Algorithms and Complexity (AC)

Second Lecture (18th of September, 2023)

Paul Bouman & Tom van der Zanden (Based on slides by Gerhard Woeginger, Jesper Nederlof and Marie Schmidt)

September – November 2023

LNMB – Landelijk Netwerk Mathematische Besliskunde

Table of contents

Problem classes P and NP

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

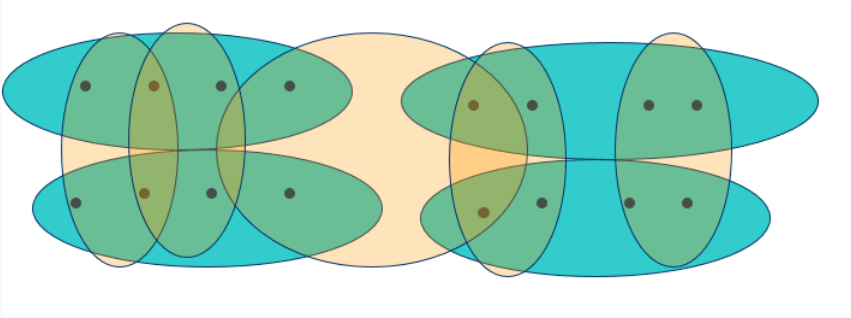
Question: do there exist some subsets S_i that form a partition of X?

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?

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

How to prove that something is hard?

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):

How to prove that something is hard?

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 to prove that something is hard?

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?

Bounds on 3-Satisfiability (3-SAT)

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?

Bounds on 3-Satisfiability (3-SAT)

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)

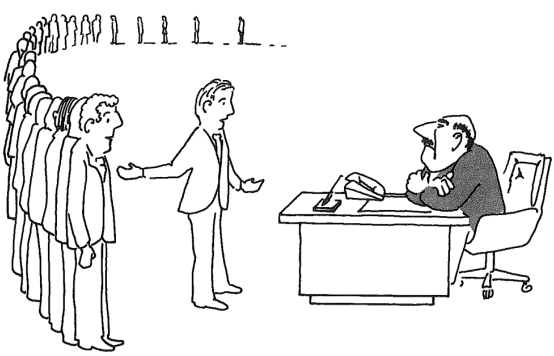
- Best found non-randomized algorithm (for 3-SAT) seems to be $O(1.3303^n)$
- Best found randomized algorithm similar ($O(1.306995^n)$?)
- No one so far has been able to prove that SAT is in $\Omega(n^c)$ for a $c > 1$!
  Or in different words: no one has been able to prove that SAT cannot be solved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

Questions?

Lower bounds are difficult to prove!

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

→ We look at a different approach.



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

Reductions

We denote by YES(X) the instances of decision problem X 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 instances 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

We denote by YES(X) the instances of decision problem X 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 instances 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

Proving reduction relations between problems

To prove that $X \leq_p Y$ for decision problems X and Y, we need to do the following:

- Find a transformation $f : X \to Y$
- Show that this transformation f can be done in polynomial time
- Show that if I is a yes-instance of $X \Rightarrow f(I)$ is a yes-instance of Y
- Show that if $f(I)$ is a yes-instance of $Y \Rightarrow I$ is a yes-instance of X

Hamiltonian cycle (HC)

Instance: an undirected graph $G = (V, E)$

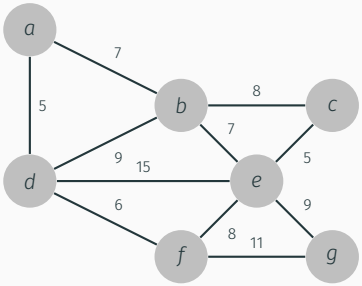
Question: does G contain a Hamiltonian cycle?

(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 HC, consisting of undirected graph $G = (V, E)$.

Proof: HC polynomially reduces to TSP

Let I be an instance of HC, 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 HC, 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 HC, 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

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 HC, 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

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 HC, 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

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 HC, 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

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: $B := n$

The transformation can be done in polynomial time. ✓

Proof: HC polynomially reduces to TSP

Let I be an instance of HC, consisting of undirected graph $G = (V, E)$. 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: $B := n$

The 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 HC, consisting of undirected graph $G = (V, E)$. 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: $B := n$

The 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 HC, consisting of undirected graph $G = (V, E)$. 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: $B := n$

The 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 HC, consisting of undirected graph $G = (V, E)$. 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: $B := n$

The 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 HC, consisting of undirected graph $G = (V, E)$. 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: $B := n$

The 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 HC, consisting of undirected graph $G = (V, E)$. 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: $B := n$

The 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 HC, consisting of undirected graph $G = (V, E)$. 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: $B := n$

The 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 HC, consisting of undirected graph $G = (V, E)$. 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: $B := n$

The 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 HC, consisting of undirected graph $G = (V, E)$. 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: $B := n$

The 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 HC, consisting of undirected graph $G = (V, E)$. 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: $B := n$

The 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 HC, consisting of undirected graph $G = (V, E)$. 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: $B := n$

The 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 HC, consisting of undirected graph $G = (V, E)$. 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: $B := n$

The 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 $\Rightarrow$ I is a yes-instance of HC ✓

Done!

Questions?

SAT $\leq_p$ CLIQUE

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:

SAT $\leq_p$ CLIQUE

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 SAT-instances with clauses $\{c_1, c_2, \ldots, c_m\}$, over $x_1, \ldots, x_n$

SAT $\leq_p$ CLIQUE

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 SAT-instances with clauses $\{c_1, c_2, \ldots, c_m\}$, over $x_1, \ldots, x_n$
we define our transformation function f converting SAT into Clique as:

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 SAT-instances with clauses $\{c_1, c_2, \ldots, c_m\}$, over $x_1, \ldots, x_n$
we define our transformation function f converting SAT into Clique as:

$$V = \{(l, i) \mid l \text{ is a literal in } c_i,\ i \in \{1, \ldots, m\}\}$$
$$E = \{\{(l, i), (l', i')\} \mid l \neq \neg l' \wedge i \neq i'\}$$
$$k = m$$

Intuition: connect pairs only if the literals are compatible and belong to different clauses.

