NP-Completeness

Algorithmique
Fall semester 2011/12

What is an Algorithm?

We haven’t really answered this question in this course.

Informally, an algorithm is a step-by-step procedure for computing a set of outputs from a set of inputs.

But formally, to define what an algorithm is, we need to concept of a Turing Machine.

This cannot be developed fully in this course, and is typically part of a course in Theoretical Computer Science.

Will work with the informal concept instead.

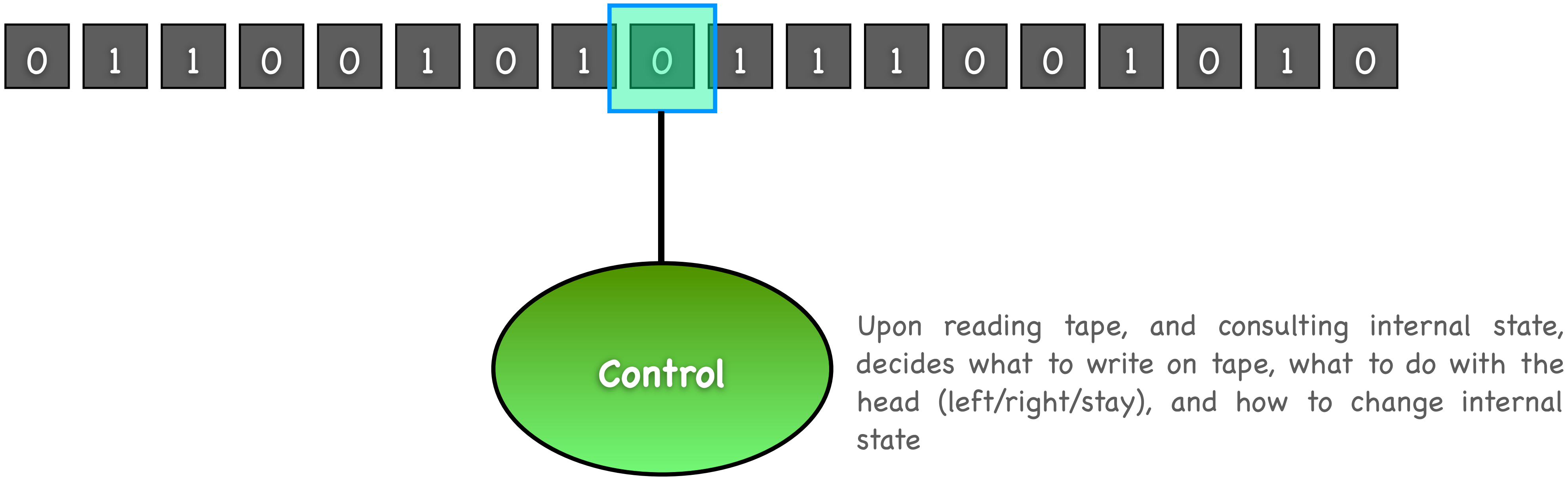
Turing Machine

Bare bone abstraction of a computer.

Consists of

- Input/Output alphabet (0-1)
- Infinite tape (memory)
- Read/Write head (I/O)
- Control (program)

Alan Turing, 1912-1954



Dictionary

Turing machine	Computer
Alphabet	0-1
Tape	Memory
Read/Write head	I/O
Control	Program
State	Program variables
# head movements	"Running time"
# cells visited	Space requirement

Suited for electronic circuits

Turing Machines

Turing machines provide a very good abstraction of the notion of computing, and a framework for studying the complexity of computational problems.

They lie at the heart of the theory of NP-completeness, which we are going to discuss in this class.

However, for time reasons we will not be able to use them. Instead, we rely on an intuitive concept of an algorithm and its running time (as the number of “operations” they use).

Computational Problem

I Set of inputs

O Set of outputs

$R \subseteq I \times O$ Relational dependency

Example

I = set of graphs

O = {true,false}

(G,true) in R iff G is connected

(G,false) in R iff G is not connected

Computational problem:

Given $\iota \in I$, find $\omega \in O$, such that $(\iota, \omega) \in R$

Example

Given G, decide whether it is connected.

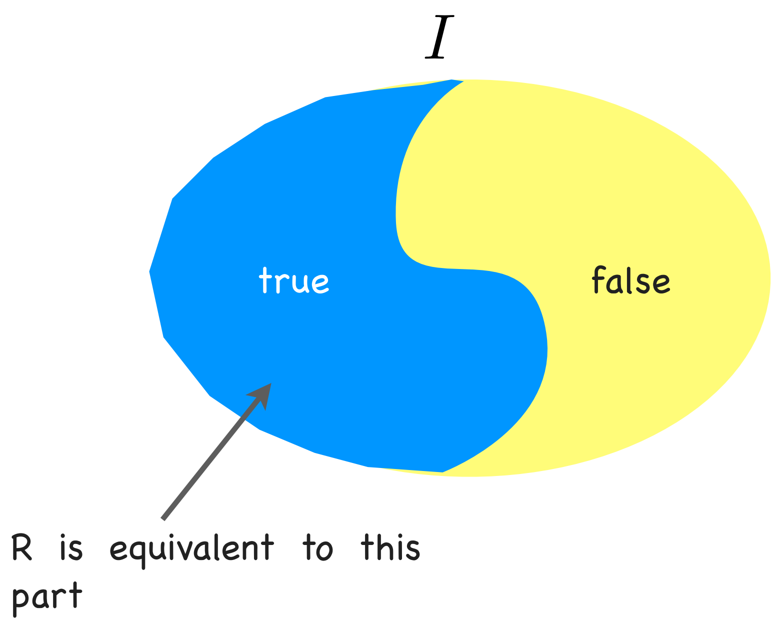
Decision Problem

I

$O = \{\text{true}, \text{false}\}$

$R \subseteq I \times O$

R is equivalent to subset of I mapping to true.



Examples

$$I = \mathbb{N}$$

$$R = \{n \in I \mid n \text{ is prime}\}$$

170141183460469231731687303715884105727 is in R

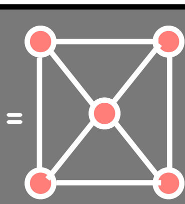
10384593717069655257060992658440191 is not in R

$$I = \text{set of graphs} \times \mathbb{N}$$

$$R = \{(G, n) \mid G \text{ has an independent set of size } \geq n\}$$

Independent set

Set of vertices no two of which are connected

G = 

(G,3) is not in R

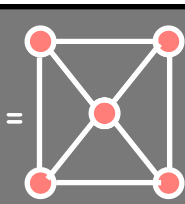
(G,2) is in R

$$I = \text{set of graphs} \times \mathbb{N}$$

$$R = \{(G, n) \mid G \text{ has a clique of size } \geq n\}$$

Clique

Set of vertices any two of which are connected

G = 

(G,4) is not in R

(G,3) is in R

Algorithms for Decision Problems

An algorithm for a decision problem $R \subseteq I$ is an algorithm which for every input $\iota \in I$ decides whether $\iota \in R$

Example

R = set of all connected graphs

Decision problem: given G, decide whether it is connected

Algorithm: DFS

Example

R = set of all prime numbers

Decision problem: given n, decide whether it is prime

Algorithm: primality testing

Example

R = {(G,n) | G graph, n integer, G has an independent set of size ≥ n}

Decision problem: decide whether G has independent set of size at least n

Algorithm: ?????

Input Lengths

The length $|\iota|$ of an input $\iota \in I$ is the number of bits sufficient to represent it.

Example

G graph on n vertices, $|G| = O(n^2)$

To represent G, we need n, and a list of n^2 bits representing the adjacency matrix of G

Example

n integer, $|n| = O(\log(n))$

Need $O(\log(n))$ bits to write down n

Polynomial Functions

A function $f : \mathbb{N} \to \mathbb{N}$ is said to be *polynomial* in $g : \mathbb{N} \to \mathbb{N}$ if there is a a polynomial p such that $f(n) = O(p(g(n)))$

Examples

- $f(n)=n$ is polynomial in $g(n) = n^2$
- $f(n) = n^3$ is polynomial in $g(n) = n$
- Any polynomial is polynomial in the function $f(n) = n$ (identity function)
- The identity function is polynomial in any polynomial function
- The function $f(n) = 2^n$ is not polynomial in the identity function

The Class P

Class of all decision problems $R \subseteq I$ for which there is an algorithm which decides for every given $\iota \in I$ whether $\iota \in R$ and for which the running time is polynomial in $|\iota|$

Example

CONNECTIVITY = {G | G is a connected graph}

CONNECTIVITY is in P (Running time of DFS is polynomial in number of vertices)

Example

I = {(G,s,t,c,M) | G directed graph, c:E -> N integer capacities, s,t distinct vertices, M integer}

FLOW = {(G,s,t,c,M) | G has an s-t flow of value at least M}

FLOW is in P:

- Size of input on n vertices is $\Theta(n^2 \log(C)+\log(M))$ where C is max capacity
- The algorithm of Ford and Fulkerson with BFS shortest path selection has running time polynomial in the input size.

P is the class of all decision problems for which there is a *polynomial time* decision algorithm.

Why Polynomial Time?

Captures the notion of “efficiency”

Is robust with respect to common theoretical transformations

The Class P

PRIMALITY = {n | n is a prime number}

PRIMALITY is in P ?

CLIQUE = {(G,n) | G has a clique of size at least n}

CLIQUE is in P ?

INDEP-SET = {(G,n) | G has an independent set of size at least n}

INDEP-SET is in P ?

The Class P

PRIMALITY = {n | n is a prime number}

PRIMALITY is in P ? Yes, but not via the naive algorithm

CLIQUE = {(G,n) | G has a clique of size at least n}

CLIQUE is in P ? Unknown

INDEP-SET = {(G,n) | G has an independent set of size at least n}

INDEP-SET is in P ? Unknown

NP Completeness

Theory of NP-completeness tries to understand why we have not been able to find polynomial time algorithms for some fundamental computational problems.

Its main assertions are of the following type:

If you could solve problem X in polynomial time, then could solve a whole lot of other very hard problems in polynomial time as well.

To some researchers, it means that there are no polynomial time algorithms for such problems.

Polynomial Reduction

We want to capture the following notion:

If we can solve decision problem X, then we can also solve decision problem Y using an algorithm for the solution of problem X. We want to reduce Y to X

How can we use an algorithm for problem X for inputs for problem Y?

We need to *transform* an input y for problem Y to a valid input x for problem X
Then we apply the algorithm for X to this new input x

If the output of the algorithm is true, then we want to deduce that y belongs to Y
If the output of the algorithm is false, then we want to deduce that y doesn't belong to Y

We want all this to be efficient, so the transformation of y to x should be efficiently computable.

Polynomial Reduction

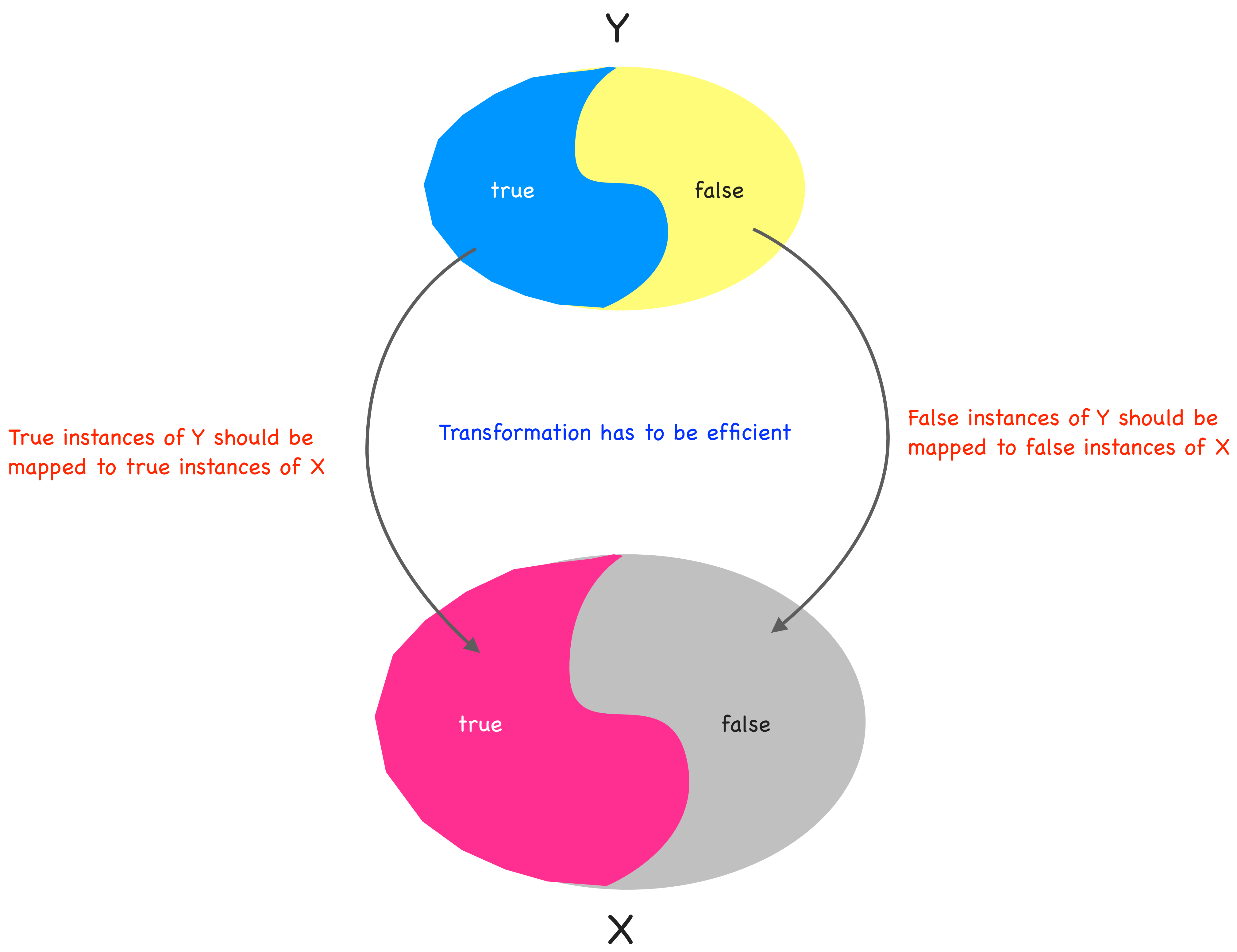
$X \subseteq I, Y \subseteq J$ decision problems.

A polynomial reduction from Y to X is a function $f : J \to I$ such that

- for all y in J, $f(y)$ is in X iff y is in Y
- There is an algorithm which computes $f(y)$ for any y in Y with a number of steps polynomial in $|y|$.

