Algorithms and Complexity (AC)

Marie Schmidt & Tom van der Zanden

(Based on slides by Gerhard Woeginger and Jesper Nederlof)

Landelijk Netwerk Mathematische Besliskunde

LNMB, Sep–Nov 2021

Note

Shorter lecture today.
Remainder of the lecture (NP-completeness proofs) is recorded as videos, see email (and coming on website soon)

Please type in the chat '[your name] is here' for the attendance list.

Definition: Complexity class NP

A decision problem X lies in the complexity class NP, if

- if it can be solved in polynomial time on a non-deterministic Turing machine (original, formal definition)
- (or, alternatively:) if it is solved by a *non-deterministic* algorithm with polynomial time complexity.
- (or, alternatively:) if **the YES-instances** of X possess certificates of polynomial length that can be verified in polynomial time.

Definition: Complexity class NP

A decision problem X lies in the complexity class NP, if

- if it can be solved in polynomial time on a non-deterministic Turing machine (original, formal definition)
- (or, alternatively:) if it is solved by a *non-deterministic* algorithm with polynomial time complexity.
- (or, alternatively:) if **the YES-instances** of X possess certificates of polynomial length that can be verified in polynomial time.

Remark: A decision problem X lies in the complexity class co-NP, if **the NO-instances** of X possess certificates of polynomial length that can be verified in polynomial time. More on this in lecture 3.

Example for problem in NP: Subset Sum

Decision problem Subset Sum (SS)

Instance: positive integers $a_1, \ldots, a_n$; a bound b

Question: does there exist an index set $I \subseteq \{1, \ldots, n\}$ with $\sum_{i \in I} a_i = b$?

Example: $(a_1, \ldots, a_{12}) = (1, \ldots, 12), b = 50$. Yes or no instance?

Example for problem in NP: Subset Sum

Decision problem Subset Sum (SS)

Instance: positive integers $a_1, \ldots, a_n$; a bound b

Question: does there exist an index set $I \subseteq \{1, \ldots, n\}$ with $\sum_{i \in I} a_i = b$?

Example: $(a_1, \ldots, a_{12}) = (1, \ldots, 12), b = 50$. Yes or no instance?
Yes: $1 + 2 + 3 + 4 + 6 + 7 + 8 + 9 + 10 = 50$.

Example for problem in NP: Subset Sum

Decision problem Subset Sum (SS)

Instance: positive integers $a_1, \ldots, a_n$; a bound b

Question: does there exist an index set $I \subseteq \{1, \ldots, n\}$ with $\sum_{i \in I} a_i = b$?

Example: $(a_1, \ldots, a_{12}) = (1, \ldots, 12), b = 50$. Yes or no instance?
Yes: $1 + 2 + 3 + 4 + 6 + 7 + 8 + 9 + 10 = 50$.

Question

What's a good NP-certificate for SS?

Example for problem in NP: Exact cover

Decision problem Exact cover (Ex-Cov)

Instance: a ground set X; subsets $S_1, \ldots, S_m$ of X

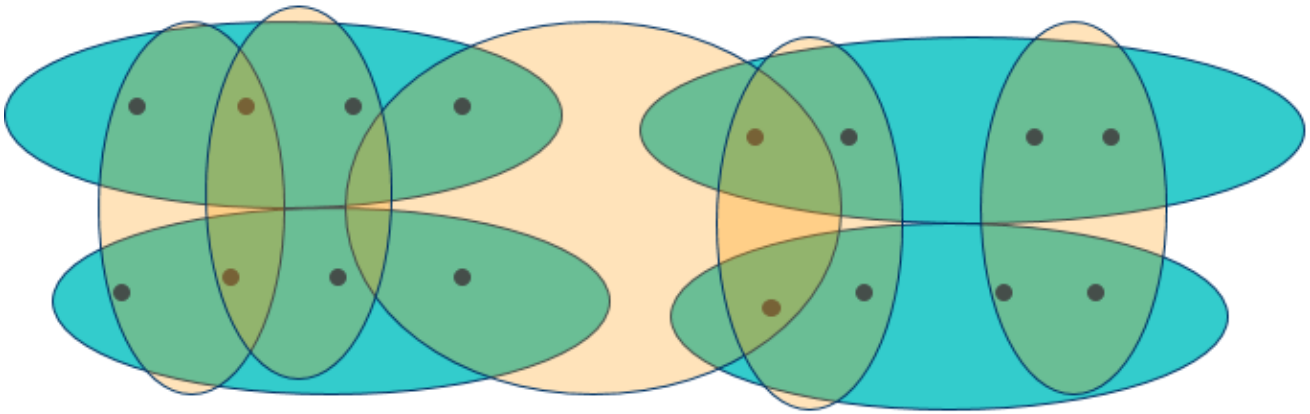
Question: do there exist some subsets S_i that form a partition of X?

Example for problem in NP: Exact cover

Decision problem Exact cover (Ex-Cov)

Instance: a ground set X; subsets $S_1, \ldots, S_m$ of X

Question: do there exist some subsets S_i that form a partition of X?



Question

What's a good NP-certificate for Ex-Cov?

Example for problem in NP: Integer programming

Integer linear programming (ILP)- Decision version

Instance: an integer matrix A; an integer vector b

Question: does there exist an integer vector x with $Ax \leq b$?

Question

What's a good NP-certificate for ILP?

Questions?

Integer linear programming (ILP)- Decision version

Instance: an integer matrix A; an integer vector b

Question: does there exist an integer vector x with $Ax \leq b$?

Question

What's a good NP-certificate for ILP?

P versus NP

- P = class of all problems that are easy to solve
  P stands for Polynomial Time
- NP = huge class of problems that fulfill some soft condition
  NP contains lots of interesting and important decision problems
  NP stands for Non-deterministic Polynomial Time

P versus NP

- P = class of all problems that are easy to solve
  P stands for Polynomial Time
- NP = huge class of problems that fulfill some soft condition
  NP contains lots of interesting and important decision problems
  NP stands for Non-deterministic Polynomial Time

Big open question

P=NP ????