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 SAT-instances with clauses $\{c_1, c_2, \ldots, c_m\}$, over $x_1, \ldots, x_n$
we define our transformation function f converting SAT into Clique as:

$$V = \{(l, i) \mid l \text{ is a literal in } c_i,\ i \in \{1, \ldots, m\}\}$$
$$E = \{\{(l, i), (l', i')\} \mid l \neq \neg l' \wedge i \neq i'\}$$
$$k = m$$

Intuition: connect pairs only if the literals are compatible and belong to different clauses.

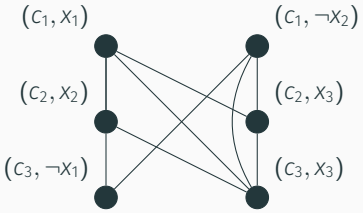
This is a polynomial-time transformation. ✓

Consider the following small example:

$c_1 = \{x_1, \neg x_2\}$

$c_2 = \{x_2, x_3\}$

$c_3 = \{\neg x_1, x_3\}$

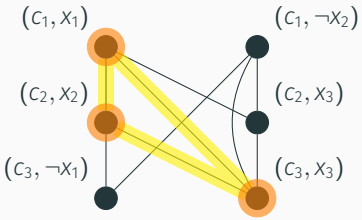


Consider the following small example:

$c_1 = \{x_1, \neg x_2\}$

$c_2 = \{x_2, x_3\}$

$c_3 = \{\neg x_1, x_3\}$



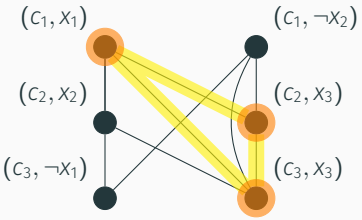
This 3-clique corresponds to the truth assignment where all variables are true.

Consider the following small example:

$c_1 = \{x_1, \neg x_2\}$

$c_2 = \{x_2, x_3\}$

$c_3 = \{\neg x_1, x_3\}$



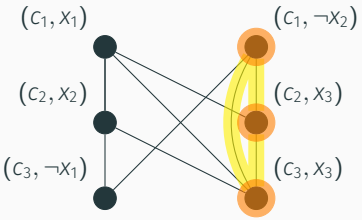
This 3-clique corresponds to $x_1 = \text{true}, x_3 = \text{true}$ and x_2 undetermined.

Consider the following small example:

$c_1 = \{x_1, \neg x_2\}$

$c_2 = \{x_2, x_3\}$

$c_3 = \{\neg x_1, x_3\}$



This 3-clique corresponds to $x_2 = \text{false}, x_3 = \text{true}$ and x_1 undetermined.

SAT $\leq_p$ CLIQUE

Clique

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

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?

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 clause C_i, find the first literal that is true according to t and select it in a clique V'.

Clique

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

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 clause C_i, find the first literal that is true according to t and select it in a clique V'.
- As we pick one node per clause C_i and t ensures only compatible literals are selected, V' is a Clique.

Clique

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

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 clause C_i, find the first literal that is true according to t and select it in a clique V'.
- As we pick one node per clause C_i and t ensures only compatible literals are selected, V' is a Clique.
- As there are k literals, V' is a k-clique.

SAT $\leq_p$ CLIQUE

Clique

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

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

$\Rightarrow$ ✓

$\Leftarrow$

- Let V' be a clique of size k.

SAT $\leq_p$ CLIQUE

Clique

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

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

$\Rightarrow$ ✓

$\Leftarrow$

- Let V' be a clique of size k.
- As the vertices belonging to the same clause are not connected, there must be exactly one vertex in V' corresponding to each clause.

Clique

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

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

$\Rightarrow$ ✓

$\Leftarrow$

- Let V' be a clique of size k.
- As the vertices belonging to the same clause are not connected, there must be exactly one vertex in V' corresponding to each clause.
- Construct a (partial) truth assignment satisfying the literals associated with the vertices in V'. As incompatible literals are disconnected they cannot be included in the same Clique.

Clique

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

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

$\Rightarrow$ ✓

$\Leftarrow$

- Let V' be a clique of size k.
- As the vertices belonging to the same clause are not connected, there must be exactly one vertex in V' corresponding to each clause.
- Construct a (partial) truth assignment satisfying the literals associated with the vertices in V'. As incompatible literals are disconnected they cannot be included in the same Clique.
- If any variables are not assigned a truth value, assign truth values arbitrarily (e.g. true).

Clique

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

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

$\Rightarrow$ ✓

$\Leftarrow$ ✓

Questions?

Reductions are Transitive

Lemma

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

Reductions are Transitive

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 and NP-completeness

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 to prove X is *NP-hard*.

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 to prove X is *NP-hard*.

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?

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 extra slides for the proof

- Stephen Cook (born 1939):

American-Canadian computer scientist and mathematician

- Leonid Levin (born 1948):

Russian computer scientist, discovered the result somewhat earlier

Cook-Levin theorem

For a instance $I \in X$ of a problem $X \in \text{NP}$, there is a procedure $f(I, C) : X \times \Sigma \to \{0, 1\}$ that can check in polynomial time if C is a valid certificate for I.

Observe:

- For any given instance I of input length n, the length of C and the number of elementary operations to perform $f(I, C)$ are poly(n).

Cook-Levin theorem

For a instance $I \in X$ of a problem $X \in \text{NP}$, there is a procedure $f(I, C) : X \times \Sigma \to \{0, 1\}$ that can check in polynomial time if C is a valid certificate for I.

Observe:

- For any given instance I of input length n, the length of C and the number of elementary operations to perform $f(I, C)$ are $\text{poly}(n)$.
- The number of intermediate states a computer or machine performing $f(I, C)$ can take is thus also $\text{poly}(n)$.