In this case, we write $Y \leq_P X$ or $Y \leq X$

Polynomial Reduction



Example

CLIQUE $\leq$ INDEP-SET

(G,m) input to CLIQUE

H complement of G:

- if there is no edge between u,v, add (u,v) to the edges of H
- If there is an edge between (u,v) in G, don't put that edge in H

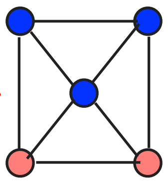
f: (G,m) -> (H,m) is the reduction

true only for general graphs.

f can be calculated in time polynomial in n, which is polynomial in $|(G,m)| = O(n^2+\log(m))$

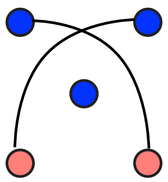
f(G,m) in INDEP-SET iff (H,m) in INDEP-SET iff there are m nodes in H no two of which are connected, iff there are m nodes in G any two of which are connected, iff (G,m) in CLIQUE

G



Clique

H



Independent

INDEP-SET $\leq$ CLIQUE

Same reasoning as above

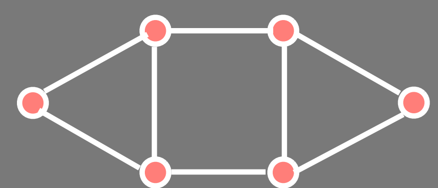
Testing membership to INDEP-SET is "as difficult" as testing membership to "CLIQUE"

Example

A *node cover* in a graph is a set S of vertices such that every edge has at least one endpoint in S.

NODE-COVER = {(G,m) | G has a node cover with at most m elements}

G =



(G,3) is not in NODE-COVER

(G,4) is in NODE-COVER

INDEP-SET $\leq$ NODE-COVER

(G,m) input to INDEP-SET

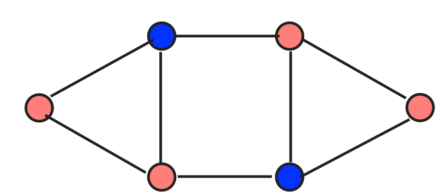
f: (G,m) -> (G,n-m) is the reduction where n is number of vertices of G

f can be calculated in time polynomial in the input

f(G,m) in NODE-COVER iff (G,n-m) in NODE-COVER iff there are n-m nodes in G such that any edge has at least one endpoint in this set iff any two nodes outside this set are not connected iff (G,m) is in INDEP-SET

Independent

G



Node cover

NODE-COVER $\leq$ INDEP-SET

similar to the above

Transitivity

$$X \leq Y \text{ and } Y \leq Z \implies X \leq Z$$

Suppose that f is a reduction from X to Y, so a is in X iff f(a) is in Y

Suppose that g is a reduction from Y to Z, so b is in Y iff g(b) is in Z

Let h(a) = g(f(a)).

a is in x iff f(a) is in Y iff g(f(a)) is in Z.

h can be computed efficiently, since g and f can.

So, h is a polynomial reduction from X to Z.

The Class NP

Some decision problems, like NODE-COVER and INDEPENDENT-SET have defied attempts at finding efficient decision algorithms.

However, these problems have an interesting feature: while it may be difficult to decide for every input whether it belongs to these sets, it is rather easy to *prove* the claim that an input belongs to these sets.

Examples

COMPOSITES = {n | n is not prime}

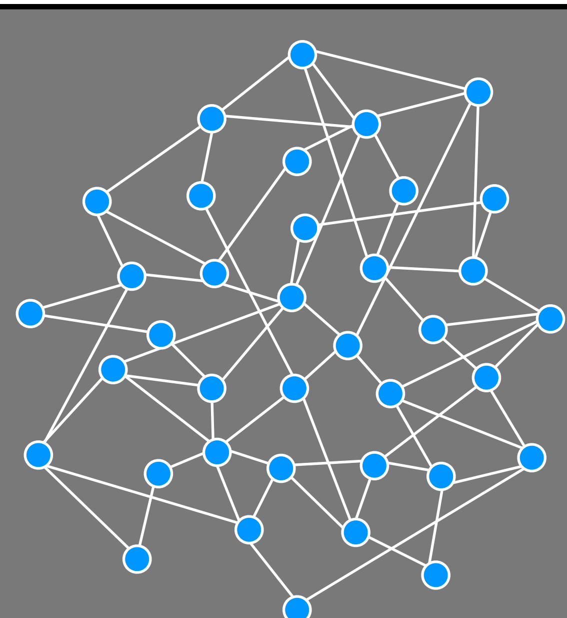
2391803447024455176592534890675171580150168271695179737339732O9 is in COMPOSITES!

You don't need to factor the number to see that I am right: I provide you with a "short" proof:

2391803447024455176592534890675171580150168271695179737339732O9 =

15465456498352886242205023347881 * 15465456498352886242205023347889

You just need to perform the multiplication to get convinced (which is much easier than factoring the integer).

G = 

(G,10) is in INDEP-SET

Finding an independent set of size 10 may be difficult.

Examples

COMPOSITES = {n | n is not prime}

239180344702445517659253489067517158015016827169517973733973209 is in COMPOSITES!

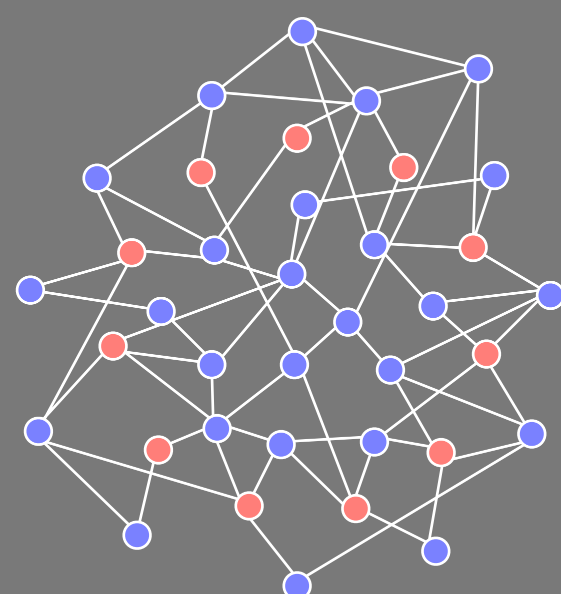
You don't need to factor the number to see that I am right: I provide you with a "short" proof:

239180344702445517659253489067517158015016827169517973733973209 =

154654564983528862422050233347881 * 154654564983528862422050233347889

You just need to perform the multiplication to get convinced (which is much easier than factoring the integer).

G =



(G,10) is in INDEP-SET

Finding an independent set of size 10 may be difficult.

But proving that I have one is simple.

The Class NP

Decision problems in the class NP are problems R for which there is an efficient *proof* of membership to R.

There is a function $f : I \times \{0,1\}^* \rightarrow \{\text{true},\text{false}\}$ such that

$$R = \{\ c \mid f(c,w)=\text{true for some string } w\}$$

and $f(c,w)$ can be calculated in time polynomial in $|c|$. f is called a *verifier*, and w is called a *witness* of membership of c to R.

Closed under Polynomial Reductions

If X is in NP, and $Y \leq X$, then Y is in NP as well.

Use reduction f from Y to X.

Take any input a from the set of inputs of Y.

Transform via f -> f(a)

There is witness w and poly-time function g such that g(f(a),w)=1 if f(a) is in X

w is a witness for a, g together with f is a verifier.

Includes P

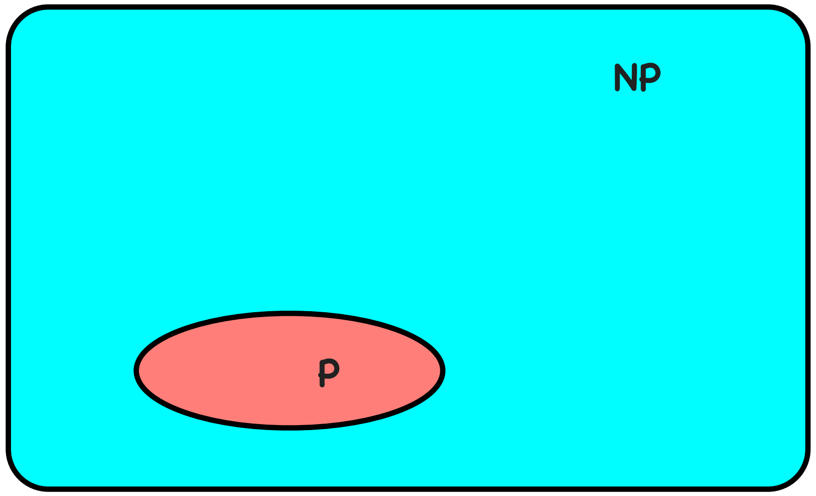
P is a subset of NP

Suppose that R is in P, and that f is a poly-time algorithm for R.

Given a, the witness w is empty, and the verifier is f itself: note that f(a)=1 iff a is in R.

Are P and NP equal?

Million dollar question.....

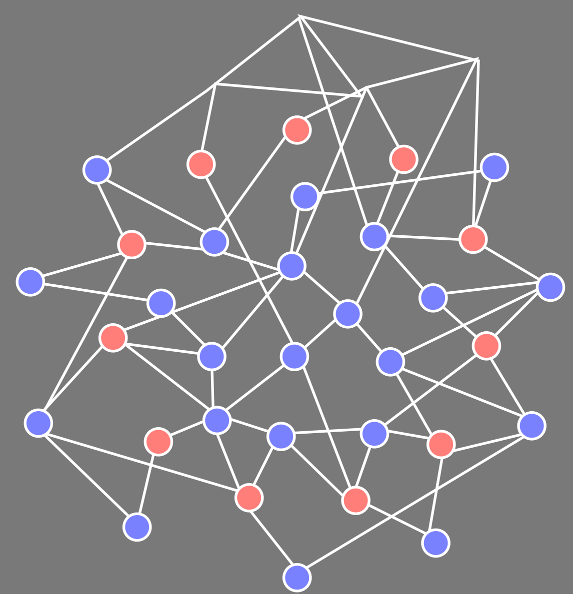


World according to some (top) researchers

Examples

$$\text{INDEP-SET} \in \text{NP}$$

G =



(G,10) is in INDEP-SET

The witness is a set of 10 or more vertices.

The verifier checks whether none of these vertices are connected.

The verifier can do this in time polynomial in the length of the graph.

$$\text{NODE-COVER} \leq \text{INDEP-SET} \implies \text{NODE-COVER} \in \text{NP}$$

$$\text{CLIQUE} \leq \text{INDEP-SET} \implies \text{CLIQUE} \in \text{NP}$$

Examples

$$\text{COMPOSITES} \in \text{NP}$$

COMPOSITES = {n | n is not prime}

2391803447024455176592534890675171580150168271695179737339732O9 is in COMPOSITES!

The witness consists of two integers larger than 1

15465456498352886242205023347881 , 15465456498352886242205023347889

The verifier multiplies these numbers and checks whether the result is the original number

The verifier can perform this task in time polynomial in the number of digits of the original number (using the school method)

PRIMES

$$\text{PRIMES} \in \text{NP?}$$

Given an integer n, how can I convince you that it is prime?

What is a witness of primality? What is the verifier?

What is NOT a verifier is an algorithm that checks all the potential divisors of n.

This is because this algorithm runs in time proportional to sqrt(n), which is NOT polynomial in the input length |n| = O(log(n)).

What can be done?

Elementary Number Theory

p is prime if and only if there is an integer w such that $w^{(p-1)/p_1}$, $w^{(p-1)/p_2}$, ..., $w^{(p-1)/p_k}$ is not 1 modulo p, where $p_1, p_2, \ldots, p_k$ are all the prime divisors of p-1

Not difficult, but without proof.

"Witness" for the primality of p:

w, and $p_1,\ldots,p_k$.

Steps for verifying primality of p:

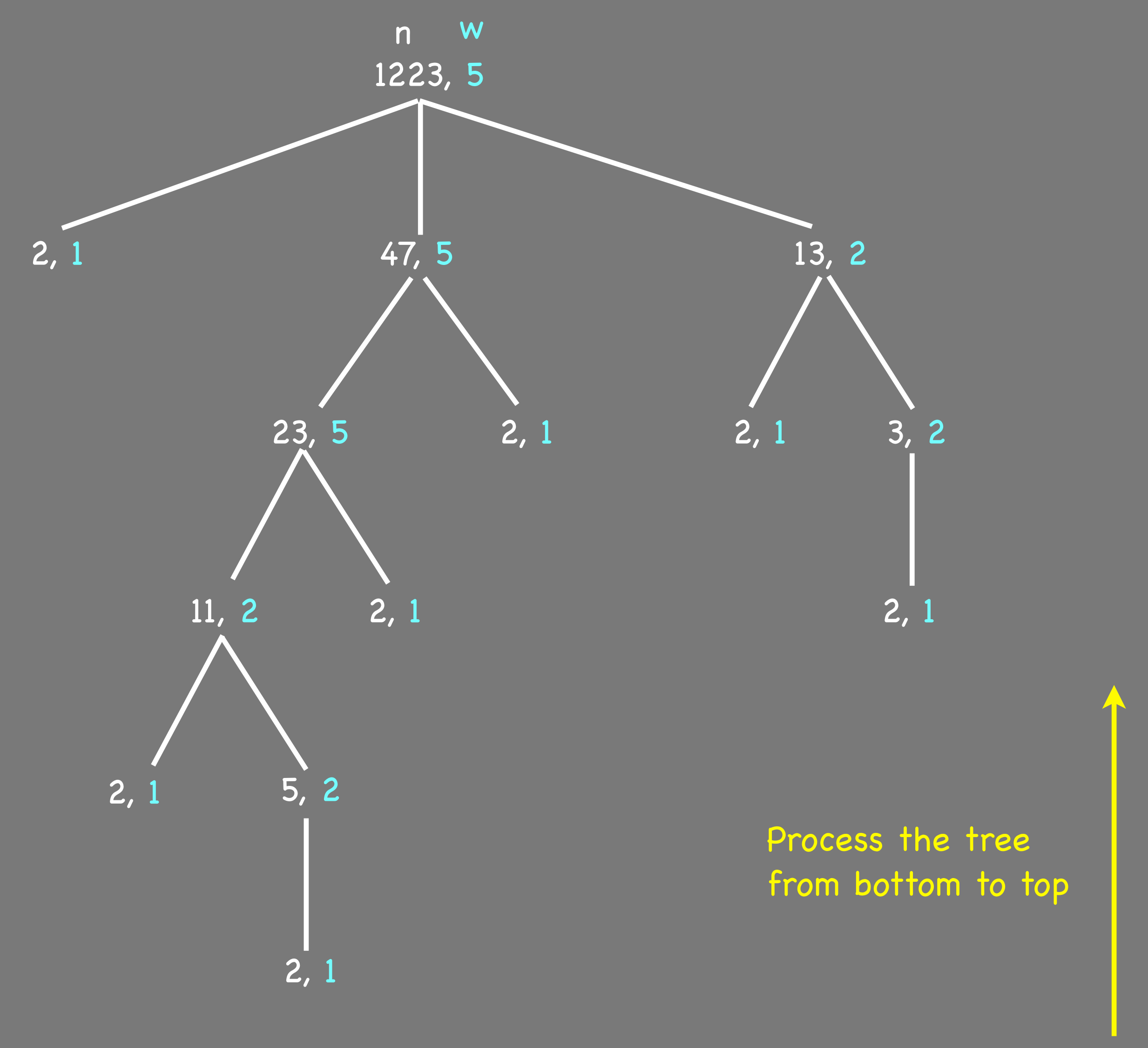
- Check that $p_1,\ldots,p_k$ are all prime The primality of these has to be proved recursively
- Check that $p_1,\ldots,p_k$ are all the prime divisors of p-1 Easy to do (division)
- Check that $w^{(p-1)/p_1}, \ldots, w^{(p-1)/p_k} \not\equiv 1 \bmod p$ Easy to do (binary method of exponentiation)