P versus NP

- P = class of all problems that are easy to solve
  P stands for Polynomial Time
- NP = huge class of problems that fulfill some soft condition
  NP contains lots of interesting and important decision problems
  NP stands for Non-deterministic Polynomial Time

Big open question

P=NP ????

Answer YES:

- would trigger a revolution in computing

P versus NP

- P = class of all problems that are easy to solve
  P stands for Polynomial Time
- NP = huge class of problems that fulfill some soft condition
  NP contains lots of interesting and important decision problems
  NP stands for Non-deterministic Polynomial Time

Big open question

P=NP ????

Answer YES:
- would trigger a revolution in computing

Answer NO:
- that's what most people expect

To prove that a problem

- can be solved in $O(n \log n)$, $O(n^3)$, etc
- is in P
- is in NP

is straightforward (although not always easy):

To prove that a problem

- can be solved in $O(n \log n)$, $O(n^3)$, etc
- is in P
- is in NP

is straightforward (although not always easy):

$\rightarrow$ Find an algorithm that runs in time $O(n \log n)$ / $O(n^3)$ / ... / polynomial time / non-deterministic polynomial time.

To prove that a problem

- can be solved in $O(n \log n)$, $O(n^3)$, etc
- is in P
- is in NP

is straightforward (although not always easy):

$\rightarrow$ Find an algorithm that runs in time $O(n \log n)$ / $O(n^3)$ / ... / polynomial time / non-deterministic polynomial time.

How do we prove that a problem can**not** be solved in a certain time?

"I can't find an efficient algorithm, because no such algorithm is possible!"

Proving lower bounds (short note)

Example: 'Find maximum element from unsorted list'

Input: A list of numbers $m_1, m_2, \ldots m_n$, a number M.
Question: Is there an element $\geq M$ in the list.

Proving lower bounds (short note)

Example: 'Find maximum element from unsorted list'

Input: A list of numbers $m_1, m_2, \ldots m_n$, a number M.
Question: Is there an element $\geq M$ in the list.

How fast can you solve this problem?

Proving lower bounds (short note)

Example: 'Find maximum element from unsorted list'

Input: A list of numbers $m_1, m_2, \ldots m_n$, a number M.
Question: Is there an element $\geq M$ in the list.

How fast can you solve this problem?

- can be done in time $O(n)$
- requires time $\Omega(n)$
- thus: $\Theta(n)$

Proving lower bounds (short note)

Example: 'Find maximum element from unsorted list'

Input: A list of numbers $m_1, m_2, \ldots m_n$, a number M.
Question: Is there an element $\geq M$ in the list.

How fast can you solve this problem?

- can be done in time $O(n)$
- requires time $\Omega(n)$
- thus: $\Theta(n)$

Note: Most problems need time $\Omega(n)$ to be solved.
Can you think of one that does not?

Proving lower bounds (short note)

Example: Decision problem 'Find maximum element from unsorted list'

Input: A list of numbers $m_1, m_2, \ldots m_n$, a number M.
Question: Is there an element $\geq M$ in the list.

How fast can you solve this problem?

- can be done in time $O(n)$
- requires time $\Omega(n)$
- thus: $\Theta(n)$

Note: Most problems need time $\Omega(n)$ to be solved.
Can you think of one that does not?

3-Satisfiability (3-SAT)

Instance:

a set of logical variables $X := \{x_1, \ldots, x_n\}$ and a set of clauses C of three literals over X

Question: does there exist a truth assignment for X that simultaneously satisfies all clauses in C?

3-Satisfiability (3-SAT)

Instance:
a set of logical variables $X := \{x_1, \ldots, x_n\}$ and a set of clauses C of three literals over X

Question: does there exist a truth assignment for X that simultaneously satisfies all clauses in C?

From stackexchange
(https://cstheory.stackexchange.com/questions/1060/best-upper-bounds-on-sat?rq=1 and
https://cstheory.stackexchange.com/questions/93/what-are-the-best-current-lower-bounds-on-3sat)
(retrieved 5.9.21)

- Best found non-randomized algorithm (for 3-SAT) seems to be $O(1.3303^n)$
- Best found randomized algorithm similar ($O(1.306995^n)$?)
- No one so far has been able to prove that SAT is in $\Omega(n^C)$ for a $c > 1$! Or in different words: no one has been able to prove that SAT cannot be solved in linear time!

3-Satisfiability (3-SAT)

Instance:
a set of logical variables $X := \{x_1, \ldots, x_n\}$ and a set of clauses C of three literals over X

Question: does there exist a truth assignment for X that simultaneously satisfies all clauses in C?

From stackexchange
(https://cstheory.stackexchange.com/questions/1060/best-upper-bounds-on-sat?rq=1 and
https://cstheory.stackexchange.com/questions/93/what-are-the-best-current-lower-bounds-on-3sat)
(retrieved 5.9.21)

- Best found non-randomized algorithm (for 3-SAT) seems to be $O(1.3303^n)$
- Best found randomized algorithm similar ($O(1.306995^n)$?)
- No one so far has been able to prove that SAT is in $\Omega(n^C)$ for a $c > 1$! Or in different words: no one has been able to prove that SAT cannot be solved in linear time!

Lower bounds are difficult to prove!

Lower bounds on problem complexity tend to be rare / weak / difficult to prove.

$\rightarrow$ We look at a different approach.

"I can't find an efficient algorithm, but neither can all these famous people."

Reductions

For a decision problem X, we denote by YES(X) the instances of that problem for which the answer is 'yes'.

Definition

For two decision problems X and Y, we say that X (polynomially) **reduces** to Y (and we write $X \leq_p Y$)
if there exists a polynomial time transformation f that translates instance of X into instances of Y with $I \in$YES(X) $\iff$ $f(I) \in$YES(Y).

Often, we omit the word 'polynomially' and just say that 'X reduces to Y'.

Reductions

For a decision problem X, we denote by YES(X) the instances of that problem for which the answer is 'yes'.

Definition

For two decision problems X and Y, we say that X (polynomially) **reduces** to Y (and we write $X \leq_p Y$)
if there exists a polynomial time transformation f that translates instance of X into instances of Y with $I \in$YES(X) $\iff$ $f(I) \in$YES(Y).