Cook-Levin theorem

For a instance $I \in X$ of a problem $X \in \text{NP}$, there is a procedure $f(I, C) : X \times \Sigma \to \{0, 1\}$ that can check in polynomial time if C is a valid certificate for I.

Observe:

- For any given instance I of input length n, the length of C and the number of elementary operations to perform $f(I, C)$ are poly(n).
- The number of intermediate states a computer or machine performing $f(I, C)$ can take is thus also poly(n).
- The way in which two consecutive states can change is constrained by the elementary operations of the computer/machine.

Cook-Levin theorem

For a instance $I \in X$ of a problem $X \in \text{NP}$, there is a procedure $f(I, C) : X \times \Sigma \to \{0, 1\}$ that can check in polynomial time if C is a valid certificate for I.

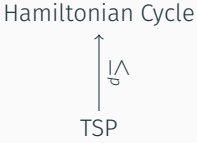
Observe:

- For any given instance I of input length n, the length of C and the number of elementary operations to perform $f(I, C)$ are poly(n).
- The number of intermediate states a computer or machine performing $f(I, C)$ can take is thus also poly(n).
- The way in which two consecutive states can change is constrained by the elementary operations of the computer/machine.
- With a lot of work we can use these intermediate states and the structure of the elementary operations to construct a large SAT-formula, where the boolean variables define C and the result of the formula is the output $f(I, C)$.

Questions?

Reductions

$$\text{SAT} \xleftarrow{\leq_p} \text{Clique}$$



NP-hardness proofs

NP-hardness: 3-SAT

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?

Theorem

3-SAT is NP-hard (and NP-complete).

NP-hardness: 3-SAT

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?

Theorem

3-SAT is NP-hard (and NP-complete).

Proof: We show SAT $\leq_p$ 3-SAT.

NP-hardness: 3-SAT

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?

Theorem

3-SAT is NP-hard (and NP-complete).

Proof: We show SAT $\leq_p$ 3-SAT.

Construct a new set of clauses C' and variables $X' = X \cup Y$ where Y are new auxiliary variables.

- Add all clauses $c \in C$ with three literals directly to C'.
- All clauses $c \in C$ with one or two literals are augmented by dummy variables. We add additional clauses that force the dummy variables to be false. (details ommitted)
- For all clauses $c \in C$ with more than three literals, we use a more elaborate trick.

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?

Consider a clause $c = (l_1, l_2, l_3, \ldots, l_k)$.

Introduce $k - 3$ auxiliary logical variables $a_1, a_2, \ldots, a_{k-3}$.

Split c into $k - 2$ clauses of three literals as follows:

$$(l_1, l_2, a_1),\ (\neg a_1, l_3, a_2),\ (\neg a_2, l_4, a_3),\ \ldots,\ (\neg a_{k-3}, l_{k-1}, l_k)$$

We can use the $k - 3$ auxiliary logical variables to satisfy all but one of the $k - 2$ new clauses. Therefore, we need to satisfy at least one clause with a literal originally occurring in c.

If we know which l_i is true, we can easily find truth values for the auxiliary variables such that the other clauses are satisfied. It suffices to consider them greedily in the order a_1 to a_{k-3}.

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?

Theorem

3-SAT is NP-hard (and NP-complete).

- Reduction can be performed in $O(nm)$ time, which is polynomial. ✓

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?

Theorem

3-SAT is NP-hard (and NP-complete).

- Reduction can be performed in $O(nm)$ time, which is polynomial. ✓
- SAT $\Rightarrow$ 3-SAT: use a satisfying truth assignment for SAT, and find satisfying truth values for the auxiliary variables easily. ✓

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

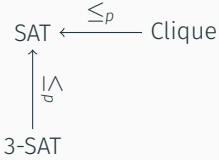
Question: does there exist a truth assignment for X that simultaneously satisfies all clauses in C?

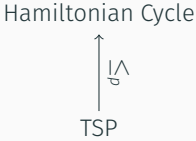
Theorem

3-SAT is NP-hard (and NP-complete).

- Reduction can be performed in $O(nm)$ time, which is polynomial. ✓
- SAT $\Rightarrow$ 3-SAT: use a satisfying truth assignment for SAT, and find satisfying truth values for the auxiliary variables easily. ✓
- SAT $\Leftarrow$ 3-SAT: remove all auxiliary variables from the satisfying truth assignment for 3-SAT to find a valid truth assignment for the SAT instance. ✓

Reductions





Questions?

NP-hardness: Integer programming

Integer programming (ILP) - Decision version

Instance: an integer matrix A; an integer vector b

Question: does there exist an integer vector y with $Ay \leq b$?

Theorem

ILP is NP-hard (and NP-complete).

NP-hardness: Integer programming

Integer programming (ILP) - Decision version

Instance: an integer matrix A; an integer vector b

Question: does there exist an integer vector y with $Ay \leq b$?

Theorem

ILP is NP-hard (and NP-complete).

Proof: by reduction from SAT.

Recall we have variables $X := \{x_1, \ldots, x_n\}$ and a set of clauses $C = \{c_1, \ldots, c_m\}$ over X.

We define a transformation f : SAT $\rightarrow$ ILP as follows:

- Create two ILP-variables y_i and y_{n+i} for each logical variable x_i with $i \in \{1, \ldots, n\}$. Introduce bijection $\pi(l)$ mapping literal l to index i if $l = x_i$, and to index $n + i$ if $l = \neg x_i$.
- Add constraints $y_i + y_{n+i} = 1$ for each $i \in \{1, \ldots, n\}$, and $y \geq 0$ (this implies y is binary).
- Add constraints $\sum_{l \in c} y_{\pi(l)} \geq 1$ for each clause $c \in C$.
- Convert this to ILP standard form to obtain matrix A and vector b.

Integer programming (ILP) - Decision version

Instance: an integer matrix A; an integer vector b

Question: does there exist an integer vector y with $Ay \leq b$?