Witness is a tree with node labels (called the Pratt tree)

Verifier processes the tree to obtain proof of primality

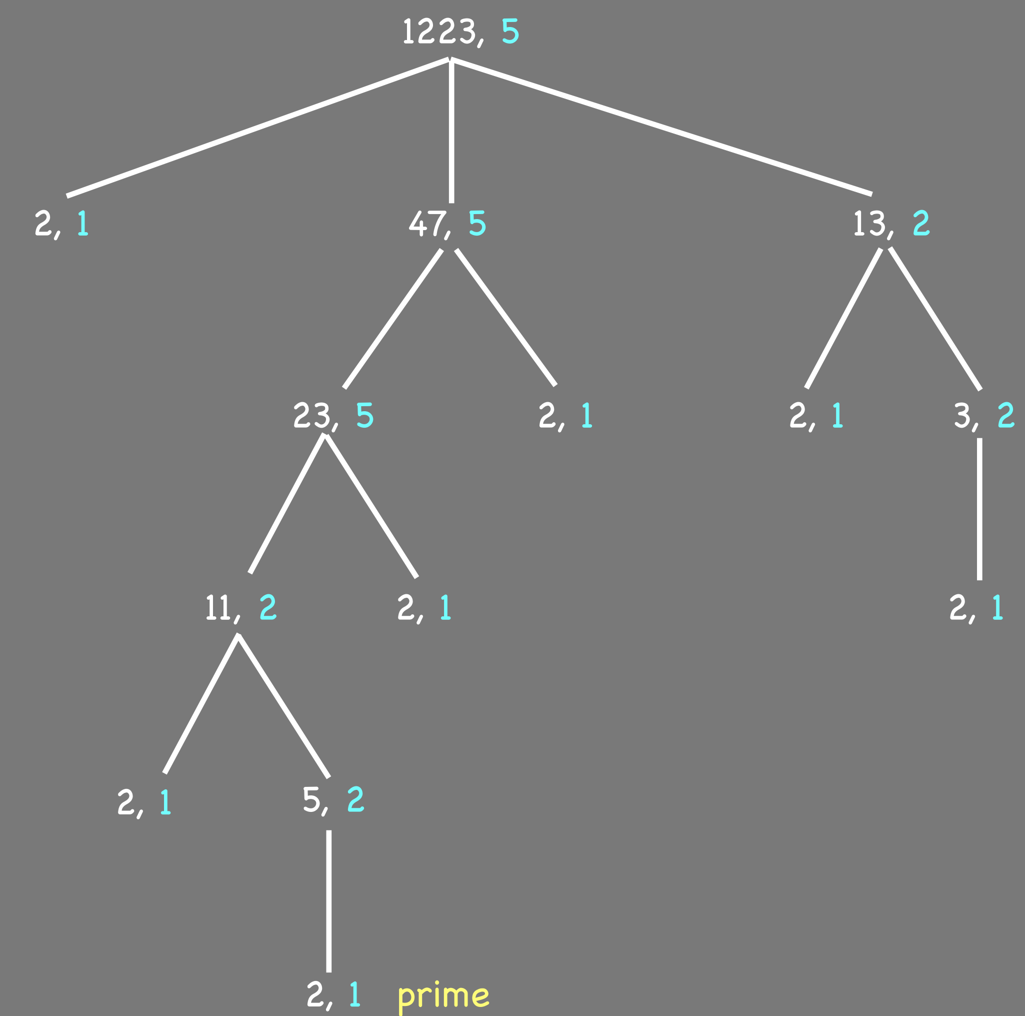
Vaughan Pratt, 1944 -

Example: Pratt Tree



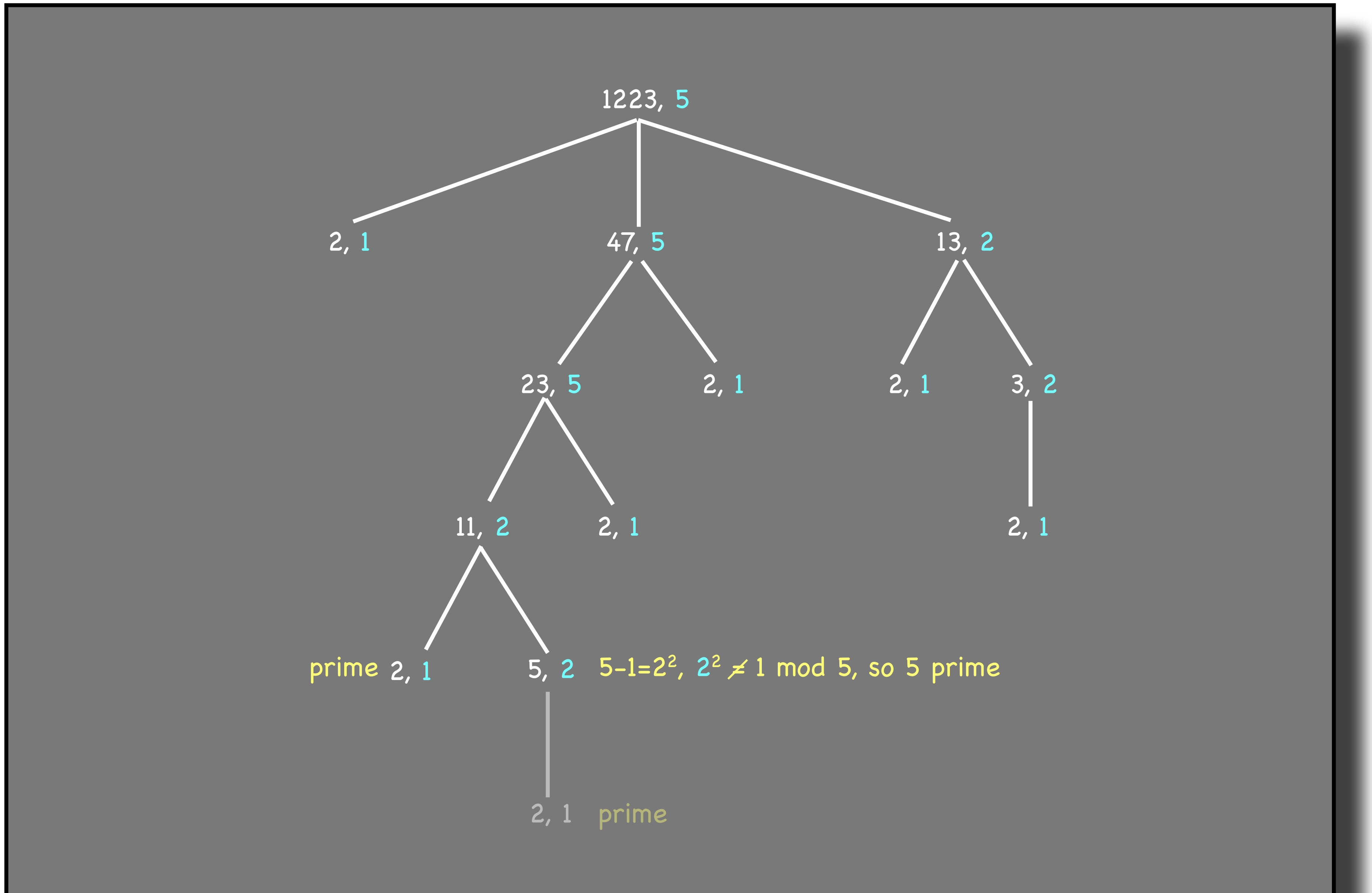
Pratt tree proving that 1223 is prime

Example: Pratt Tree



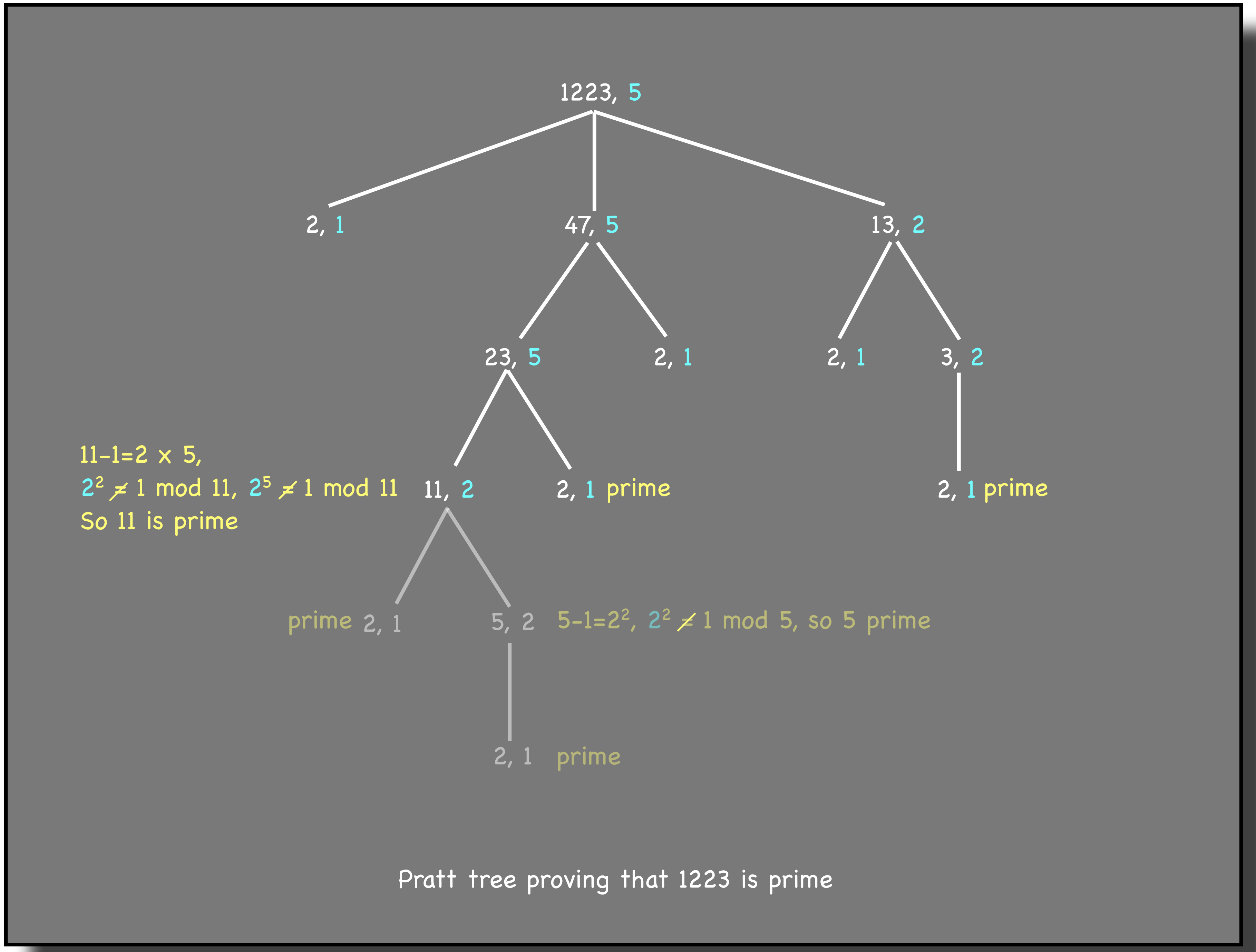
Pratt tree proving that 1223 is prime

Example: Pratt Tree

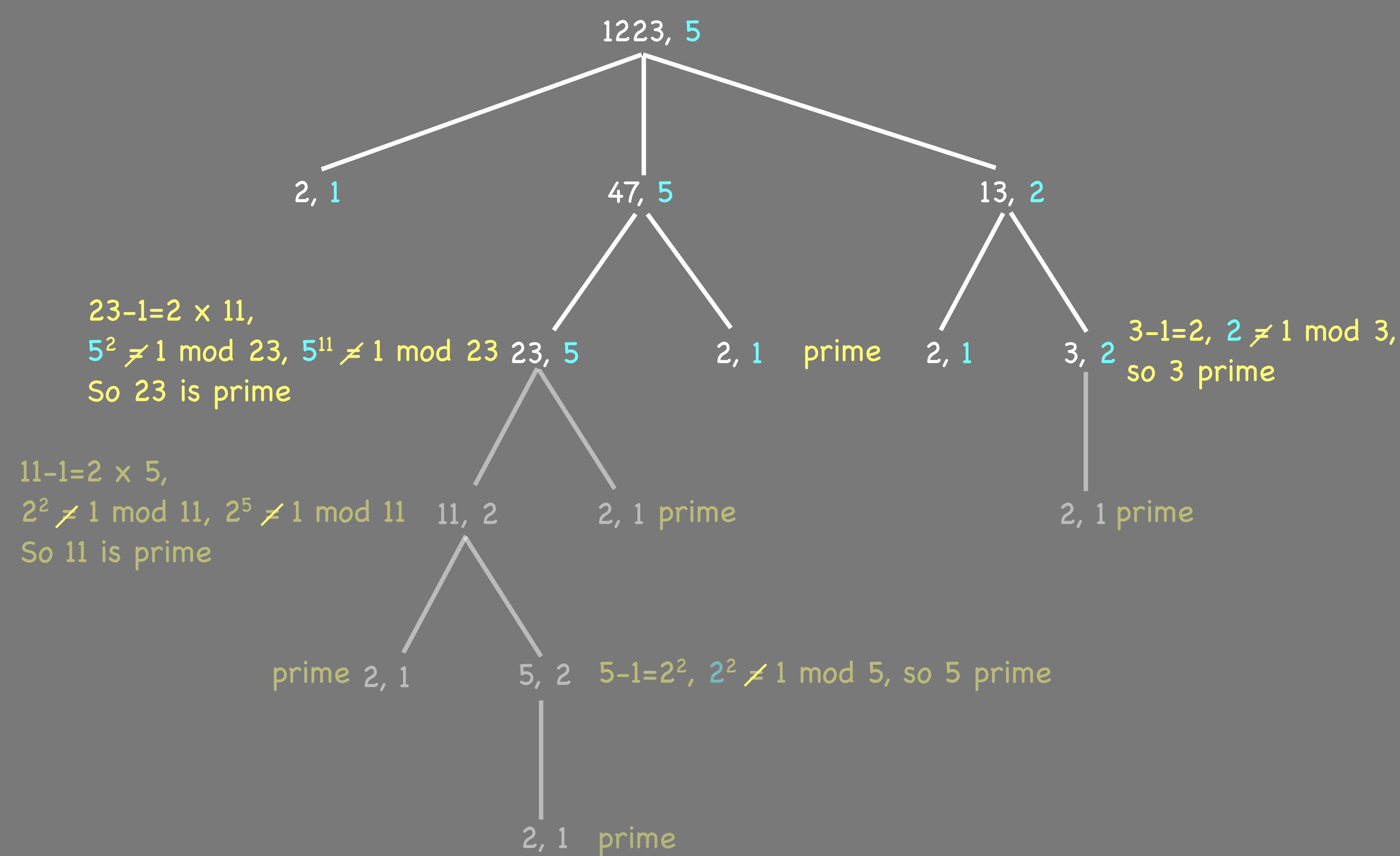


Pratt tree proving that 1223 is prime

Example: Pratt Tree

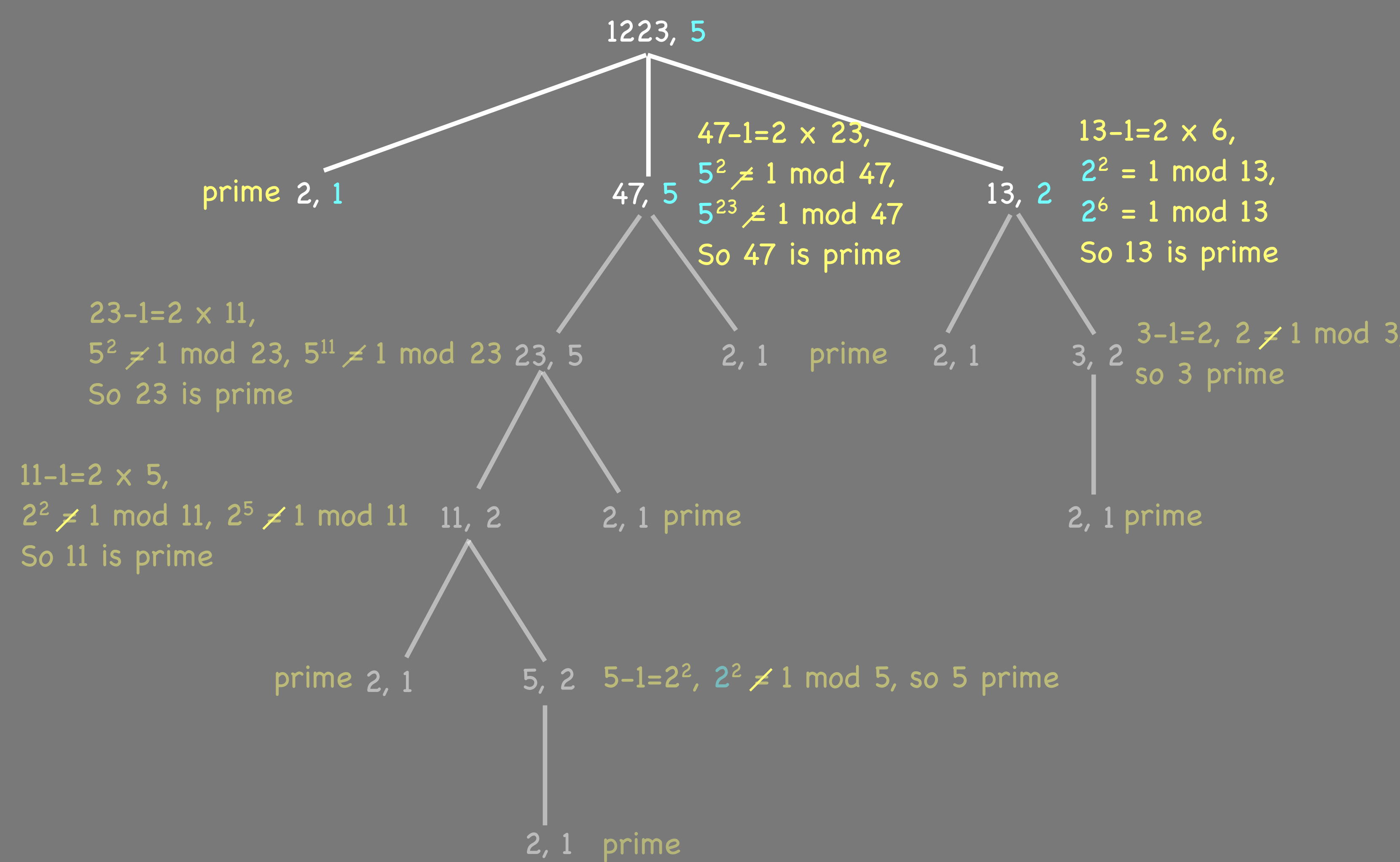


Example: Pratt Tree



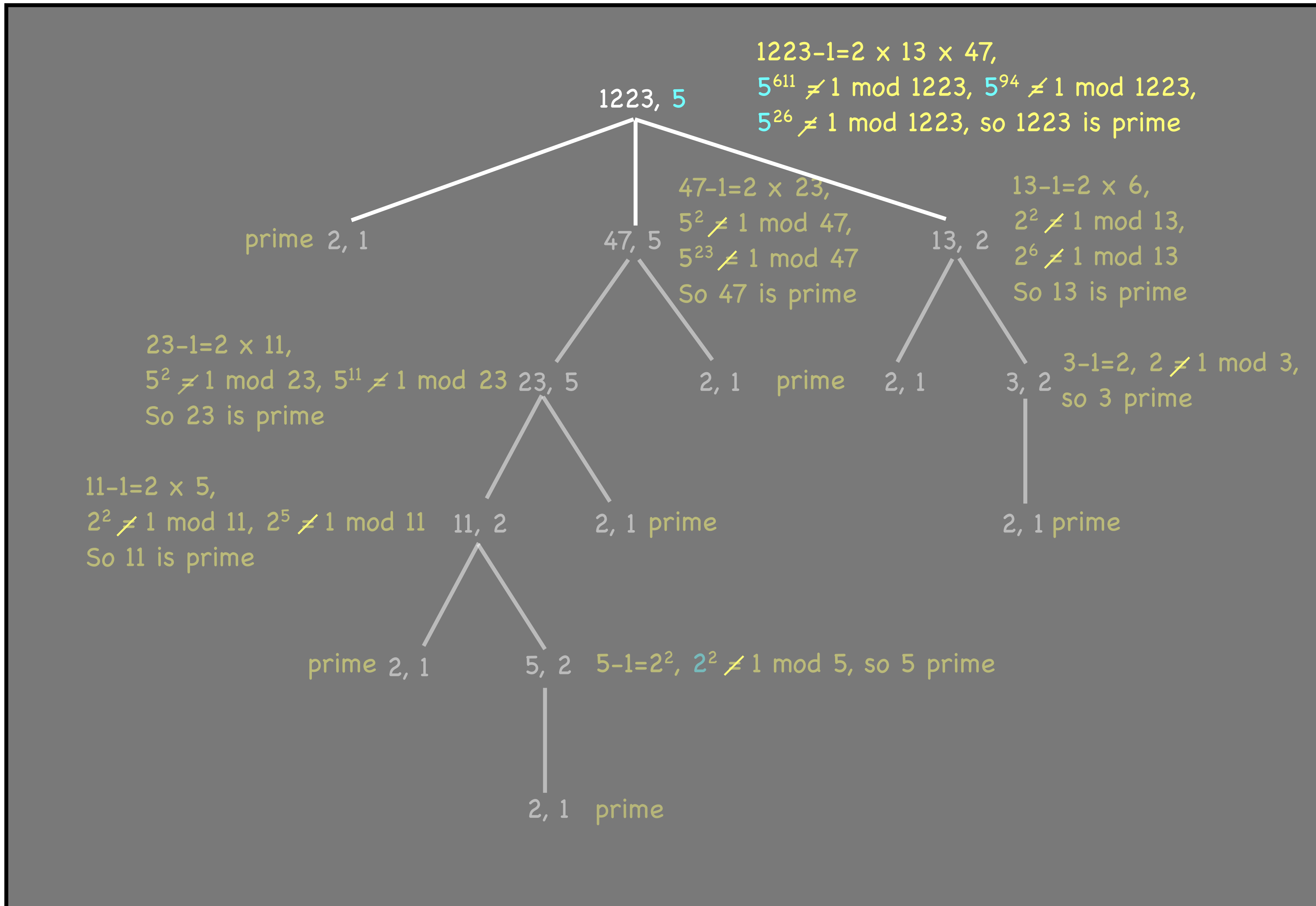
Pratt tree proving that 1223 is prime

Example: Pratt Tree



Pratt tree proving that 1223 is prime

Example: Pratt Tree



Pratt tree proving that 1223 is prime

PRIMES in NP

Careful analysis shows that:

- Pratt tree has size polynomial in $|n| = O(\log(n))$
- Verification can be done in time polynomial in $\log(n)$.

$$\text{PRIMES} \in \text{NP}$$

NP-Completeness

A decision problem R is called <u>*NP-complete*</u> if

- R is in NP
- For *any* problem X in NP there is a reduction from X to R: $X \leq R$

Is this an empty definition?

It is not at all clear that such a problem even exists.

Such a problem is a *hardest* problem in NP (if you can solve it efficiently, then you can solve any problem in NP efficiently)

Do such problems exist?

A Little Logic

A *boolean variable* x is a variable that can take values in the set {false,true} (or {0,1})

From boolean variables x and y we can obtain new variables via logical operations

x	y	$\neg x$	$x \vee y$	$x \wedge y$
0	0	1	0	0
0	1	1	1	0
1	0	0	1	0
1	1	0	1	1

$\neg x$ = not x

$x \vee y$ = x or y

$x \wedge y$ = x and y

Literals, Clauses, Formulas

Examples

Variable	$x_1, x_2, \ldots, x_n$	$x_1, x_3, \ldots$
Literal	$\lambda \in \{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$	$\neg x_5, x_{10}, \neg x_{220}, \ldots$
Clause	$C = \lambda_1 \vee \lambda_2 \vee \cdots \vee \lambda_t$	$x_1 \vee \neg x_3 \vee \neg x_{10} \vee x_4$
Formula	$F = C_1 \wedge C_2 \wedge \cdots \wedge C_\ell$	$(x_1 \vee \neg x_3 \vee \neg x_{10} \vee x_4) \wedge (x_2 \vee x_5 \vee x_9) \wedge (\neg x_6 \vee x_8)$

Length of a Formula

x_{10} → 0.1010 — Bit representation of 10 - The first “0.” is to say that the literal is not negated.

$\neg x_5$ → 1.101 — Bit representation of 5 - The first “1.” is to say that the literal is negated.

$x_1 \vee \neg x_3 \vee \neg x_{10} \vee x_4$ → 0.1*1.11*1.10101*0.100 — “*” indicates the OR operation

$(x_1 \vee \neg x_3 \vee \neg x_{10} \vee x_4)\ (x2 \vee x5 \vee x9) \wedge (\neg x6 \vee x8)$ →
0.1*1.11*1.10101*0.100|0.10*0.101*0.1001|1.110*0.100 — “|” indicates the AND operation

With n variables, length of formula is proportional to log(n) x number of literals

Satisfiability Problem

Formula F is called ***satisfiable*** if there is a setting of the variables that makes the formula evaluate to "true" (or 1).

Such a setting of the variables is called a ***satisfying assignment.***

$$(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3 \vee x_4) \wedge (\neg x_3 \vee x_4) \wedge (x_4 \vee x_2 \vee x_3)$$

has satisfying assignment