Often, we omit the word 'polynomially' and just say that 'X reduces to Y'.

Intuition:

- X can be modelled as a special case of Y
- the 'computational hardness' of X is upper bounded by Y's
- If Y is easy, then also X is easy
- If X is difficult, then also Y is difficult

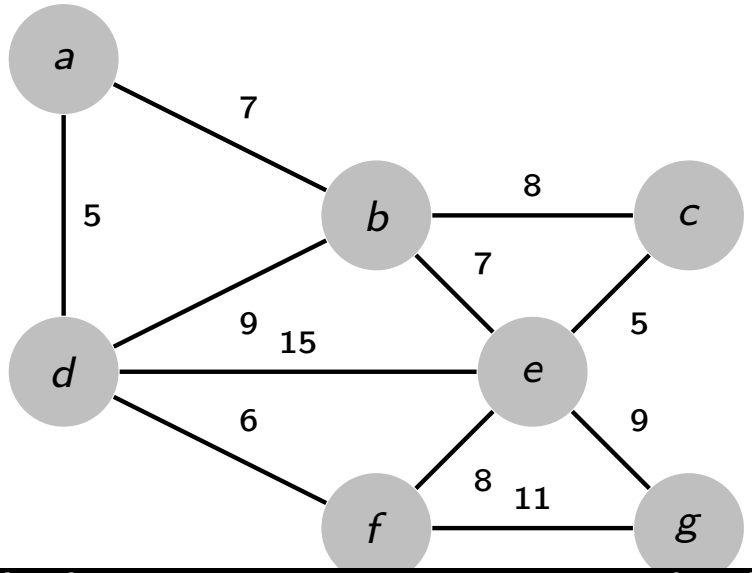
Hamiltonian cycle / TSP

Hamiltonian cycle (HC)

Instance: an undirected graph $G = (V, E)$
Question: does G contain a Hamiltonian cycle?
(a simple cycle that visits every vertex exactly once)

Travelling Salesman Problem (TSP)

Instance: cities $1, \ldots, n$; distances $d(i, j)$; a bound B
Question: does there exist a roundtrip of length at most B?



Theorem

HC $\leq_p$ TSP.

Proof: next slide .

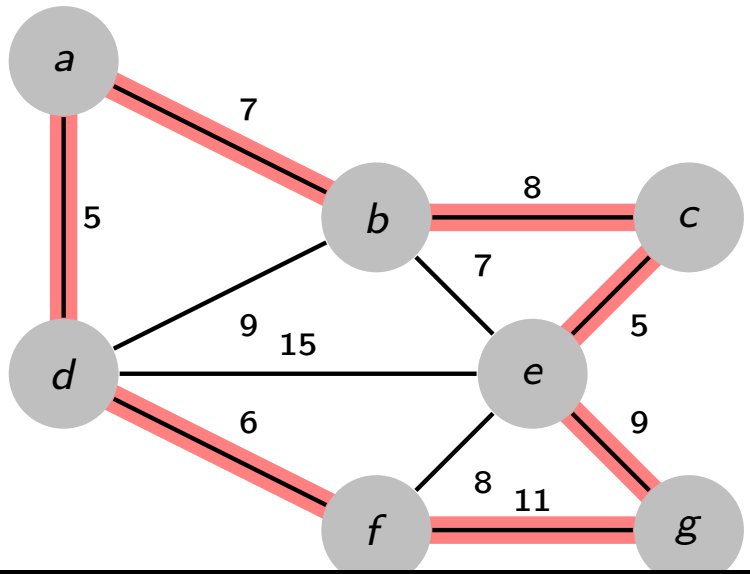
Hamiltonian cycle / TSP

Hamiltonian cycle (HC)

Instance: an undirected graph $G = (V, E)$
Question: does G contain a Hamiltonian cycle?
(a simple cycle that visits every vertex exactly once)

Travelling Salesman Problem (TSP)

Instance: cities $1, \ldots, n$; distances $d(i, j)$; a bound B
Question: does there exist a roundtrip of length at most B?



Theorem

HC $\leq_p$ TSP.

Proof: next slide .

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.
To define the polynomial transformation f, we specify how we construct from I:

- cities $1, \ldots, n$
- distances $d(i, j)$
- bound B

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.
To define the polynomial transformation f, we specify how we construct from I:

- cities $1, \ldots, n$
- distances $d(i, j)$
- bound B

Then we need to show that

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.
To define the polynomial transformation f, we specify how we construct from I:

- cities $1, \ldots, n$
- distances $d(i, j)$
- bound B

Then we need to show that

- This transformation f can be done in polynomial time
- If I is a yes-instance of HC $\Rightarrow$ $f(I)$ is a yes-instance of TSP
- If $f(I)$ is a yes-instance of TSP $\Rightarrow$ I is a yes-instance of TSP

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.

We define the polynomial transformation f of instance I of HC to instance $f(I)$ of TSP:

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.

We define the polynomial transformation f of instance I of HC to instance $f(I)$ of TSP:

- cities: introduce one city c_i for each $v_i \in V$

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.

We define the polynomial transformation f of instance I of HC to instance $f(I)$ of TSP:

- cities: introduce one city c_i for each $v_i \in V$
- distances: set $d(v, w) := \begin{cases} 1 & \text{if } e = \{v, w\} \in E \\ n+1 & \text{otherwise} \end{cases}$

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.

We define the polynomial transformation f of instance I of HC to instance $f(I)$ of TSP:

- cities: introduce one city c_i for each $v_i \in V$
- distances: set $d(v, w) := \begin{cases} 1 & \text{if } e = \{v, w\} \in E \\ n+1 & \text{otherwise} \end{cases}$
- bound: $B := n$

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.

We define the polynomial transformation f of instance I of HC to instance $f(I)$ of TSP:

- cities: introduce one city c_i for each $v_i \in V$
- distances: set $d(v, w) := \begin{cases} 1 & \text{if } e = \{v, w\} \in E \\ n+1 & \text{otherwise} \end{cases}$
- bound: $B := n$

This transformation can be done in polynomial time. ✓

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.