Theorem

ILP is NP-hard (and NP-complete).

Proof: by reduction from SAT.

Recall we have variables $X := \{x_1, \ldots, x_n\}$ and a set of clauses $C = \{c_1, \ldots, c_m\}$ over X.

We define a transformation f : SAT $\rightarrow$ ILP as follows:

- Create two ILP-variables y_i and y_{n+i} for each logical variable x_i with $i \in \{1, \ldots, n\}$. Introduce bijection $\pi(l)$ mapping literal l to index i if $l = x_i$, and to index $n + i$ if $l = \neg x_i$.
- Add constraints $y_i + y_{n+i} = 1$ for each $i \in \{1, \ldots, n\}$, and $y \geq 0$ (this implies y is binary).
- Add constraints $\sum_{l \in c} y_{\pi(l)} \geq 1$ for each clause $c \in C$.
- Convert this to ILP standard form to obtain matrix A and vector b.

This can be performed in polynomial time. ✓

This can be performed in polynomial time. ✓

To prove: If I is a YES-instance of SAT $\Rightarrow f(I)$ is a YES-instance of ILP.

This can be performed in polynomial time. ✓

To prove: If I is a YES-instance of SAT $\Rightarrow f(I)$ is a YES-instance of ILP.

- Since I is a YES-instance, there is a truth-assignment t that satisfies all clauses of I.

This can be performed in polynomial time. ✓

To prove: If I is a YES-instance of SAT $\Rightarrow f(I)$ is a YES-instance of ILP.

- Since I is a YES-instance, there is a truth-assignment t that satisfies all clauses of I.
- We define an integer vector y where $y_i = 1$ if $t(x_i)$, and $y_{n+i} = 1$ if $\neg t(x_i)$, and 0 otherwise.

This can be performed in polynomial time. ✓

To prove: If I is a YES-instance of SAT $\Rightarrow f(I)$ is a YES-instance of ILP.

- Since I is a YES-instance, there is a truth-assignment t that satisfies all clauses of I.
- We define an integer vector y where $y_i = 1$ if $t(x_i)$, and $y_{n+i} = 1$ if $\neg t(x_i)$, and 0 otherwise. This satisfies the $y_i + y_{n+i} = 1$ constraints and $y \geq 0$ constraints by definition.

This can be performed in polynomial time. ✓

To prove: If I is a YES-instance of SAT $\Rightarrow f(I)$ is a YES-instance of ILP.

- Since I is a YES-instance, there is a truth-assignment t that satisfies all clauses of I.
- We define an integer vector y where $y_i = 1$ if $t(x_i)$, and $y_{n+i} = 1$ if $\neg t(x_i)$, and 0 otherwise. This satisfies the $y_i + y_{n+i} = 1$ constraints and $y \geq 0$ constraints by definition.
- Furthermore, as t satisfies each clause $c_i \in C$, each clause c_i will contain at least one literal which is made true by t.

This can be performed in polynomial time. ✓

To prove: If I is a YES-instance of SAT $\Rightarrow f(I)$ is a YES-instance of ILP.

- Since I is a YES-instance, there is a truth-assignment t that satisfies all clauses of I.
- We define an integer vector y where $y_i = 1$ if $t(x_i)$, and $y_{n+i} = 1$ if $\neg t(x_i)$, and 0 otherwise. This satisfies the $y_i + y_{n+i} = 1$ constraints and $y \geq 0$ constraints by definition.
- Furthermore, as t satisfies each clause $c_i \in C$, each clause c_i will contain at least one literal which is made true by t.
- Consequently, for each clause the corresponding ILP-constraint will contain at least one true literal l' which implies $y_{\pi(l')} = 1$.

This can be performed in polynomial time. ✓
If I is a YES-instance of SAT $\Rightarrow$ $f(I)$ is a YES-instance of ILP. ✓

- Since I is a YES-instance, there is a truth-assignment t that satisfies all clauses of I.
- We define an integer vector y where $y_i = 1$ if $t(x_i)$, and $y_{n+i} = 1$ if $\neg t(x_i)$, and 0 otherwise. This satisfies the $y_i + y_{n+i} = 1$ constraints and $y \geq 0$ constraints by definition.
- Furthermore, as t satisfies each clause $c_i \in C$, each clause c_i will contain at least one literal which is made true by t.
- Consequently, for each clause the corresponding ILP-constraint will contain at least one true literal l' which implies $y_{\pi(l')} = 1$.
- Hence, the sum of the variables of each constraint related to a clause will be at least 1, satisfying the constraints related to the clauses.

This can be performed in polynomial time. ✓

If I is a YES-instance of SAT $\Rightarrow$ $f(I)$ is a YES-instance of ILP. ✓

To prove: If $f(I)$ is a YES-instance of ILP, I is a YES-instance of SAT.

This can be performed in polynomial time. ✓

If I is a YES-instance of SAT $\Rightarrow$ $f(I)$ is a YES-instance of ILP. ✓

To prove: If $f(I)$ is a YES-instance of ILP, I is a YES-instance of SAT.

- As $f(I)$ is a YES-instance, there must be a feasible vector $y \in \{0,1\}^{2n}$ for $f(I)$.

This can be performed in polynomial time. ✓

If I is a YES-instance of SAT $\Rightarrow$ $f(I)$ is a YES-instance of ILP. ✓

To prove: If $f(I)$ is a YES-instance of ILP, I is a YES-instance of SAT.

- As $f(I)$ is a YES-instance, there must be a feasible vector $y \in \{0,1\}^{2n}$ for $f(I)$.
- Construct a truth assignment t from this vector y where $t(x_i) = \text{true} \Leftrightarrow y_i = 1$.

This can be performed in polynomial time. ✓

If I is a YES-instance of SAT $\Rightarrow$ $f(I)$ is a YES-instance of ILP. ✓

To prove: If $f(I)$ is a YES-instance of ILP, I is a YES-instance of SAT.