$$(x_1,x_2,x_3,x_4)=(0,0,0,1)$$

Checking whether a formula is satisfiable can be hard: there may be only 1 satisfying assignment (out of the 2^n possible).

The difference between this and 0 satisfying assignments is "very small".

Satisfiability Problem

SAT = { F | F satisfiable formula } is called the *satisfiability problem*

Associated computational problem: given F, check whether it is satisfiable.

SAT is in NP: "witness" for a satisfiable formula is a satisfying assignment. Formula can be evaluated at the given assignment in time polynomial in length of the formula.

The Cook-Levin Theorem

Stephen Cook, 1939 -

Leonid Levin, 1948 -

SAT is NP-complete.

If you can find an polynomial time algorithm for solving SAT, then you can solve other hard problems, like for example factoring integers, finding large independent sets, cliques, node covers, etc.

Are there Other NP-Complete Problems?

Many more than you might think....

k-clause: clause with exactly k literals

k-formula: formula in which every clause is a k-clause

k-SAT = $\{ F \mid F$ is a satisfiable k formula $\}$

Theorem: k-SAT is NP-complete for k larger than 2.

Example: 3-SAT

How do we prove that 3-SAT is NP-complete?

Step 1: Show that 3-SAT is in NP   It better; otherwise it cannot be NP-complete

Step 2: Show that SAT ≤ 3-SAT   For X in NP, we have X ≤ SAT ≤ 3-SAT, so X ≤ 3-SAT.

3-SAT is in NP: witness is satisfying assignment.

SAT ≤ 3-SAT: Need to find polynomial reduction from SAT to 3-SAT

Reduction of SAT to 3-SAT

Transform a formula F into a 3-formula G such that F is satisfiable iff G is.

Will do this clause-by-clause. Here an example when clause has ≥ 3 literals.

$C = (x_1 \lor x_5 \lor \neg x_6 \lor x_7 \lor x_8 \lor x_{12})$ Original clause

$F_1 = (x_1 \lor x_5 \lor \neg x_6 \lor x_7 \lor \neg y_1) \land (y_1 \lor x_8 \lor x_{12})$ Introduce new variable, take last two literals, and make new formula

F_1 is satisfiable iff C is:

C satisfiable ⇒ set y_1=0 is x_8 or x_{12} are 1, else set y_1=1 ⇒ F_1 satisfiable

F_1 satisfiable: if y_1=0, then x_1 or x_{12} are 1, else one of the other literals is 1 ⇒ C is satisfiable

Reduction of SAT to 3-SAT

Transform a formula F into a 3-formula G such that F is satisfiable iff G is.

Will do this clause-by-clause. Here an example when clause has ≥ 3 literals.

$C = (x_1 \vee x_5 \vee \neg x_6 \vee x_7 \vee x_8 \vee x_{12})$ Original clause

$F_1 = (x_1 \vee x_5 \vee \neg x_6 \vee x_7 \vee \neg y_1) \wedge (y_1 \vee x_8 \vee x_{12})$ Introduce new variable, take last two literals, and make new formula

F_1 is satisfiable iff C is:

C satisfiable $\Rightarrow$ set $y_1=0$ is x_8 or x_{12} are 1, else set $y_1=1$ $\Rightarrow$ F_1 satisfiable

F_1 satisfiable: if $y_1=0$, then x_1 or x_{12} are 1, else one of the other literals is 1 $\Rightarrow$ C is satisfiable

$F_2 = (x_1 \vee x_5 \vee \neg x_6 \vee \neg y_2) \wedge (y_2 \vee x_7 \vee \neg y_1) \wedge (y_1 \vee x_8 \vee x_{12})$ Repeat process

Reduction of SAT to 3-SAT

Transform a formula F into a 3-formula G such that F is satisfiable iff G is.

Will do this clause-by-clause. Here an example when clause has ≥ 3 literals.