We define the polynomial transformation f of instance I of HC to instance $f(I)$ of TSP:

- cities: introduce one city c_i for each $v_i \in V$
- distances: set $d(v, w) := \begin{cases} 1 & \text{if } e = \{v, w\} \in E \\ n+1 & \text{otherwise} \end{cases}$
- bound: $B := n$

This transformation can be done in polynomial time. ✓

If I is a yes-instance of HC

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.

We define the polynomial transformation f of instance I of HC to instance $f(I)$ of TSP:

- cities: introduce one city c_i for each $v_i \in V$
- distances: set $d(v, w) := \begin{cases} 1 & \text{if } e = \{v, w\} \in E \\ n+1 & \text{otherwise} \end{cases}$
- bound: $B := n$

This transformation can be done in polynomial time. ✓

If I is a yes-instance of HC
$\Rightarrow$ there is a simple cycle $(v_{(1)}, v_{(2)}, v_{(3)}, \ldots, v_{(n)}, v_{(1)})$ in G that visits every vertex exactly once

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.

We define the polynomial transformation f of instance I of HC to instance $f(I)$ of TSP:

- cities: introduce one city c_i for each $v_i \in V$
- distances: set $d(v, w) := \begin{cases} 1 & \text{if } e = \{v, w\} \in E \\ n+1 & \text{otherwise} \end{cases}$
- bound: $B := n$

This transformation can be done in polynomial time. ✓

If I is a yes-instance of HC
$\Rightarrow$ there is a simple cycle $(v_{(1)}, v_{(2)}, v_{(3)}, \ldots, v_{(n)}, v_{(1)})$ in G that visits every vertex exactly once
$\Rightarrow$ $C := (c_{(1)}, c_{(2)}, c_{(3)}, \ldots, c_{(n)}, c_{(1)})$ is a roundtrip of cities that visits every city

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.

We define the polynomial transformation f of instance I of HC to instance $f(I)$ of TSP:

- cities: introduce one city c_i for each $v_i \in V$
- distances: set $d(v, w) := \begin{cases} 1 & \text{if } e = \{v, w\} \in E \\ n+1 & \text{otherwise} \end{cases}$
- bound: $B := n$

This transformation can be done in polynomial time. ✓

If I is a yes-instance of HC
$\Rightarrow$ there is a simple cycle $(v_{(1)}, v_{(2)}, v_{(3)}, \ldots, v_{(n)}, v_{(1)})$ in G that visits every vertex exactly once
$\Rightarrow$ $C := (c_{(1)}, c_{(2)}, c_{(3)}, \ldots, c_{(n)}, c_{(1)})$ is a roundtrip of cities that visits every city
$\Rightarrow$ The length of C is n, because C contains n edges of length 1

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.

We define the polynomial transformation f of instance I of HC to instance $f(I)$ of TSP:

- cities: introduce one city c_i for each $v_i \in V$
- distances: set $d(v, w) := \begin{cases} 1 & \text{if } e = \{v, w\} \in E \\ n+1 & \text{otherwise} \end{cases}$
- bound: $B := n$

This transformation can be done in polynomial time. ✓

If I is a yes-instance of HC
$\Rightarrow$ there is a simple cycle $(v_{(1)}, v_{(2)}, v_{(3)}, \ldots, v_{(n)}, v_{(1)})$ in G that visits every vertex exactly once
$\Rightarrow$ $C := (c_{(1)}, c_{(2)}, c_{(3)}, \ldots, c_{(n)}, c_{(1)})$ is a roundtrip of cities that visits every city
$\Rightarrow$ The length of C is n, because C contains n edges of length 1
$\Rightarrow$ $f(I)$ is a yes-instance of TSP ✓

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.

We define the polynomial transformation f of instance I of HC to instance $f(I)$ of TSP:

- cities: introduce one city c_i for each $v_i \in V$
- distances: set $d(v, w) := \begin{cases} 1 & \text{if } e = \{v, w\} \in E \\ n+1 & \text{otherwise} \end{cases}$
- bound: $B := n$

This transformation can be done in polynomial time. ✓
If I is a yes-instance of HC $\Rightarrow$ $f(I)$ is a yes-instance of TSP ✓

If $f(I)$ is a yes-instance of TSP

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.

We define the polynomial transformation f of instance I of HC to instance $f(I)$ of TSP:

- cities: introduce one city c_i for each $v_i \in V$
- distances: set $d(v, w) := \begin{cases} 1 & \text{if } e = \{v, w\} \in E \\ n+1 & \text{otherwise} \end{cases}$
- bound: $B := n$

This transformation can be done in polynomial time. ✓
If I is a yes-instance of HC $\Rightarrow$ $f(I)$ is a yes-instance of TSP ✓

If $f(I)$ is a yes-instance of TSP
$\Rightarrow$ there is a roundtrip R in $f(I)$ that visits all cities and has at most length $\leq B = n$

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.

We define the polynomial transformation f of instance I of HC to instance $f(I)$ of TSP:

- cities: introduce one city c_i for each $v_i \in V$
- distances: set $d(v, w) := \begin{cases} 1 & \text{if } e = \{v, w\} \in E \\ n+1 & \text{otherwise} \end{cases}$
- bound: $B := n$

This transformation can be done in polynomial time. ✓
If I is a yes-instance of HC $\Rightarrow$ $f(I)$ is a yes-instance of TSP ✓

If $f(I)$ is a yes-instance of TSP
$\Rightarrow$ there is a roundtrip R in $f(I)$ that visits all cities and has at most length $\leq B = n$
$\Rightarrow$ this roundtrip contains every city at most once (otherwise it would be longer than n), so we can write $R = (c_{(1)}, c_{(2)}, c_{(3)}, \ldots, c_{(n)}, c_{(1)})$

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.

