in \Sigma^*$ in time

$$|x|^{f(\kappa(x))} + f(\kappa(x)).$$

In other words, *XP* is the class of all *slice-wise polynomial* problems.

Recall *slicing* as an island of tractability technique

This should illustrate why it can be viewed as crude compared to parameterized complexity: *it is the worst case of the hierarchy*

Conclusion

This completes the small tour of tractability hunting

This completes the small tour of tractability hunting

Good books to read:

Computational Complexity by *Christos H. Paradimitriou*

An excellent treatment of the central concepts in computational complexity.

Parameterized Complexity Theory by *Jörg Flum and Martin Grohe*

A standard text on parameterized complexity theory.

This completes the small tour of tractability hunting

Good books to read:

Computational Complexity by *Christos H. Paradimitriou*

An excellent treatment of the central concepts in computational complexity.

Parameterized Complexity Theory by *Jörg Flum and Martin Grohe*

A standard text on parameterized complexity theory.

For now: *Thanks for listening and enjoy the rest of the course!*