$C = (x_1 \vee x_5 \vee \neg x_6 \vee x_7 \vee x_8 \vee x_{12})$ Original clause

$F_1 = (x_1 \vee x_5 \vee \neg x_6 \vee x_7 \vee \neg y_1) \wedge (y_1 \vee x_8 \vee x_{12})$ Introduce new variable, take last two literals, and make new formula

F_1 is satisfiable iff C is:
C satisfiable $\Rightarrow$ set $y_1=0$ is x_8 or x_{12} are 1, else set $y_1=1$ $\Rightarrow$ F_1 satisfiable
F_1 satisfiable: if $y_1=0$, then x_1 or x_{12} are 1, else one of the other literals is 1 $\Rightarrow$ C is satisfiable

$F_2 = (x_1 \vee x_5 \vee \neg x_6 \vee \neg y_2) \wedge (y_2 \vee x_7 \vee \neg y_1) \wedge (y_1 \vee x_8 \vee x_{12})$ Repeat process

$F_3 = (x_1 \vee x_5 \vee \neg y_3) \wedge (y_3 \neg x_6 \vee \neg y_2) \wedge (y_2 \vee x_7 \vee \neg y_1) \wedge (y_1 \vee x_8 \vee x_{12})$

F3 is now a 3-formula which is satisfiable iff C is

Reduction of SAT to 3-SAT

Transform a formula F into a 3-formula G such that F is satisfiable iff G is.

Will do this clause-by-clause. Here an example when clause has ≥ 3 literals.

$C = (x_1 \vee x_5 \vee \neg x_6 \vee x_7 \vee x_8 \vee x_{12})$ Original clause

$F_1 = (x_1 \vee x_5 \vee \neg x_6 \vee x_7 \vee \neg y_1) \wedge (y_1 \vee x_8 \vee x_{12})$ Introduce new variable, take last two literals, and make new formula

F_1 is satisfiable iff C is:
C satisfiable $\Rightarrow$ set y_1=0 is x_8 or x_{12} are 1, else set y_1=1 $\Rightarrow$ F_1 satisfiable
F_1 satisfiable: if y_1=0, then x_1 or x_{12} are 1, else one of the other literals is 1 $\Rightarrow$ C is satisfiable

$F_2 = (x_1 \vee x_5 \vee \neg x_6 \vee \neg y_2) \wedge (y_2 \vee x_7 \vee \neg y_1) \wedge (y_1 \vee x_8 \vee x_{12})$ Repeat process

$F_3 = (x_1 \vee x_5 \vee \neg y_3) \wedge (y_3 \neg x_6 \vee \neg y_2) \wedge (y_2 \vee x_7 \vee \neg y_1) \wedge (y_1 \vee x_8 \vee x_{12})$

F3 is now a 3-formula which is satisfiable iff C is

Repeat this process for all the clauses.

Each clause with $l \geq 3$ literals becomes formula with l-2 clauses and l-3 new variables.

Size of new formula is polynomial in size of old. Valid polynomial reduction.

Reduction of SAT to 3-SAT

Clauses with 1 or 2 literals:

$C = (x_1 \vee x_5)$ □ $(x_1 \vee x_5 \vee \neg y_1) \wedge (y_1 \vee x_1 \vee x_5)$

$C = \neg x_{10}$ □ $(\neg x_{10} \vee \neg y_1) \wedge (y_1 \vee \neg x_{10})$ Now reduce to previous case

Method works also for k-SAT, $k \geq 3$.

What about 2-SAT? It turns out that it is in P. Without proof.

MAX-2-SAT

2-SAT is in P.

But if a 2-formula is not satisfiable, can we efficiently find the maximum number of satisfiable clauses in the formula?

F is a 1-2-formula if all clauses in *F* have at most 2 literals.

MAX-2-SAT = $\{(F,m) \mid F$ 1-2-formula for which there is an assignment satisfying $\geq m$ clauses$\}$

$G = x \wedge y \wedge z \wedge w \wedge (\neg x \vee \neg y) \wedge (\neg y \vee \neg z) \wedge (\neg x \vee \neg z) \wedge (x \vee \neg w) \wedge (y \vee \neg w) \wedge (z \vee \neg w)$

(G,7) is in MAX-2-SAT: $x= 0$, $y = 0$, $z = 1$, $w = 0$ satisfies the 7 clauses
z, $(\neg x \vee \neg y)$, $(\neg y \vee \neg z)$, $(\neg x \vee \neg z)$, $(x \vee \neg w)$, $(y \vee \neg w)$, $(z \vee \neg w)$

Theorem:

- If $(x \vee y \vee z) = 1$, then there is assignment for w such that 7 clauses of G are satisfied.
- If $(x \vee y \vee z) = 0$, then no matter how w is chosen, at most 6 clauses of G are satisfied.

So, (G,8) is not in MAX-2-SAT.

MAX-2-SAT

If $x \vee y \vee z = 1$, then w can be chosen such that exactly seven of the other clauses are 1.

x	y	z	w	¬x ∨ ¬y	¬z ∨ ¬y	¬x ∨ ¬z	x ∨ ¬w	y ∨ ¬w	z ∨ ¬w	x∨y∨z
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	1	1	1	0	0	0	0
0	0	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	0	1	1
0	1	0	0	1	1	1	1	1	1	1
0	1	0	1	1	1	1	0	1	0	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	0	1	0	1	1	1
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	0	1
1	0	1	0	1	1	0	1	1	1	1
1	0	1	1	1	1	0	1	0	1	1
1	1	0	0	0	1	1	1	1	1	1
1	1	0	1	0	1	1	1	1	0	1
1	1	1	0	0	0	0	1	1	1	1
1	1	1	1	0	0	0	1	1	1	1

MAX-2-SAT

If $x \vee y \vee z = 1$, then w can be chosen such that exactly seven of the other clauses are 1.

x	y	z	w	¬x ∨ ¬y	¬z ∨ ¬y	¬x ∨ ¬z	x ∨ ¬w	y ∨ ¬w	z ∨ ¬w	x∨y∨z
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	1	1	1	0	0	0	0
0	0	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	0	1	1
0	1	0	0	1	1	1	1	1	1	1
0	1	0	1	1	1	1	0	1	0	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	0	1	0	1	1	1
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	0	1
1	0	1	0	1	1	0	1	1	1	1
1	0	1	1	1	1	0	1	0	1	1
1	1	0	0	0	1	1	1	1	1	1
1	1	0	1	0	1	1	1	1	0	1
1	1	1	0	0	0	0	1	1	1	1
1	1	1	1	0	0	0	1	1	1	1

MAX-2-SAT

If $x \vee y \vee z = 1$, then w can be chosen such that exactly seven of the other clauses are 1.

x	y	z	w	¬x ∨ ¬y	¬z ∨ ¬y	¬x ∨ ¬z	x ∨ ¬w	y ∨ ¬w	z ∨ ¬w	x∨y∨z
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	1	1	1	0	0	0	0
0	0	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	0	1	1
0	1	0	0	1	1	1	1	1	1	1
0	1	0	1	1	1	1	0	1	0	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	0	1	0	1	1	1
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	0	1
1	0	1	0	1	1	0	1	1	1	1
1	0	1	1	1	1	0	1	0	1	1
1	1	0	0	0	1	1	1	1	1	1
1	1	0	1	0	1	1	1	1	0	1
1	1	1	0	0	0	0	1	1	1	1
1	1	1	1	0	0	0	1	1	1	1

Choose w = 0

MAX-2-SAT

If $x \vee y \vee z = 1$, then w can be chosen such that exactly seven of the other clauses are 1.

x	y	z	w	¬x ∨ ¬y	¬z ∨ ¬y	¬x ∨ ¬z	x ∨ ¬w	y ∨ ¬w	z ∨ ¬w	x∨y∨z
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	1	1	1	0	0	0	0
0	0	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	0	1	1
0	1	0	0	1	1	1	1	1	1	1
0	1	0	1	1	1	1	0	1	0	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	0	1	0	1	1	1
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	0	1
1	0	1	0	1	1	0	1	1	1	1
1	0	1	1	1	1	0	1	0	1	1
1	1	0	0	0	1	1	1	1	1	1
1	1	0	1	0	1	1	1	1	0	1
1	1	1	0	0	0	0	1	1	1	1
1	1	1	1	0	0	0	1	1	1	1

MAX-2-SAT

If $x \vee y \vee z = 0$, then no matter how we choose w, at most six of the clauses are 1.

x	y	z	w	¬x ∨ ¬y	¬z ∨ ¬y	¬x ∨ ¬z	x ∨ ¬w	y ∨ ¬w	z ∨ ¬w	x∨y∨z
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	1	1	1	0	0	0	0
0	0	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	0	1	1
0	1	0	0	1	1	1	1	1	1	1
0	1	0	1	1	1	1	0	1	0	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	0	1	0	1	1	1
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	0	1
1	0	1	0	1	1	0	1	1	1	1
1	0	1	1	1	1	0	1	0	1	1
1	1	0	0	0	1	1	1	1	1	1
1	1	0	1	0	1	1	1	1	0	1
1	1	1	0	0	0	0	1	1	1	1
1	1	1	1	0	0	0	1	1	1	1

MAX-2-SAT is NP-Complete

Will first show that MAX-2-SAT is in NP.

The witness is a satisfying assignment: for every clause, we check whether the clause is satisfied and keep track of the number of satisfied clauses. We check whether this number is $\geq m$.

MAX-2-SAT is NP-Complete

Will first show that MAX-2-SAT is in NP.

The witness is a satisfying assignment: for every clause, we check whether the clause is satisfied and keep track of the number of satisfied clauses. We check whether this number is $\geq$ m.

Next, we show that 3-SAT $\leq$ MAX-2-SAT.

$F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ 3-formula. Will transform it into (H,7m) where H is a 1-2-formula such that F is satisfiable iff (H,7m) is in MAX-2-SAT, i.e., there is assignment satisfying $\geq$ 7m clauses of H.

MAX-2-SAT is NP-Complete

Will first show that MAX-2-SAT is in NP.

The witness is a satisfying assignment: for every clause, we check whether the clause is satisfied and keep track of the number of satisfied clauses. We check whether this number is ≥ m.

Next, we show that 3-SAT ≤ MAX-2-SAT.

$F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ 3-formula. Will transform it into (H,7m) where H is a 1-2-formula such that F is satisfiable iff (H,7m) is in MAX-2-SAT, i.e., there is assignment satisfying ≥ 7m clauses of H.

Introduce new variables $w_1, \ldots, w_m$.

$C_j = \lambda_1 \vee \lambda_2 \vee \lambda_3$

$D_j = \lambda_1 \wedge \lambda_2 \wedge \lambda_3 \wedge w_j \wedge (\neg\lambda_1 \vee \neg\lambda_2) \wedge (\neg\lambda_2 \vee \neg\lambda_3) \wedge (\neg\lambda_1 \vee \neg\lambda_3) \wedge (\lambda_1 \vee \neg w_j) \wedge (\lambda_2 \vee \neg w_j) \wedge (\lambda_3 \vee \neg w_j)$

$H = D_1 \wedge D_2 \wedge \cdots \wedge D_m$ (10m clauses)

Transformation is poly-time.

MAX-2-SAT is NP-Complete

Will first show that MAX-2-SAT is in NP.

The witness is a satisfying assignment: for every clause, we check whether the clause is satisfied and keep track of the number of satisfied clauses. We check whether this number is $\geq$ m.

Next, we show that 3-SAT $\leq$ MAX-2-SAT.

$F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ 3-formula. Will transform it into (H,7m) where H is a 1-2-formula such that F is satisfiable iff (H,7m) is in MAX-2-SAT, i.e., there is assignment satisfying $\geq$ 7m clauses of H.

Introduce new variables $w_1, \ldots, w_m$.

$C_j = \lambda_1 \vee \lambda_2 \vee \lambda_3$ □

$D_j = \lambda_1 \wedge \lambda_2 \wedge \lambda_3 \wedge w_j \wedge (\neg\lambda_1 \vee \neg\lambda_2) \wedge (\neg\lambda_2 \vee \neg\lambda_3) \wedge (\neg\lambda_1 \vee \neg\lambda_3) \wedge (\lambda_1 \vee \neg w_j) \wedge (\lambda_2 \vee \neg w_j) \wedge (\lambda_3 \vee \neg w_j)$

$H = D_1 \wedge D_2 \wedge \cdots \wedge D_m$ (10m clauses)

Transformation is poly-time.

- F satisfiable:
  - By theorem, there is setting for the w_j such that each D_j has 7 satisfied clauses.
  - (H,7m) is in MAX-2-SAT

MAX-2-SAT is NP-Complete

Will first show that MAX-2-SAT is in NP.

The witness is a satisfying assignment: for every clause, we check whether the clause is satisfied and keep track of the number of satisfied clauses. We check whether this number is $\geq$ m.

Next, we show that 3-SAT $\leq$ MAX-2-SAT.

$F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ 3-formula. Will transform it into (H,7m) where H is a 1-2-formula such that F is satisfiable iff (H,7m) is in MAX-2-SAT, i.e., there is assignment satisfying $\geq$ 7m clauses of H.

Introduce new variables $w_1, \dots, w_m$.

$C_j = \lambda_1 \vee \lambda_2 \vee \lambda_3$ □

$D_j = \lambda_1 \wedge \lambda_2 \wedge \lambda_3 \wedge w_j \wedge (\neg\lambda_1 \vee \neg\lambda_2) \wedge (\neg\lambda_2 \vee \neg\lambda_3) \wedge (\neg\lambda_1 \vee \neg\lambda_3) \wedge (\lambda_1 \vee \neg w_j) \wedge (\lambda_2 \vee \neg w_j) \wedge (\lambda_3 \vee \neg w_j)$

$H = D_1 \wedge D_2 \wedge \cdots \wedge D_m$ (10m clauses)

Transformation is poly-time.

- F satisfiable:
  - By theorem, there is setting for the w_j such that each D_j has 7 satisfied clauses.
  - (H,7m) is in MAX-2-SAT

- F not satisfiable:
  - By theorem, no matter how w_j is chosen, D_j cannot have more than 6 satisfied clauses.
  - Hence H cannot have more than 6m satisfied clauses.
  - (H,7m) is not in MAX-2-SAT

Not-all-equal 3-SAT

An assignment $(b_1,\ldots,b_n)$ to boolean variables $x_1,\ldots,x_n$ *not-all-equal-satisfies* a clause $C = (\lambda_1 \vee \lambda_2 \vee \cdots \vee \lambda_k)$ if there is at least one satisfied and one unsatisfied literal under this assignment.

Example: (1,1,1) does NOT NAE-satisfy $(x_1 \vee x_2 \vee x_3)$.

NAE-3-SAT = { F | F is a NAE-satisfiable 3-formula }

NAE-3-SAT is NP-Complete

Will first show that NAE-3-SAT is in NP.

The witness is a satisfying assignment: for every clause, check that the assignment NAE-satisfies the clause. Can be done in polynomial time.

NAE-3-SAT is NP-Complete

Will first show that NAE-3-SAT is in NP.

The witness is a satisfying assignment: for every clause, check that the assignment NAE-satisfies the clause. Can be done in polynomial time.

Next, we show that 3-SAT ≤ NAE-3-SAT.