We define the polynomial transformation f of instance I of HC to instance $f(I)$ of TSP:

- cities: introduce one city c_i for each $v_i \in V$
- distances: set $d(v, w) := \begin{cases} 1 & \text{if } e = \{v, w\} \in E \\ n+1 & \text{otherwise} \end{cases}$
- bound: $B := n$

This transformation can be done in polynomial time. ✓
If I is a yes-instance of HC $\Rightarrow$ $f(I)$ is a yes-instance of TSP ✓

If $f(I)$ is a yes-instance of TSP
$\Rightarrow$ there is a roundtrip R in $f(I)$ that visits all cities and has at most length $\leq B = n$
$\Rightarrow$ this roundtrip contains every city at most once (otherwise it would be longer than n), so we can write $R = (c_{(1)}, c_{(2)}, c_{(3)}, \ldots, c_{(n)}, c_{(1)})$
Furthermore, $d(c_{(i)}, c_{(i+1)}) = 1$ for all $i = 1, \ldots, n-1$ and $d(c_{(n)}, c_{(1)}) = 1$, because otherwise R would be longer than n

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.

We define the polynomial transformation f of instance I of HC to instance $f(I)$ of TSP:

- cities: introduce one city c_i for each $v_i \in V$
- distances: set $d(v, w) := \begin{cases} 1 & \text{if } e = \{v, w\} \in E \\ n+1 & \text{otherwise} \end{cases}$
- bound: $B := n$

This transformation can be done in polynomial time. ✓
If I is a yes-instance of HC $\Rightarrow$ $f(I)$ is a yes-instance of TSP ✓

If $f(I)$ is a yes-instance of TSP
$\Rightarrow$ there is a roundtrip R in $f(I)$ that visits all cities and has at most length $\leq B = n$
$\Rightarrow$ this roundtrip contains every city at most once (otherwise it would be longer than n), so we can write $R = (c_{(1)}, c_{(2)}, c_{(3)}, \ldots, c_{(n)}, c_{(1)})$
Furthermore, $d(c_{(i)}, c_{(i+1)}) = 1$ for all $i = 1, \ldots, n-1$ and $d(c_{(n)}, c_{(1)}) = 1$, because otherwise R would be longer than n
$\Rightarrow$ The sequence of nodes in G $C := (v_{(1)}, v_{(2)}, v_{(3)}, \ldots, v_{(n)}, v_{(1)})$ is a cycle, because $\{c_{(i)}, c_{(i+1)}\}$ for all $i = 1, \ldots, n-1$ and $\{c_{(n)}, c_{(1)}\}$ are contained in E

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.

We define the polynomial transformation f of instance I of HC to instance $f(I)$ of TSP:

- cities: introduce one city c_i for each $v_i \in V$
- distances: set $d(v, w) := \begin{cases} 1 & \text{if } e = \{v, w\} \in E \\ n+1 & \text{otherwise} \end{cases}$
- bound: $B := n$

This transformation can be done in polynomial time. ✓
If I is a yes-instance of HC $\Rightarrow$ $f(I)$ is a yes-instance of TSP ✓

If $f(I)$ is a yes-instance of TSP
$\Rightarrow$ there is a roundtrip R in $f(I)$ that visits all cities and has at most length $\leq B = n$
$\Rightarrow$ this roundtrip contains every city at most once (otherwise it would be longer than n), so we can write $R = (c_{(1)}, c_{(2)}, c_{(3)}, \ldots, c_{(n)}, c_{(1)})$
Furthermore, $d(c_{(i)}, c_{(i+1)}) = 1$ for all $i = 1, \ldots, n-1$ and $d(c_{(n)}, c_{(1)}) = 1$, because otherwise R would be longer than n
$\Rightarrow$ The sequence of nodes in G $C := (v_{(1)}, v_{(2)}, v_{(3)}, \ldots, v_{(n)}, v_{(1)})$ is a cycle, because $\{c_{(i)}, c_{(i+1)}\}$ for all $i = 1, \ldots, n-1$ and $\{c_{(n)}, c_{(1)}\}$ are contained in E
$\Rightarrow$ I is a yes-instance of HC ✓

Proof: HC polynomially reduces to TSP

Let I be an instance of HP, consisting of undirected graph $G = (V, E)$.
We want to tranform I into an instance $f(I)$ of TSP.

We define the polynomial transformation f of instance I of HC to instance $f(I)$ of TSP:

- cities: introduce one city c_i for each $v_i \in V$
- distances: set $d(v, w) := \begin{cases} 1 & \text{if } e = \{v, w\} \in E \\ n+1 & \text{otherwise} \end{cases}$
- bound: $B := n$

This transformation can be done in polynomial time. ✓
If I is a yes-instance of HC $\Rightarrow$ $f(I)$ is a yes-instance of TSP ✓

If $f(I)$ is a yes-instance of TSP
$\Rightarrow$ there is a roundtrip R in $f(I)$ that visits all cities and has at most length $\leq B = n$
$\Rightarrow$ this roundtrip contains every city at most once (otherwise it would be longer than n), so we can write $R = (c_{(1)}, c_{(2)}, c_{(3)}, \ldots, c_{(n)}, c_{(1)})$
Furthermore, $d(c_{(i)}, c_{(i+1)}) = 1$ for all $i = 1, \ldots, n-1$ and $d(c_{(n)}, c_{(1)}) = 1$, because otherwise R would be longer than n
$\Rightarrow$ The sequence of nodes in G $C := (v_{(1)}, v_{(2)}, v_{(3)}, \ldots, v_{(n)}, v_{(1)})$ is a cycle, because $\{c_{(i)}, c_{(i+1)}\}$ for all $i = 1, \ldots, n-1$ and $\{c_{(n)}, c_{(1)}\}$ are contained in E
$\Rightarrow$ I is a yes-instance of HC ✓

Clique

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain a clique of size (at least) k?

Theorem

SAT $\leq_p$ CLIQUE.

Proof:

Clique

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain a clique of size (at least) k?

Theorem

SAT $\leq_p$ CLIQUE.

Proof: Given a set of clauses $\{c_1, c_2, \ldots, c_m\}$, over $x_1, \ldots, x_n$

Clique

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain a clique of size (at least) k?

Theorem

SAT $\leq_p$ CLIQUE.