- As $f(I)$ is a YES-instance, there must be a feasible vector $y \in \{0,1\}^{2n}$ for $f(I)$.
- Construct a truth assignment t from this vector y where $t(x_i) = \text{true} \Leftrightarrow y_i = 1$.
- As every clause $c \in C$ of I corresponds to a constraint in $f(I)$ that sums to at least one, we can find a variable index i' such that $y_{i'} = 1$ and for which literal $\pi^{-1}(i')$ is true wrt t.

This can be performed in polynomial time. ✓
If I is a YES-instance of SAT $\Rightarrow$ $f(I)$ is a YES-instance of ILP. ✓
If $f(I)$ is a YES-instance of ILP, I is a YES-instance of SAT. ✓

- As $f(I)$ is a YES-instance, there must be a feasible vector $y \in \{0,1\}^{2n}$ for $f(I)$.
- Construct a truth assignment t from this vector y where $t(x_i) = \text{true} \Leftrightarrow y_i = 1$.
- As every clause $c \in C$ of I corresponds to a constraint in $f(I)$ that sums to at least one, we can find a variable index i' such that $y_{i'} = 1$ and for which literal $\pi^{-1}(i')$ is true wrt t.
- Hence, every clause must contain at least one literal that satisfies it.

NP-hardness: Integer programming

Integer programming (ILP)

Instance: an integer matrix A; an integer vector b

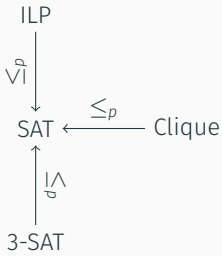
Question: does there exist an integer vector y with $Ay \leq b$?

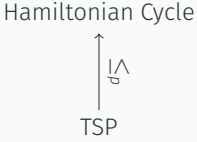
Theorem

SAT $\leq_p$ ILP, and therefore ILP is NP-hard (and NP-complete).

Consequence: Every problem in NP can be modelled as an ILP.

Reductions


ILP
$\leq_p$
SAT
$\leq_p$
Clique
$\leq_p$
3-SAT


Hamiltonian Cycle
$\leq_p$
TSP

Questions?

Clique

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

Independent set (IS)

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain an independent set of size (at least) k?
(a set of vertices that does not span any edge)

Vertex cover (VC)

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain a vertex cover of size (at most) k?
(a set of vertices that touches every edge)

Complement Graph

For a graph $G = (V, E)$, the complement graph $\bar{G} = (V, \bar{E})$ is characterized by:

$$\forall (v, w) \in V^2 : (v, w) \in E \Leftrightarrow (v, w) \notin \bar{E}$$

Theorem

The following three statements are equivalent:

1. $S \subseteq V$ *is a Clique of* G
2. S *is an Independent Set of* $\bar{G}$
3. $V - S$ *is a Vertex Cover of* $\bar{G}$

(Proof is omitted, but many resources exist, e.g. the Combinatorial Optimization book)

NP-hardness: Clique

Clique

Instance: a graph $G = (V, E)$; an integer k

Question: does G contain a clique of size (at least) k?

Theorem

CLIQUE is NP-hard (and NP-complete).

Proof: SAT is NP-hard and SAT $\leq_p$ CLIQUE. (the reduction was discussed earlier)

Independent set (IS)

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain an independent set of size (at least) k?
(a set of vertices that does not span any edge)

Theorem

Independent Set is NP-hard (and NP-complete).

Proof:

Independent set (IS)

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain an independent set of size (at least) k?
(a set of vertices that does not span any edge)

Theorem

Independent Set is NP-hard (and NP-complete).

Proof:

$S \subseteq V$ is a Clique of $G \Leftrightarrow S$ is an Independent Set of $\bar{G}$

Vertex cover (VC)

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain a vertex cover of size (at most) k?
(a set of vertices that touches every edge)

Theorem

Vertex Cover is NP-hard (and NP-complete)

Proof:

Vertex cover (VC)

Instance: a graph $G = (V, E)$; an integer k
Question: does G contain a vertex cover of size (at most) k?
(a set of vertices that touches every edge)

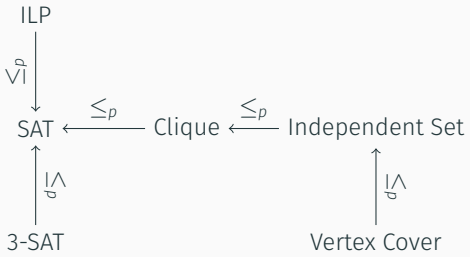
Theorem

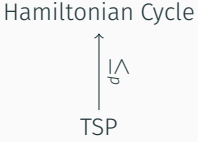
Vertex Cover is NP-hard (and NP-complete)

Proof:

S is an Independent Set of $\bar{G} \Leftrightarrow V - S$ is a Vertex Cover of $\bar{G}$

Reductions


ILP
$\leq_p$
SAT
$\leq_p$
Clique
$\leq_p$
Independent Set
$\leq_p$
3-SAT
Vertex Cover


Hamiltonian Cycle
$\leq_p$
TSP

Questions?

NP-hardness: Exact cover

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?

Theorem

(Ex-Cov) is NP-hard (and NP-complete).

Proof:

NP-hardness: Exact cover

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?

Theorem

(Ex-Cov) is NP-hard (and NP-complete).

Proof: by reduction from IS.

NP-hardness: Exact cover

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?

Theorem

(Ex-Cov) is NP-hard (and NP-complete).

Proof: by reduction from IS.

Let (G, k) with $G = (V, E)$ be an instance of IS.
We say that $E = \{e_1, \ldots, e_m\}$ and E_v is the set of edges that have vertex $v \in V$ as an endpoint.

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?

Theorem

(Ex-Cov) is NP-hard (and NP-complete).

Proof: by reduction from IS.