$F = C_1 \wedge C_2 \wedge \cdots \wedge C_k$ 3-formula. Will transform it into another 3-formula G such that G is NAE-satisfiable iff F is satisfiable.

NAE-3-SAT is NP-Complete

Will first show that NAE-3-SAT is in NP.

The witness is a satisfying assignment: for every clause, check that the assignment NAE-satisfies the clause. Can be done in polynomial time.

Next, we show that 3-SAT $\leq$ NAE-3-SAT.

$F = C_1 \wedge C_2 \wedge \cdots \wedge C_k$ 3-formula. Will transform it into another 3-formula G such that G is NAE-satisfiable iff F is satisfiable.

Introduce new variables z, $w_1, \ldots, w_k$.

$C_j = \lambda_1 \vee \lambda_2 \vee \lambda_3$ $\to$ $D_j = (\lambda_1 \vee \lambda_2 \vee w_j) \wedge (\neg w_j \vee \lambda_3 \vee z)$

$G = D_1 \wedge D_2 \wedge \cdots \wedge D_k$ (2k clauses)

NAE-3-SAT is NP-Complete

Will first show that NAE-3-SAT is in NP.

The witness is a satisfying assignment: for every clause, check that the assignment NAE-satisfies the clause. Can be done in polynomial time.

Next, we show that 3-SAT $\leq$ NAE-3-SAT.

$F = C_1 \wedge C_2 \wedge \cdots \wedge C_k$ 3-formula. Will transform it into another 3-formula G such that G is NAE-satisfiable iff F is satisfiable.

Introduce new variables $z, w_1, \ldots, w_k$.

$C_j = \lambda_1 \vee \lambda_2 \vee \lambda_3$ $\square$ $D_j = (\lambda_1 \vee \lambda_2 \vee w_j) \wedge (\neg w_j \vee \lambda_3 \vee z)$

$G = D_1 \wedge D_2 \wedge \cdots \wedge D_k$ (2k clauses)

Transformation is poly-time.

- F satisfiable:
  - if $\lambda_1 \vee \lambda_2 = 1$, set $w_j=0, z=0$
  - if $\lambda_1 \vee \lambda_2 = 0$, set $w_j=1, z=0$

$\Rightarrow$ G is NAE-satisfiable.

NAE-3-SAT is NP-Complete

Will first show that NAE-3-SAT is in NP.

The witness is a satisfying assignment: for every clause, check that the assignment NAE-satisfies the clause. Can be done in polynomial time.

Next, we show that 3-SAT $\leq$ NAE-3-SAT.

$F = C_1 \wedge C_2 \wedge \cdots \wedge C_k$ 3-formula. Will transform it into another 3-formula G such that G is NAE-satisfiable iff F is satisfiable.

Introduce new variables $z, w_1, \ldots, w_k$.

$C_j = \lambda_1 \vee \lambda_2 \vee \lambda_3$ ▯ $D_j = (\lambda_1 \vee \lambda_2 \vee w_j) \wedge (\neg w_j \vee \lambda_3 \vee z)$

$G = D_1 \wedge D_2 \wedge \cdots \wedge D_k$ (2k clauses)

Transformation is poly-time.

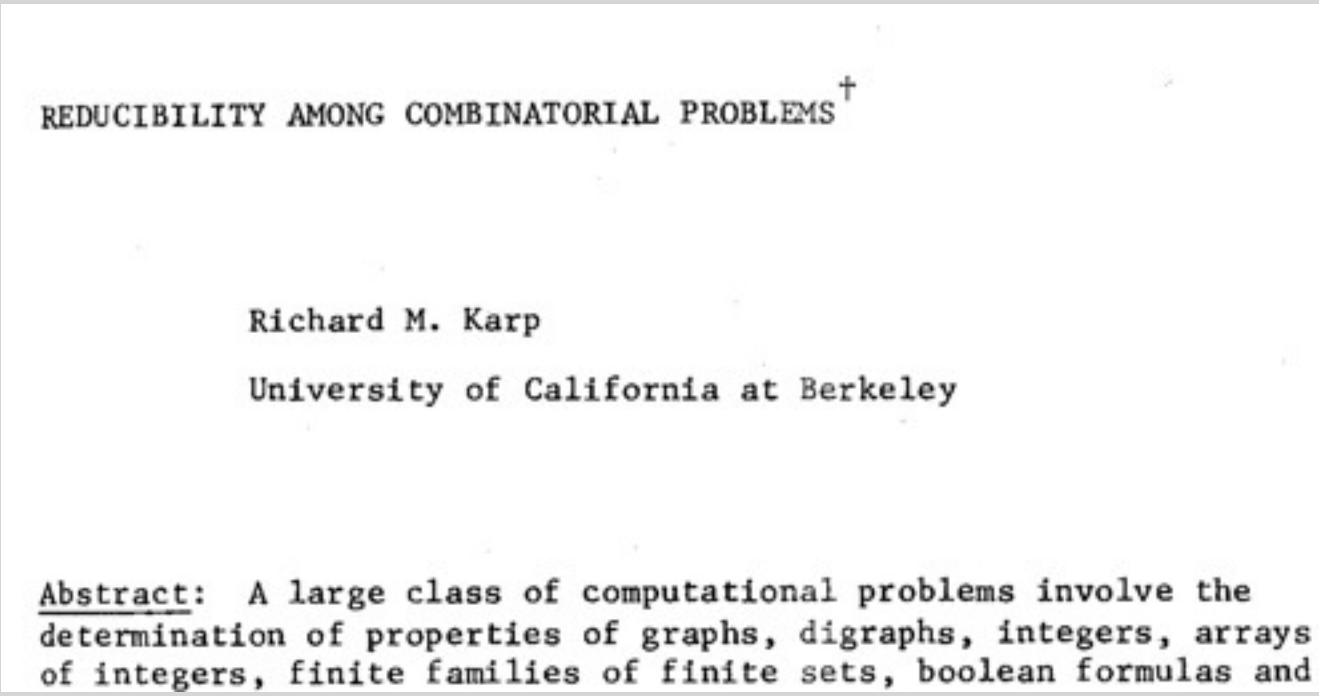
- F satisfiable:
  - if $\lambda_1 \vee \lambda_2 = 1$, set $w_j=0$, $z=0$
  - if $\lambda_1 \vee \lambda_2 = 0$, set $w_j=1$, $z=0$

  $\Rightarrow$ G is NAE-satisfiable.

- G NAE-satisfiable:
  - $(x_1,x_2,\ldots,x_n,w_1,\ldots,w_k,z=1)$ NAE-satisfies G iff $(\neg x_1,\ldots,\neg x_n,\neg w_1,\ldots,\neg w_k,z=0)$ does
  - Can assume $z = 0$
  - Then D_j is NAE-satisfiable iff $\lambda_1 \vee \lambda_2 = 1$ or $\lambda_3 = 1$, so iff C_j is satisfiable.

  $\Rightarrow$ F is satisfiable.

Lots more NP-Complete Problems



REDUCIBILITY AMONG COMBINATORIAL PROBLEMS†

Richard M. Karp

University of California at Berkeley

Abstract: A large class of computational problems involve the determination of properties of graphs, digraphs, integers, arrays of integers, finite families of finite sets, boolean formulas and

Richard Karp, 1935 –

In 1972 Richard Karp published a landmark paper entitled

Reducibility among Combinatorial Problems

This paper took the theory of NP-completeness a leap forward by compiling a list of 21 well-known computational problems which he showed to be NP-complete as well.

We will discuss some of these problems in this class.

INDEP-SET is NP-Complete

Reminder:

INDEP-SET = { (G,m) | G is a graph with an independent set of size $\geq m$}

We prove that INDEP-SET is NP-complete by showing:

- INDEP-SET is in NP .... This one we have already seen
- 3-SAT $\leq$ INDEP-SET .... This one we need to show

To prove 3-SAT ≤ INDEP-SET:

- Take any 3-formula F
- Construct (efficiently) from F a graph G and an integer m such that

F is in 3-SAT $\Leftrightarrow$ (G,m) is in INDEP-SET

INDEP-SET is NP-Complete

F 3-formula with t clauses: $F = C_1 \wedge C_2 \wedge \dots \wedge C_t$

Construct graph G such that (G,t) is in INDEP-SET iff F is satisfiable.

INDEP-SET is NP-Complete

F 3-formula with t clauses: $F = C_1 \wedge C_2 \wedge \ldots \wedge C_t$

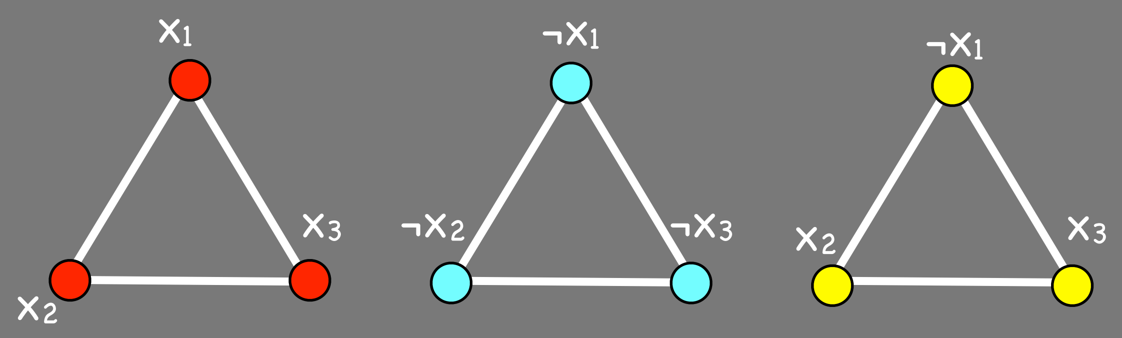
Construct graph G such that (G,t) is in INDEP-SET iff F is satisfiable.

Example:

$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

For every clause create a triangle with nodes labeled by the literals.

Connect all the nodes of the triangle.



INDEP-SET is NP-Complete

F 3-formula with t clauses: $F = C_1 \wedge C_2 \wedge \ldots \wedge C_t$

Construct graph G such that (G,t) is in INDEP-SET iff F is satisfiable.

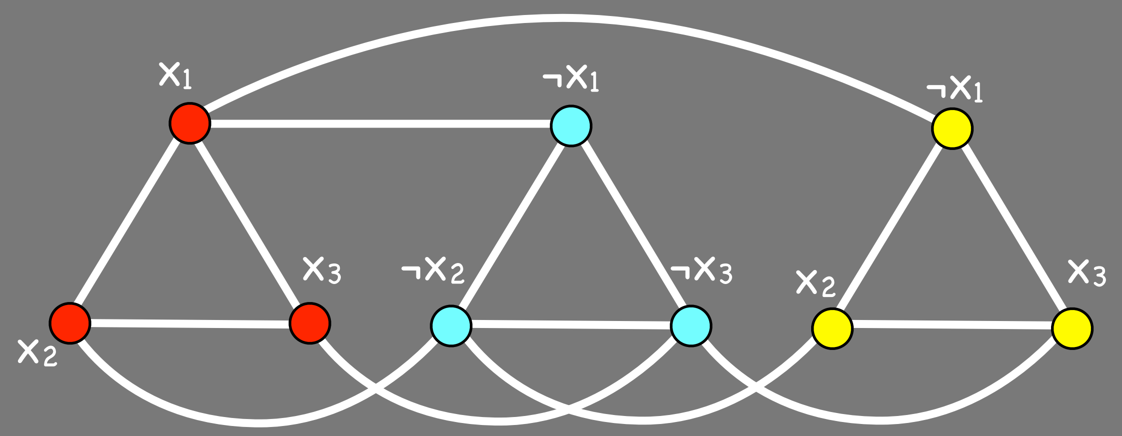
Example:

$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

For every clause create a triangle with nodes labeled by the literals.

Connect all the nodes of the triangle.

Connect literals by an edge who are negations of one another.



INDEP-SET is NP-Complete

Suppose that F is satisfiable.

In every clause, pick a literal λ that is satisfied. Mark corresponding node in graph G.

The set of marked nodes forms independent set of size t, so (G,t) is in INDEP-SET.

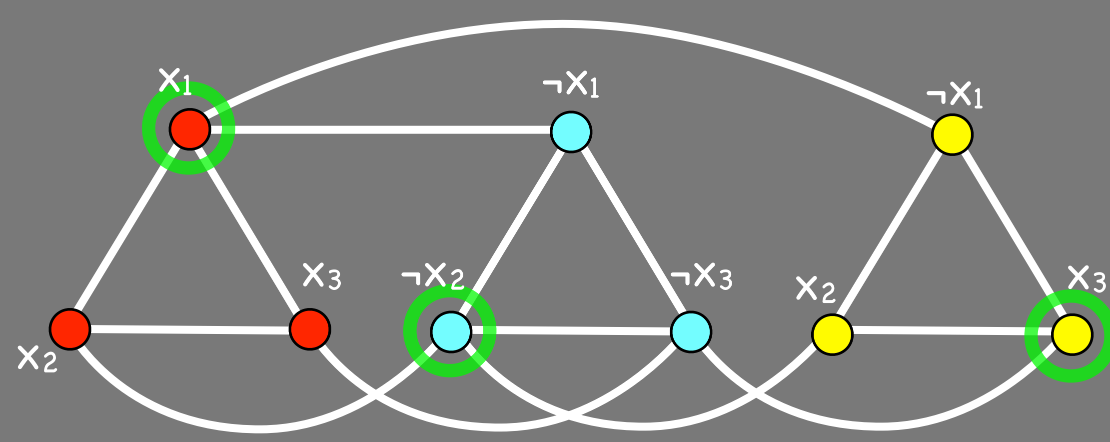
Example:

$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

Pick satisfying assignment (1,0,1):

Pick literal x_1 in the first clause, $\neg x_2$ in the second, and x_3 in the third.

Corresponding nodes form an independent set.



INDEP-SET is NP-Complete

Suppose that I is an independent set of G of size ≥ t.

|I| cannot be larger than t: each triangle can contribute at most one node to I and there are t triangles.

Take assignment that satisfies all literals corresponding to nodes in I (exists, since no "contradictions" allowed)

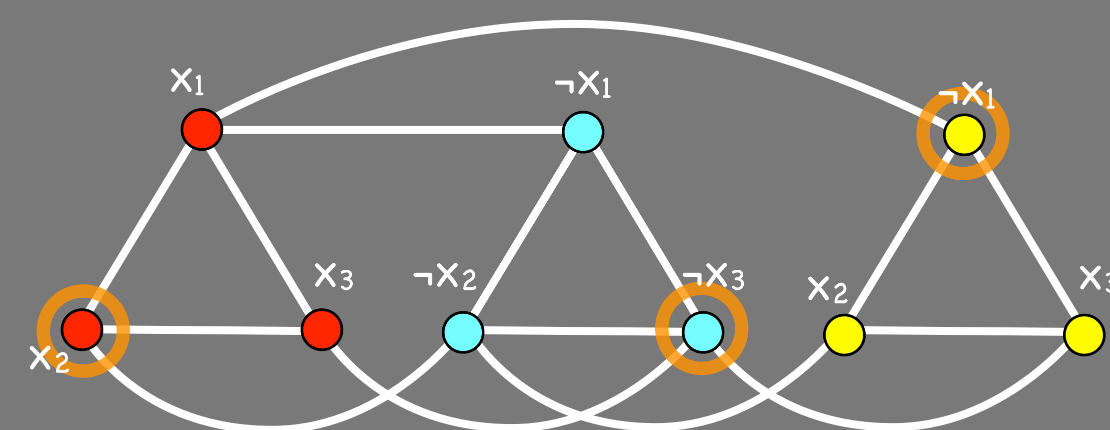
This is a satisfying assignment for F, since every clause has at least one satisfied literal.