Proof: Given a set of clauses $\{c_1, c_2, \ldots, c_m\}$, over $x_1, \ldots, x_n$
define instance instance of clique (our function f):

Clique

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain a clique of size (at least) k?

Theorem

SAT $\leq_p$ CLIQUE.

Proof: Given a set of clauses $\{c_1, c_2, \ldots, c_m\}$, over $x_1, \ldots, x_n$ define instance instance of clique (our function f):

$$V = \{(l, i) | l \text{ is a literal in } c_i\}$$
$$E = \{\{(l, i), (l', i')\} | l \neq \neg l' \wedge i \neq i'\}$$
$$k = m$$

Clique

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain a clique of size (at least) k?

Theorem

SAT $\leq_p$ CLIQUE.

Proof: Given a set of clauses $\{c_1, c_2, \ldots, c_m\}$, over $x_1, \ldots, x_n$ define instance instance of clique (our function f):

$$V = \{(l, i) | l \text{ is a literal in } c_i\}$$
$$E = \{\{(l, i), (l', i')\} | l \neq \neg l' \wedge i \neq i'\}$$
$$k = m$$

This is a polynomial-time transformation. ✓

Clique

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain a clique of size (at least) k?

Theorem

SAT $\leq_p$ CLIQUE.

Proof: Given a set of clauses $\{c_1, c_2, \ldots, c_m\}$, over $x_1, \ldots, x_n$
define instance instance of clique (our function f):

$$V = \{(l, i) | l \text{ is a literal in } c_i\}$$
$$E = \{\{(l, i), (l', i')\} | l \neq \neg l' \wedge i \neq i'\}$$
$$k = m$$

This is a polynomial-time transformation. ✓

Claim: There is a truth assignment such that all clauses are fulfilled $\Leftrightarrow$ there is a clique of size m

Clique

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain a clique of size (at least) k?

Theorem

SAT $\leq_p$ CLIQUE.

Proof: Given a set of clauses $\{c_1, c_2, \ldots, c_m\}$, over $x_1, \ldots, x_n$
define instance instance of clique (our function f):

$$V = \{(l, i) | l \text{ is a literal in } c_i\}$$
$$E = \{\{(l, i), (l', i')\} | l \neq \neg l' \wedge i \neq i'\}$$
$$k = m$$

This is a polynomial-time transformation. ✓

Claim: There is a truth assignment such that all clauses are fulfilled $\Leftrightarrow$ there is a clique of size m

$\Rightarrow$
Let t be a satisfying truth assignment.

Clique

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain a clique of size (at least) k?

Theorem

SAT $\leq_p$ CLIQUE.

Proof: Given a set of clauses $\{c_1, c_2, \ldots, c_m\}$, over $x_1, \ldots, x_n$
define instance instance of clique (our function f):

$$V = \{(l, i) | l \text{ is a literal in } c_i\}$$
$$E = \{\{(l, i), (l', i')\} | l \neq \neg l' \wedge i \neq i'\}$$
$$k = m$$

This is a polynomial-time transformation. ✓

Claim: There is a truth assignment such that all clauses are fulfilled $\Leftrightarrow$ there is a clique of size m

$\Rightarrow$
Let t be a satisfying truth assignment. For each i, denote by $l(i)$ the first literal in clause c_i that is true.

Clique

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain a clique of size (at least) k?

Theorem

SAT $\leq_p$ CLIQUE.

Proof: Given a set of clauses $\{c_1, c_2, \ldots, c_m\}$, over $x_1, \ldots, x_n$
define instance instance of clique (our function f):

$$V = \{(l, i) | l \text{ is a literal in } c_i\}$$
$$E = \{\{(l, i), (l', i')\} | l \neq \neg l' \wedge i \neq i'\}$$
$$k = m$$

This is a polynomial-time transformation. ✓

Claim: There is a truth assignment such that all clauses are fulfilled $\Leftrightarrow$ there is a clique of size m

$\Rightarrow$
Let t be a satisfying truth assignment. For each i, denote by $l(i)$ the first literal in clause c_i that is true. Then $V' := \{(l(i), i) : i = 1, \ldots, m\}$ is a clique, because any two nodes $(l(i), i)$, $(l(i'), i')$ are connected by an edge, as $i' \neq i$ and $l(i') \neq \neg l(i)$, as $l(i)$ and $\neg l(i)$ cannot be simultaneously true. ✓

Clique

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain a clique of size (at least) k?

Theorem

SAT $\leq_p$ CLIQUE.

Proof: Given a set of clauses $\{c_1, c_2, \ldots, c_m\}$, over $x_1, \ldots, x_n$

Clique

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain a clique of size (at least) k?

Theorem

SAT $\leq_p$ CLIQUE.

Proof: Given a set of clauses $\{c_1, c_2, \ldots, c_m\}$, over $x_1, \ldots, x_n$
define instance instance of clique (our function f):

Clique

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain a clique of size (at least) k?

Theorem

SAT $\leq_p$ CLIQUE.

Proof: Given a set of clauses $\{c_1, c_2, \ldots, c_m\}$, over $x_1, \ldots, x_n$ define instance instance of clique (our function f):

$$V = \{(l, i) | l \text{ is a literal in } c_i\}$$
$$E = \{\{(l, i), (l', i')\} | l \neq \neg l' \wedge i \neq i'\}$$
$$k = m$$

$\Leftarrow$
Let V' be a clique of size m.

Clique

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain a clique of size (at least) k?

Theorem

SAT $\leq_p$ CLIQUE.

Proof: Given a set of clauses $\{c_1, c_2, \ldots, c_m\}$, over $x_1, \ldots, x_n$
define instance instance of clique (our function f):

$$V = \{(l, i) | l \text{ is a literal in } c_i\}$$
$$E = \{\{(l, i), (l', i')\} | l \neq \neg l' \wedge i \neq i'\}$$
$$k = m$$

$\Leftarrow$
Let V' be a clique of size m. We define $t(l) = \text{true}$ if there is an i such that $(l, i) \in V'$. Or more precisely, if $l = x$ for variable x we define $t(x) = \text{true}$, if $l = \neg x$ we define $t(x) = \text{false}$.