Let (G, k) with $G = (V, E)$ be an instance of IS.
We say that $E = \{e_1, \ldots, e_m\}$ and E_v is the set of edges that have vertex $v \in V$ as an endpoint. Define an instance of (Ex-Cov) as follows:

- Introduce set elements $X := \{1, \ldots, m, m+1, \ldots, m+k\}$
- For each $v \in V$ and $h \in \{1, \ldots, k\}$, define a subset $S^k_{vh} = \{i \mid \forall e_i \in E_v\} \cup \{m+h\}$
- For each $e_i \in E$, define a subset $S^e_i = \{i\}$

We can select any of the generated sets:

$$S = \left\{ S^k_{vh} \mid \forall v \in V, \forall h \in \{1, \ldots, k\} \right\} \cup \{S^e_i \mid \forall e_i \in E\}$$

As we have $|E| + k$ elements and $k|V| + |E|$ subsets, this transformation can be performed in polynomial time.

As we have $|E| + k$ elements and $k|V| + |E|$ subsets, this transformation can be performed in polynomial time. ✓

As we have $|E| + k$ elements and $k|V| + |E|$ subsets, this transformation can be performed in polynomial time. ✓

Show: Independent Set $\Leftarrow$ Exact Cover

As we have $|E| + k$ elements and $k|V| + |E|$ subsets, this transformation can be performed in polynomial time. ✓

Show: Independent Set $\Leftarrow$ Exact Cover

We argue that if S is a solution to (Ex-Cov) $V^* = \{v \mid \forall S^k_{vh} \in S, \forall h \in \{1, \ldots, k\}\}$ is an independent set of size k.

As we have $|E| + k$ elements and $k|V| + |E|$ subsets, this transformation can be performed in polynomial time. ✓

Show: Independent Set $\Leftarrow$ Exact Cover

We argue that if S is a solution to (Ex-Cov) $V^* = \{v \mid \forall S_{vh}^k \in S, \forall h \in \{1, \ldots, k\}\}$ is an independent set of size k.

- As the elements $\{m + 1, \ldots, m + k\}$ occur at most once in any subset, we have $|V^*| = k$.

As we have $|E| + k$ elements and $k|V| + |E|$ subsets, this transformation can be performed in polynomial time. ✓

Show: Independent Set $\Leftarrow$ Exact Cover

We argue that if S is a solution to (Ex-Cov) $V^* = \{v \mid \forall S^k_{vh} \in S, \forall h \in \{1, \ldots, k\}\}$ is an independent set of size k.

- As the elements $\{m + 1, \ldots, m + k\}$ occur at most once in any subset, we have $|V^*| = k$.
- For any edge $e_j \in E = (v, u)$, there can not be two sets S^k_{vh} and $S^k_{uh'}$ in S, as both sets will contain element j.

As we have $|E| + k$ elements and $k|V| + |E|$ subsets, this transformation can be performed in polynomial time. ✓

Show: Independent Set $\Leftarrow$ Exact Cover

We argue that if S is a solution to (Ex-Cov) $V^* = \{v \mid \forall S^k_{vh} \in S, \forall h \in \{1, \ldots, k\}\}$ is an independent set of size k.

- As the elements $\{m + 1, \ldots, m + k\}$ occur at most once in any subset, we have $|V^*| = k$.
- For any edge $e_j \in E = (v, u)$, there can not be two sets S^k_{vh} and $S^k_{uh'}$ in S, as both sets will contain element j.
- Hence V^* must be an independent set.

As we have $|E| + k$ elements and $k|V| + |E|$ subsets, this transformation can be performed in polynomial time. ✓

Independent Set $\Leftarrow$ Exact Cover ✓

Show: Independent Set $\Rightarrow$ Exact Cover

As we have $|E| + k$ elements and $k|V| + |E|$ subsets, this transformation can be performed in polynomial time. ✓

Independent Set $\Leftarrow$ Exact Cover ✓

Show: Independent Set $\Rightarrow$ Exact Cover

We argue that if a sequence $V^* = [v_1, \ldots, v_k]$ length k contains an independent set,

$$S = \left\{ S^k_{ih} \mid v_i \in V^* \text{ at position } h \right\} \cup \left\{ S^e_j \mid \forall e_j = (v, u) \in E, v \notin V^*, u \notin V^* \right\}$$

is an exact cover.

As we have $|E| + k$ elements and $k|V| + |E|$ subsets, this transformation can be performed in polynomial time. ✓

Independent Set $\Leftarrow$ Exact Cover ✓

Show: Independent Set $\Rightarrow$ Exact Cover

We argue that if a sequence $V^* = [v_1, \ldots, v_k]$ length k contains an independent set,

$$S = \left\{ S^k_{ih} \mid v_i \in V^* \text{ at position } h \right\} \cup \left\{ S^e_j \mid \forall e_j = (v, u) \in E, v \notin V^*, u \notin V^* \right\}$$

is an exact cover.

- As V^* is a sequence of length k, exactly one S^k_{ih} for each $h \in \{1, \ldots k\}$ is in S.
  Hence all elements $m + 1, \ldots, m + k$ are covered.

As we have $|E| + k$ elements and $k|V| + |E|$ subsets, this transformation can be performed in polynomial time. ✓

Independent Set $\Leftarrow$ Exact Cover ✓

Show: Independent Set $\Rightarrow$ Exact Cover

We argue that if a sequence $V^* = [v_1, \ldots, v_k]$ length k contains an independent set,

$$S = \left\{ S^k_{ih} \mid v_i \in V^* \text{ at position } h \right\} \cup \left\{ S^e_j \mid \forall e_j = (v, u) \in E, v \notin V^*, u \notin V^* \right\}$$

is an exact cover.

- As V^* is a sequence of length k, exactly one S^k_{ih} for each $h \in \{1, \ldots k\}$ is in S.
  Hence all elements $m + 1, \ldots, m + k$ are covered.
- These sets can not contain any overlapping edge indices as V^* is an independent set.