Example:

$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

Pick independent set given on the right.

This corresponds to the satisfying assignment: $x_1=0$, $x_2=1$, $x_3=0$



INDEP-SET is NP-Complete

We already showed that:

- CLIQUE ≤ INDEP-SET ≤ CLIQUE
- NODE-COVER ≤ INDEP-SET ≤ NODE-COVER

So, we obtain

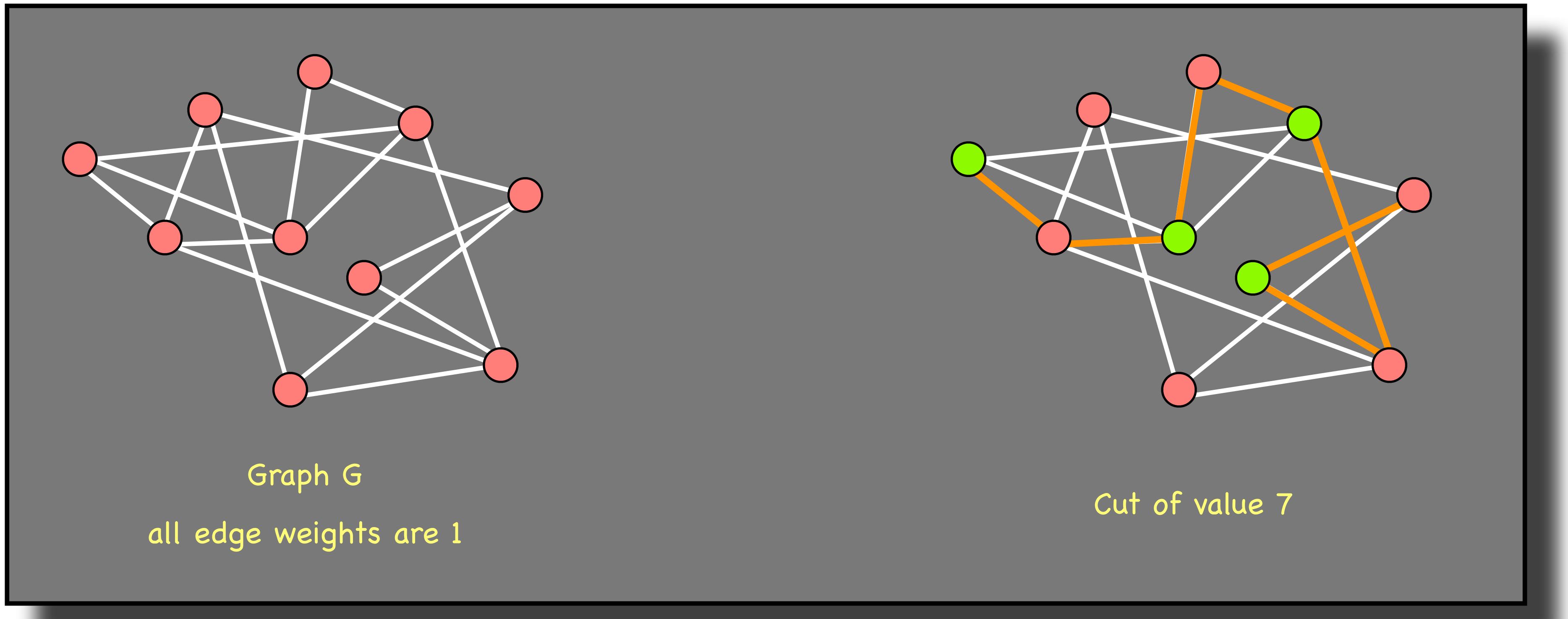
CLIQUE and NODE-COVER are NP-complete as well.

Cuts

G=(V,E) graph, w: E □ **N** positive integer weights on the edges.

A *cut* in G is a partition of the nodes into two subsets A and B.

The *value* of the cut is the sum of the edge values of all edges connecting some node in A to some node in B.



Min-Cut

MIN-CUT = { $(G,w,\ m)$ | G graph, w edge weights, G has a cut of size $\leq m$}

This problem is in P:

(1) set out = inf

(2) For all ordered pair of distinct nodes (s,t) of G do

  (a) Run the Ford-Fulkerson algorithm on (G, s,t, w); obtain value c for the min-cut

  (b) If c is smaller than out, then replace out by c

(3) Return out

Max-Cut

MAX-CUT = { (G,w, m) | G graph, w edge weights, G has a cut of size $\geq m$}

This problem cannot be solved the same way (simply multiplying edge weights with -1 and reducing to MIN-CUT doesn't work since to solve MIN-CUT the edge values have to be positive).

What can be said about this problem?

MAX-CUT is NP-complete!

Max-Cut is NP-Complete

We prove that MAX-CUT is NP-complete by showing:

- MAX-CUT is in NP .... The witness would be a description of the cut, so easy.
- NAE-3-SAT ≤ MAX-CUT .... This one we need to show

To prove NAE-3-SAT ≤ MAX-CUT:

- Take any 3-formula F

gadget

- Construct (efficiently) from F a graph G, weights w, and an integer m such that

F is NAE-satisfiable ⇔ (G,w) has a cut of size ≥ m

The Gadget

F 3-formula with t clauses: $F = C_1 \wedge C_2 \wedge \ldots \wedge C_t$

Construct graph G with weights w such that such that (G,w,5t) is in MAX-CUT iff F is satisfiable.

The Gadget

F 3-formula with t clauses: $F = C_1 \wedge C_2 \wedge \ldots \wedge C_t$

Construct graph G with weights w such that such that (G,w,5t) is in MAX-CUT iff F is satisfiable.

Example:

$$(x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge$$
$$(x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_4)$$

The Gadget

F 3-formula with t clauses: $F = C_1 \wedge C_2 \wedge \ldots \wedge C_t$

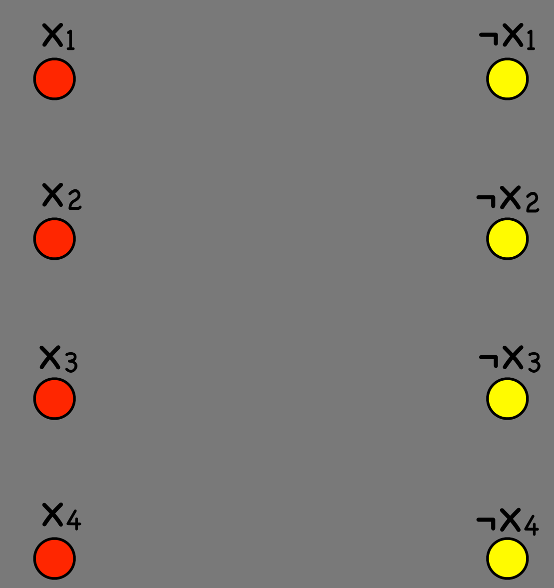
Construct graph G with weights w such that such that (G,w,5t) is in MAX-CUT iff F is satisfiable.

Example:

$(x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge$

$(x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_4)$

- Create one node for every variable and its negation



The Gadget

F 3-formula with t clauses: $F = C_1 \wedge C_2 \wedge \dots \wedge C_t$

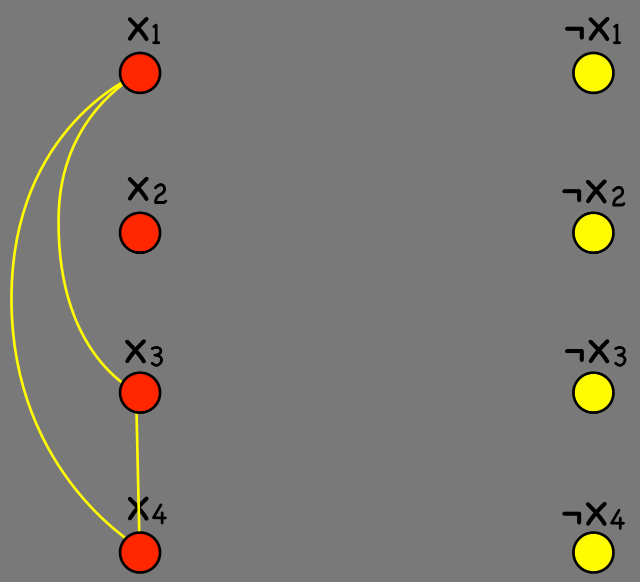
Construct graph G with weights w such that such that (G,w,5t) is in MAX-CUT iff F is satisfiable.

- Create one node for every variable and its negation
- For every clause $\lambda_1 \vee \lambda_2 \vee \lambda_3$ create triangle connecting the corresponding nodes (if some edge already exists, augment its weight by 1).

Example:

$(x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge$

$(x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_4)$



The Gadget

F 3-formula with t clauses: $F = C_1 \wedge C_2 \wedge \dots \wedge C_t$

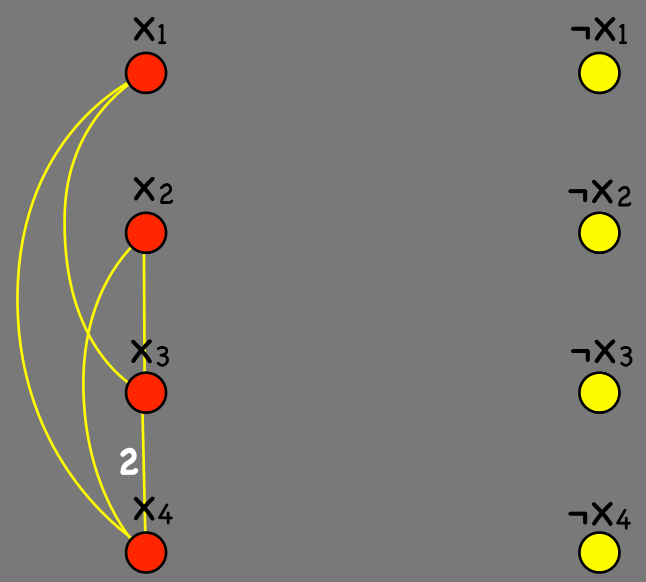
Construct graph G with weights w such that such that (G,w,5t) is in MAX-CUT iff F is satisfiable.

Example:

$(x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge$

$(x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_4)$

- Create one node for every variable and its negation
- For every clause $\lambda_1 \vee \lambda_2 \vee \lambda_3$ create triangle connecting the corresponding nodes (if some edge already exists, augment its weight by 1).



The Gadget

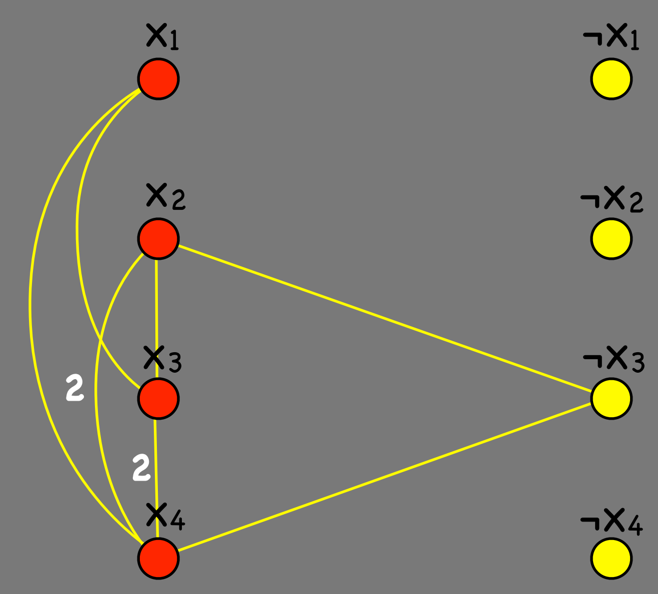
F 3-formula with t clauses: $F = C_1 \wedge C_2 \wedge \ldots \wedge C_t$

Construct graph G with weights w such that such that (G,w,5t) is in MAX-CUT iff F is satisfiable.

Example:

$(x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge$

$(x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_4)$



- Create one node for every variable and its negation
- For every clause $\lambda_1 \vee \lambda_2 \vee \lambda_3$ create triangle connecting the corresponding nodes (if some edge already exists, augment its weight by 1).

The Gadget

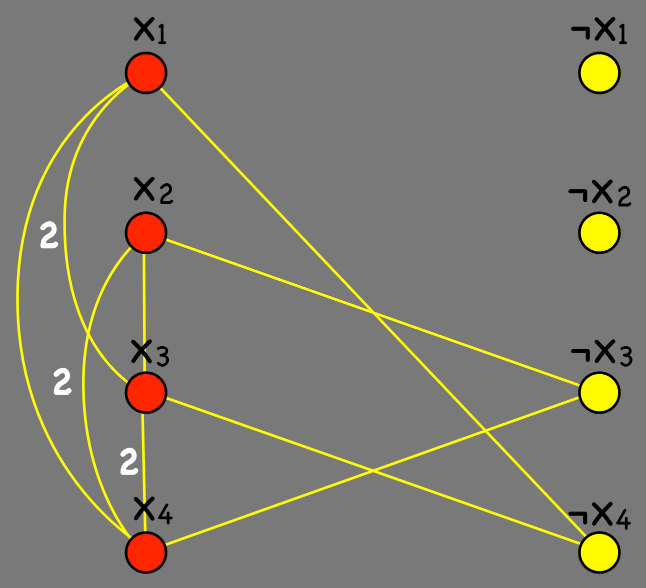
F 3-formula with t clauses: $F = C_1 \wedge C_2 \wedge \ldots \wedge C_t$

Construct graph G with weights w such that such that (G,w,5t) is in MAX-CUT iff F is satisfiable.

Example:

$(x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge$

$(x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_4)$



- Create one node for every variable and its negation
- For every clause $\lambda_1 \vee \lambda_2 \vee \lambda_3$ create triangle connecting the corresponding nodes (if some edge already exists, augment its weight by 1).