Clique

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain a clique of size (at least) k?

Theorem

SAT $\leq_p$ CLIQUE.

Proof: Given a set of clauses $\{c_1, c_2, \ldots, c_m\}$, over $x_1, \ldots, x_n$
define instance instance of clique (our function f):

$$V = \{(l, i) | l \text{ is a literal in } c_i\}$$
$$E = \{\{(l, i), (l', i')\} | l \neq \neg l' \wedge i \neq i'\}$$
$$k = m$$

$\Leftarrow$
Let V' be a clique of size m. We define $t(l) = \text{true}$ if there is an i such that $(l, i) \in V'$. Or more precisely, if $l = x$ for variable x we define $t(x) = \text{true}$, if $l = \neg x$ we define $t(x) = \text{false}$.
Then due to the definition of the edges in G, the truth assignment is conflict-free (so far).

Clique

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain a clique of size (at least) k?

Theorem

SAT $\leq_p$ CLIQUE.

Proof: Given a set of clauses $\{c_1, c_2, \ldots, c_m\}$, over $x_1, \ldots, x_n$
define instance instance of clique (our function f):

$$V = \{(l, i) | l \text{ is a literal in } c_i\}$$
$$E = \{\{(l, i), (l', i')\} | l \neq \neg l' \wedge i \neq i'\}$$
$$k = m$$

$\Leftarrow$
Let V' be a clique of size m. We define $t(l) = \text{true}$ if there is an i such that $(l, i) \in V'$. Or more precisely, if $l = x$ for variable x we define $t(x) = \text{true}$, if $l = \neg x$ we define $t(x) = \text{false}$.
Then due to the definition of the edges in G, the truth assignment is conflict-free (so far).
For the remaining variables x we define $t(x) = \text{true}$.

Clique

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain a clique of size (at least) k?

Theorem

SAT $\leq_p$ CLIQUE.

Proof: Given a set of clauses $\{c_1, c_2, \ldots, c_m\}$, over $x_1, \ldots, x_n$
define instance instance of clique (our function f):

$$V = \{(l, i) | l \text{ is a literal in } c_i\}$$
$$E = \{\{(l, i), (l', i')\} | l \neq \neg l' \wedge i \neq i'\}$$
$$k = m$$

$\Leftarrow$
Let V' be a clique of size m. We define $t(l) = \text{true}$ if there is an i such that $(l, i) \in V'$. Or more precisely, if $l = x$ for variable x we define $t(x) = \text{true}$, if $l = \neg x$ we define $t(x) = \text{false}$.
Then due to the definition of the edges in G, the truth assignment is conflict-free (so far).
For the remaining variables x we define $t(x) = \text{true}$.
As the size of the clique V' is m, and we now that no two nodes (l, i) and (l', i) are incident in G, we know that the clique contains a node (l, i) for each $i = 1, \ldots, m$. So each clause c_i has at least one literal l with $t(l) = \text{true}$. ✓

Lemma

Reducibility is a transitive relation:
$X \leq_p Y$ and $Y \leq_p Z$ implies $X \leq_p Z$

Lemma

Reducibility is a transitive relation:
$X \leq_p Y$ and $Y \leq_p Z$ implies $X \leq_p Z$

Proof: by putting the two tranformations into series

NP-hardness

Definition

A decision problem X is *NP-hard*,
if **all** problems $Y \in NP$ can be reduced to it
(that is, if $Y \leq_p X$ holds for all $Y \in NP$)

NP-hardness

Definition

A decision problem X is *NP-hard*,
if **all** problems $Y \in NP$ can be reduced to it
(that is, if $Y \leq_p X$ holds for all $Y \in NP$)

Note that it is sufficient to show that $Y \leq_p X$ for **one NP-hard** problem Y.

Definition

A decision problem X is *NP-complete*,
if $X \in NP$ and X is NP-hard.

NP-hardness

Definition

A decision problem X is *NP-hard*,
if **all** problems $Y \in NP$ can be reduced to it
(that is, if $Y \leq_p X$ holds for all $Y \in NP$)

Note that it is sufficient to show that $Y \leq_p X$ for **one NP-hard** problem Y.

Definition

A decision problem X is *NP-complete*,
if $X \in NP$ and X is NP-hard.

Intuition:

- NP-complete problems are the hardest problems in NP
- Recall: NP is huge and contains tons of important problems
- Some people consider NP-complete problems to be intractable.

NP-hardness

Theorem

If one NP-complete problem X has a polynomial time algorithm then all NP-complete problems have polynomial time algorithms (and hence P=NP)

NP-hardness

Theorem

If one NP-complete problem X has a polynomial time algorithm then all NP-complete problems have polynomial time algorithms (and hence P=NP)

Why?

NP-hardness

Theorem

If one NP-complete problem X has a polynomial time algorithm then all NP-complete problems have polynomial time algorithms (and hence P=NP)

Why? Can reduce to X and then solve produced instance of X.

"I can't find an efficient algorithm, but neither can all these famous people."

Cook-Levin theorem (1971)