As we have $|E| + k$ elements and $k|V| + |E|$ subsets, this transformation can be performed in polynomial time. ✓

Independent Set $\Leftarrow$ Exact Cover ✓

Show: Independent Set $\Rightarrow$ Exact Cover

We argue that if a sequence $V^* = [v_1, \ldots, v_k]$ length k contains an independent set,

$$S = \left\{ S^k_{ih} \mid v_i \in V^* \text{ at position } h \right\} \cup \left\{ S^e_j \mid \forall e_j = (v, u) \in E, v \notin V^*, u \notin V^* \right\}$$

is an exact cover.

- As V^* is a sequence of length k, exactly one S^k_{ih} for each $h \in \{1, \ldots k\}$ is in S.
  Hence all elements $m + 1, \ldots, m + k$ are covered.
- These sets can not contain any overlapping edge indices as V^* is an independent set.
- For any $e_j \in E$ of which neither endpoint is in V^*, the set S^e_j is added to S.
  Hence all elements $1, \ldots, m$ are also covered.

As we have $|E| + k$ elements and $k|V| + |E|$ subsets, this transformation can be performed in polynomial time. ✓

Independent Set $\Leftarrow$ Exact Cover ✓

Show: Independent Set $\Rightarrow$ Exact Cover

We argue that if a sequence $V^* = [v_1, \ldots, v_k]$ length k contains an independent set,

$$S = \left\{ S^k_{ih} \mid v_i \in V^* \text{ at position } h \right\} \cup \left\{ S^e_j \mid \forall e_j = (v, u) \in E, v \notin V^*, u \notin V^* \right\}$$

is an exact cover.

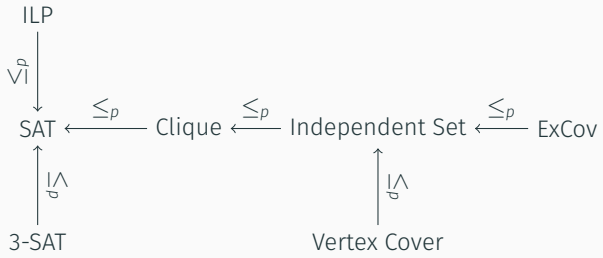
- As V^* is a sequence of length k, exactly one S^k_{ih} for each $h \in \{1, \ldots k\}$ is in S.
  Hence all elements $m + 1, \ldots, m + k$ are covered.
- These sets can not contain any overlapping edge indices as V^* is an independent set.
- For any $e_j \in E$ of which neither endpoint is in V^*, the set S^e_j is added to S.
  Hence all elements $1, \ldots, m$ are also covered.
- Since the sets S^e do not overlap with each other and are chosen such that they do not overlap with the chosen S^k sets, S must be an exact cover.

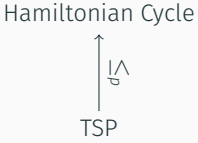
As we have $|E| + k$ elements and $k|V| + |E|$ subsets, this transformation can be performed in polynomial time. ✓

Independent Set $\Leftarrow$ Exact Cover ✓

Independent Set $\Rightarrow$ Exact Cover ✓

Reductions


ILP
$\leq_p$
SAT
$\leq_p$
Clique
$\leq_p$
Independent Set
$\leq_p$
ExCov
$\leq_p$
3-SAT
Vertex Cover


Hamiltonian Cycle
$\leq_p$
TSP

Questions?

Hamiltonian cycle (HC)

Instance: a graph (V, E)

Question: does this graph contain a Hamiltonian cycle?

Theorem

HC is NP-complete.

Hamiltonian cycle (HC)

Instance: a graph (V, E)

Question: does this graph contain a Hamiltonian cycle?

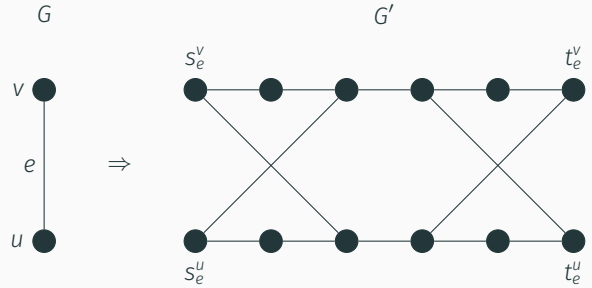
Theorem

HC is NP-complete.

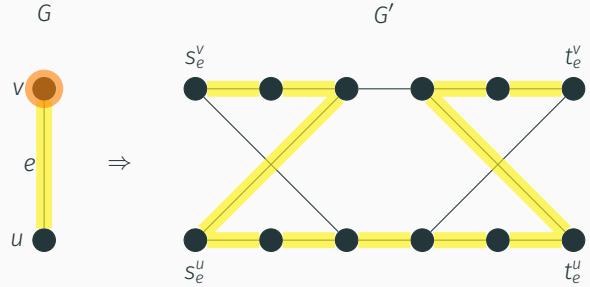
Proof: Easy to see: in NP.

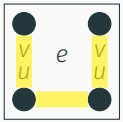
To show NP-hard: reduction from Vertex Cover.

For each edge in the original graph, create a *cover-testing gadget/component*.

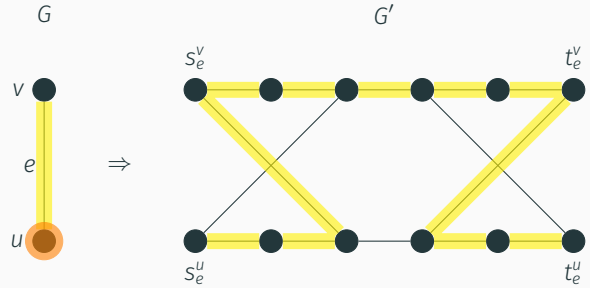


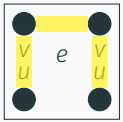
For each edge in the original graph, create a *cover-testing gadget/component*.