The Gadget

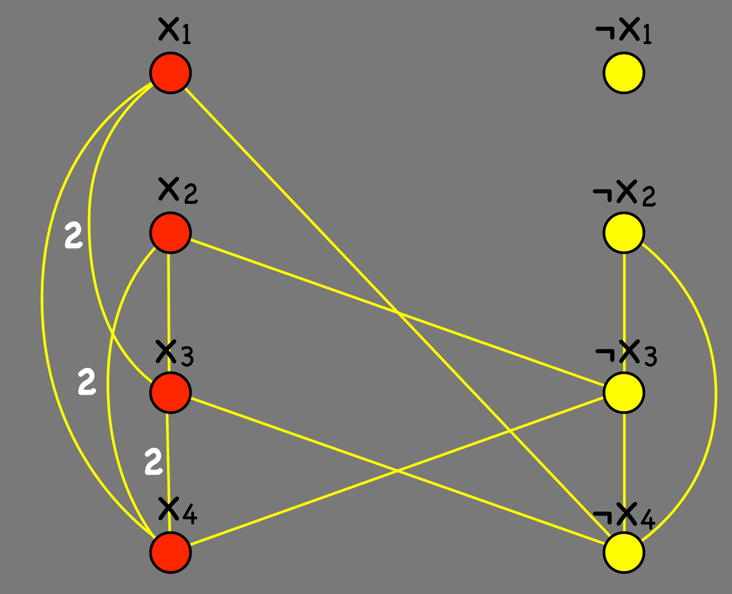
F 3-formula with t clauses: $F = C_1 \wedge C_2 \wedge \dots \wedge C_t$

Construct graph G with weights w such that such that (G,w,5t) is in MAX-CUT iff F is satisfiable.

Example:

$(x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge$

$(x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_4)$



- Create one node for every variable and its negation
- For every clause $\lambda_1 \vee \lambda_2 \vee \lambda_3$ create triangle connecting the corresponding nodes (if some edge already exists, augment its weight by 1).

The Gadget

F 3-formula with t clauses: $F = C_1 \wedge C_2 \wedge \ldots \wedge C_t$

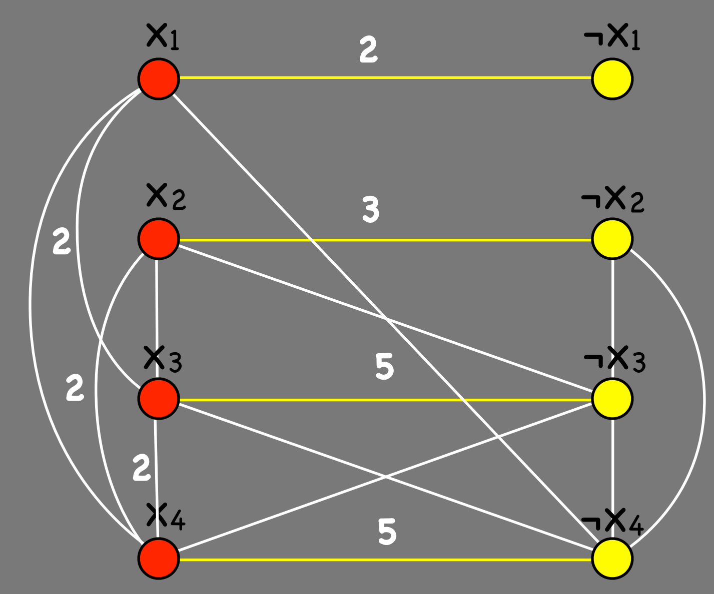
Construct graph G with weights w such that such that (G,w,5t) is in MAX-CUT iff F is satisfiable.

- Create one node for every variable and its negation
- For every clause $\lambda_1 \vee \lambda_2 \vee \lambda_3$ create triangle connecting the corresponding nodes (if some edge already exists, augment its weight by 1).
- Connect the nodes x_i and $\neg x_i$ by an edge of weight n_i, where n_i is the number of times x_i or its negation appears in the formula. (NOTE: the sum of all the n_i is the total number of literals in the formula, so 3t)

Example:

$(x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge$

$(x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_4)$



$n_1=2,\ n_2=3,\ n_3=n_4=5$

The Gadget

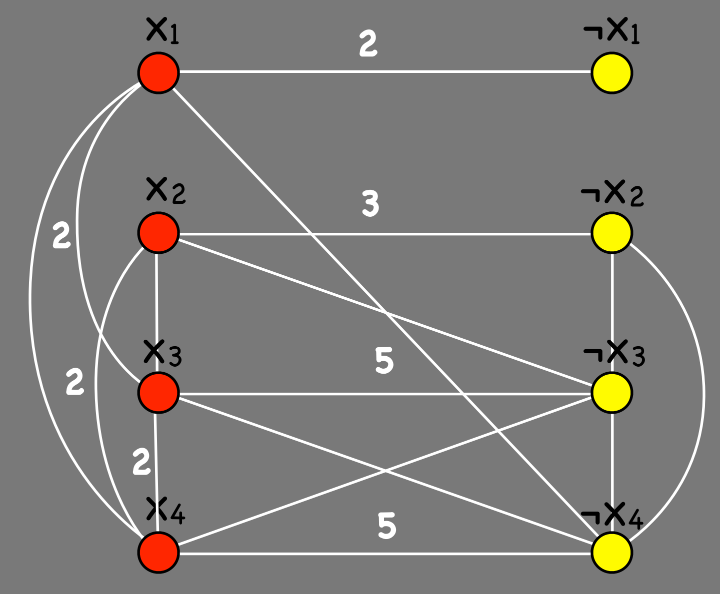
F 3-formula with t clauses: $F = C_1 \wedge C_2 \wedge \ldots \wedge C_t$

Construct graph G with weights w such that such that (G,w,5t) is in MAX-CUT iff F is satisfiable.

Example:

$(x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge$

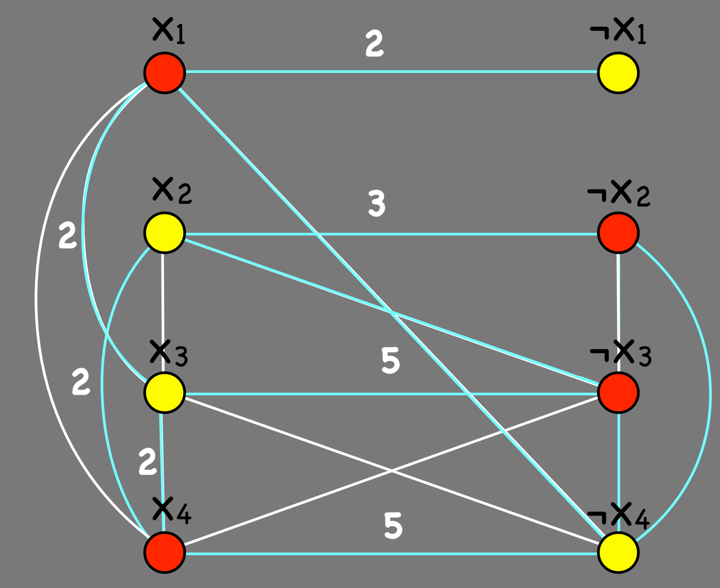
$(x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_4)$



- Create one node for every variable and its negation
- For every clause $\lambda_1 \vee \lambda_2 \vee \lambda_3$ create triangle connecting the corresponding nodes (if some edge already exists, augment its weight by 1).
- Connect the nodes x_i and $\neg x_i$ by an edge of weight n_i, where n_i is the number of times x_i or its negation appears in the formula. (NOTE: the sum of all the n_i is the total number of literals in the formula, i.e., 3t)
- Transformation is polynomial time.

The Gadget

Suppose that G has a cut (A,B) of size ≥ 5t:



Cut of value 25 = 5t

The Gadget

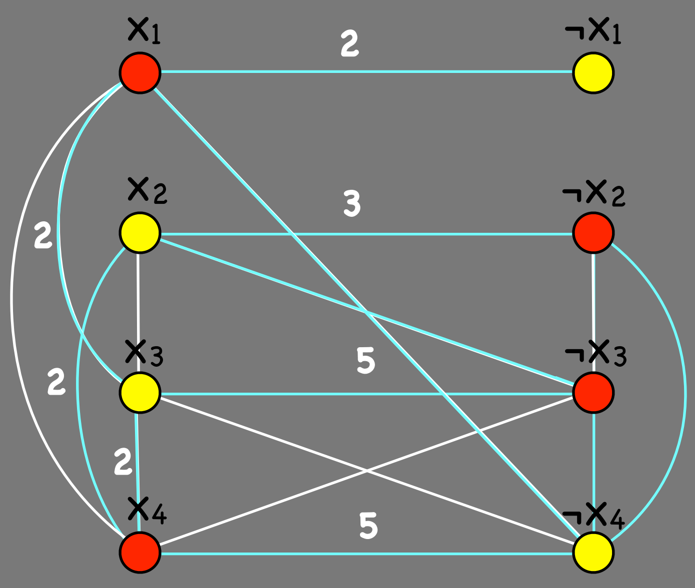
Suppose that G has a cut (A,B) of size ≥ 5t:

(1) We can assume that the variables and their negations belong to different sides

(a) If literals x_i and $\neg x_i$ belong to the same side of the cut, they contribute together at most n_i to the cut (each appearance of one of these literals contributes at most 1 to the cut).

(b) Hence, putting them at different sides at most increases the value of the cut.

(c) So, if a cut of value ≥ 5t exists, then there is a cut of value ≥ 5t in which all variables and their negations belong to different sides of the cut.



Cut of value 25 = 5t

The Gadget

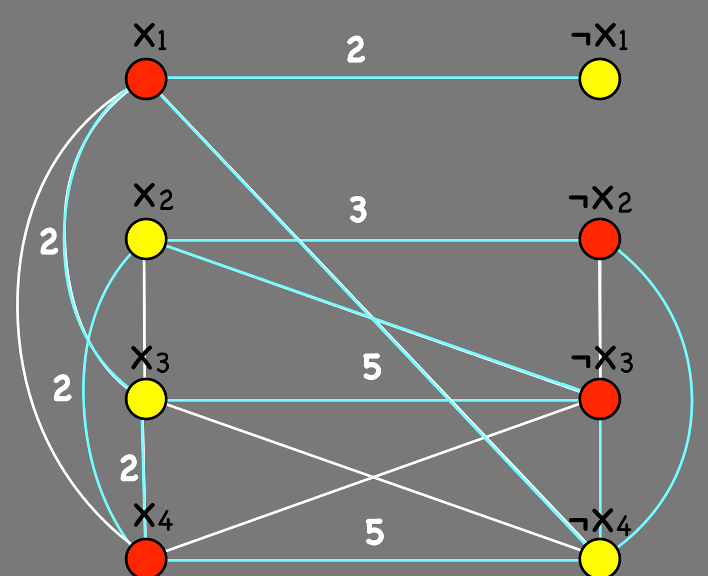
Suppose that G has a cut (A,B) of size $\geq 5t$:

(1) We can assume that the variables and their negations belong to different sides

(2) The variables and their negations contribute $\sum_i n_i = 3t$ to the cut. Remaining $\geq 2t$ has to come from the triangles.

   (a) A triangle contributes either 0 or 2 to the cut: 0 if all nodes are on the same side of the cut, 2 else.

   (b) There are t triangles, so each has to contribute exactly 2 to the cut: in each triangle one of the nodes is on one side, the other two on the other side of the cut.



Cut of value 25 = 5t

The Gadget

Suppose that G has a cut (A,B) of size ≥ 5t:

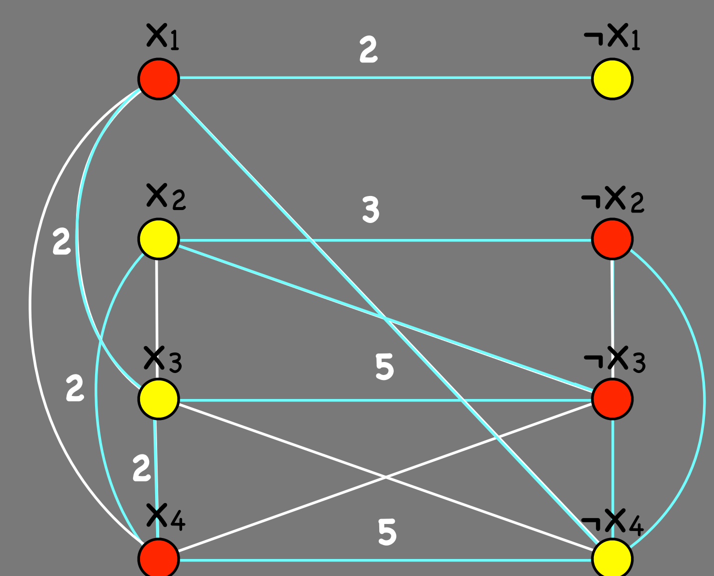
(1) We can assume that the variables and their negations belong to different sides

(2) The variables and their negations contribute $\Sigma_i \, n_i = 3t$ to the cut. Remaining ≥ 2t has to come from the triangles.

(3) Select the assignment which satisfies all the literals in A.

(a) There is no contradiction (literals and their negations are on different sides)

(b) Each clause has at least one and at most 2 satisfied literals (because each triangle contributes exactly 2 to the cut).



Assignment: x_1=1, x_2=x_3=0, x_4=1

The Gadget

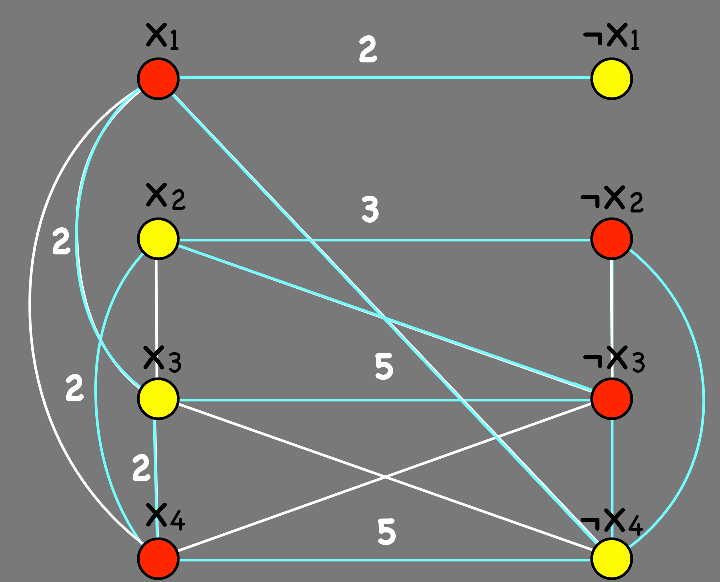
Suppose that G has a cut (A,B) of size ≥ 5t:

(1) We can assume that the variables and their negations belong to different sides

(2) The variables and their negations contribute $\Sigma_i\ n_i = 3t$ to the cut. Remaining ≥ 2t has to come from the triangles.

(3) Select the assignment which satisfies all the literals in A.

(4) F is NAE-satisfiable.



Assignment: $x_1=1$, $x_2=x_3=0$, $x_4=1$

The Gadget

Suppose that F is NAE-satisfiable. Choose an NAE-satisfying assignment.

(1) Choose as A the set of nodes corresponding to satisfied literals.

  (a) Literals x_i and $\neg x_i$ belong to different sides of the cut, so they contribute in total $\sum_i n_i = 3t$ to the cut.

  (b) Each triangle contributes 2 to the cut, so in total all triangles contribute $2t$ to the cut.

(2) The value of the corresponding cut is $5t$, so $(G,w,5t)$ is in MAX-CUT.

$(x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_4)$

has NAE-satisfying assignment $x_1=x_2=1$, $x_3=x_4=0$.

This gives cut of value $25 = 5t$.