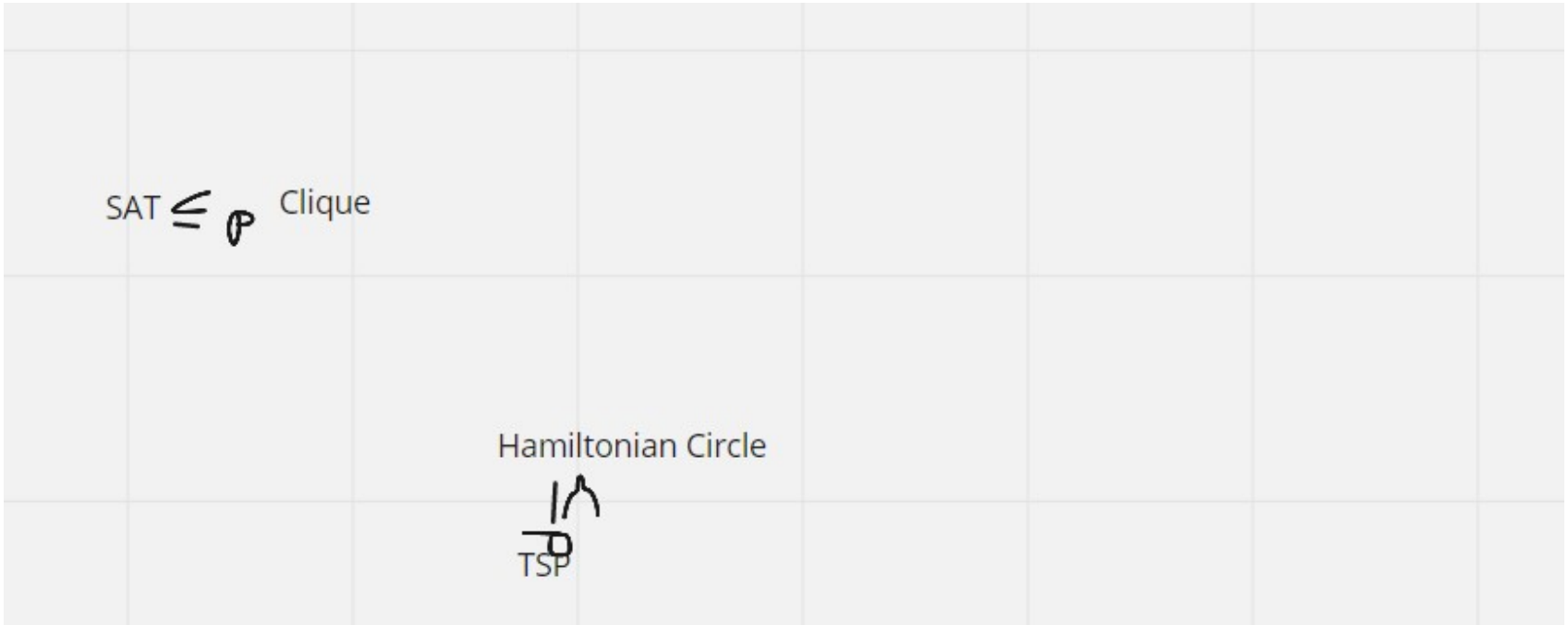
SAT is NP-complete.

see video for proof

- Stephen Cook (born 1939):
  American-Canadian computer scientist and mathematician
- Leonid Levin (born 1948):
  Russian computer scientist, discovered the result somewhat earlier

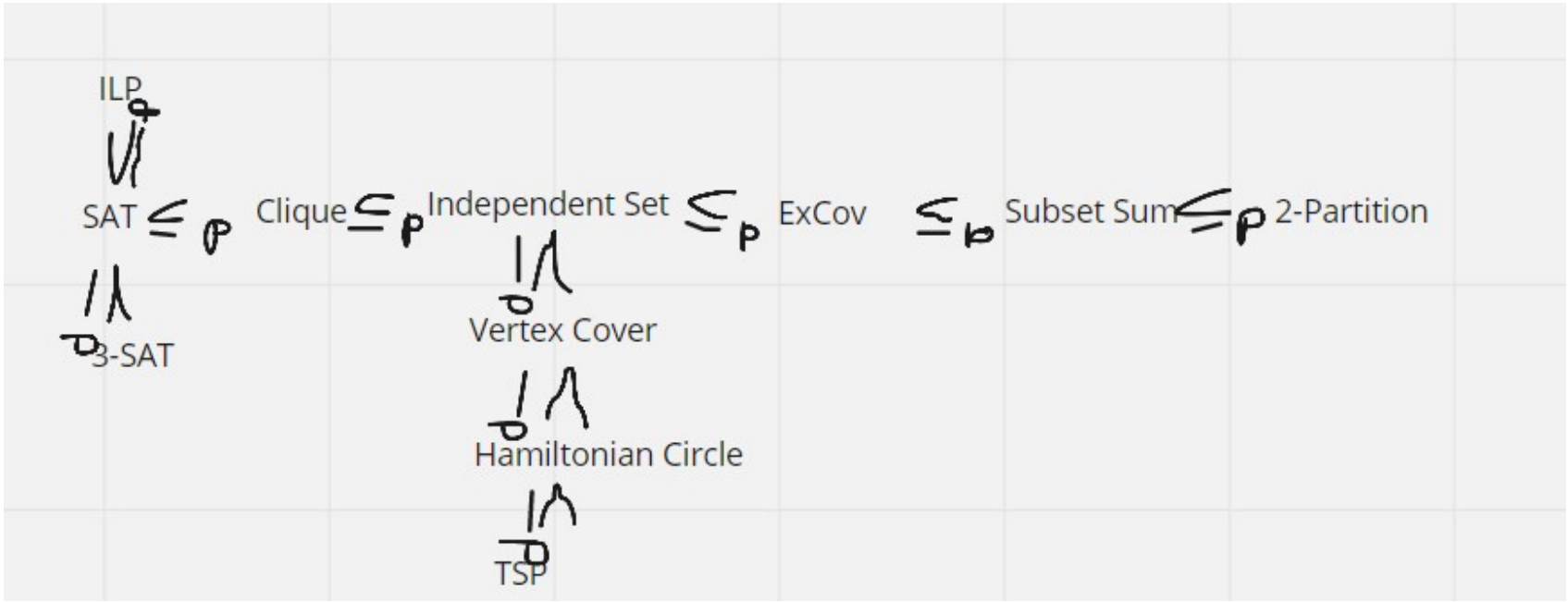
Videos

from here:



Videos

to here:



Recommended reading

Garey and Johnson. 'Algorithms and Complexity'
Lenstra and Rinnooy Kan. Computational complexity of discrete optimization problems.
Annals of Discrete Mathematics 4 (pp 121-140), 1979.
Electronic copy available on website

Cormen, Leiserson, Rivest and Stein 'Introduction to Algorithms':

- Chapter 1-3 (basics)
- Chapter 23 (minimum spanning trees)
- Chapter 34 (P, NP, NP-completeness, Cook-Levin theorem, reductions)