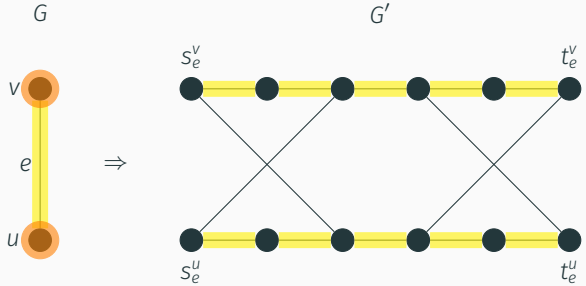


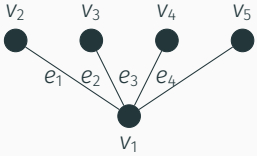
For each edge in the original graph, create a *cover-testing gadget/component*.

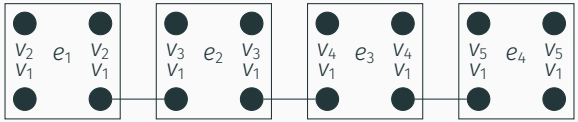




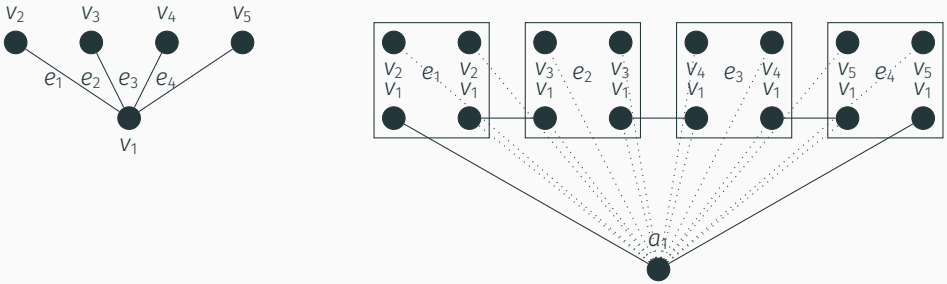
For each edge in the original graph, create a *cover-testing gadget/component*.





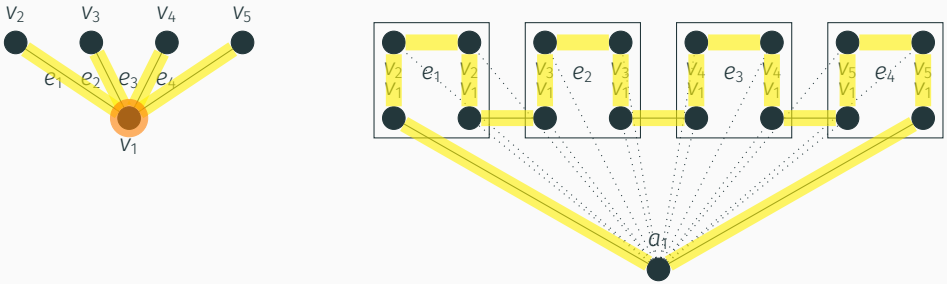


For each vertex v_i of degree 2 or greater, order the edges arbitrarily and connect the corresponding gadgets at their v_i side in a chain.



For each vertex v_i of degree 2 or greater, order the edges arbitrarily and connect the corresponding gadgets at their v_i side in a chain.

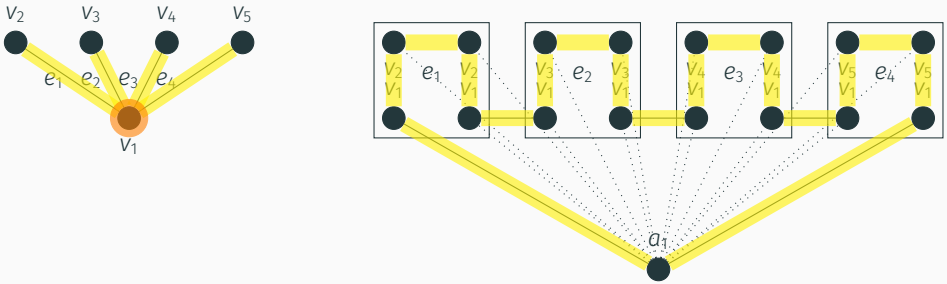
Also, introduce k *selector* vertices $\{a_1, \ldots, a_k\}$ that we connect to all terminals of the gadgets and each other (but not to the gadgets' internal nodes!).



For each vertex v_i of degree 2 or greater, order the edges arbitrarily and connect the corresponding gadgets at their v_i side in a chain.

Also, introduce k *selector* vertices $\{a_1, \ldots, a_k\}$ that we connect to all terminals of the gadgets and each other (but not to the gadgets' internal nodes!).

The idea is that if we select a vertex v in the VC graph, we can create a path through all the gadgets of edges connected to v.

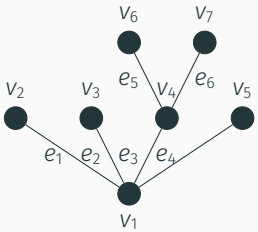


For each vertex v_i of degree 2 or greater, order the edges arbitrarily and connect the corresponding gadgets at their v_i side in a chain.

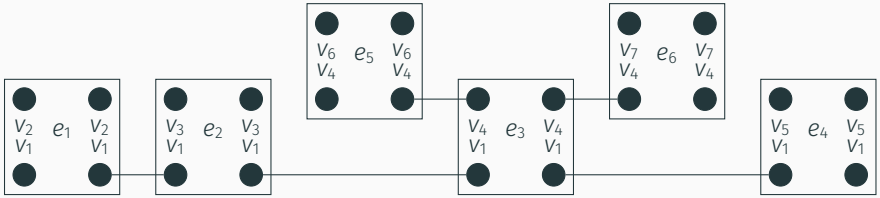
Also, introduce k *selector* vertices $\{a_1, \ldots, a_k\}$ that we connect to all terminals of the gadgets and each other (but not to the gadgets' internal nodes!).

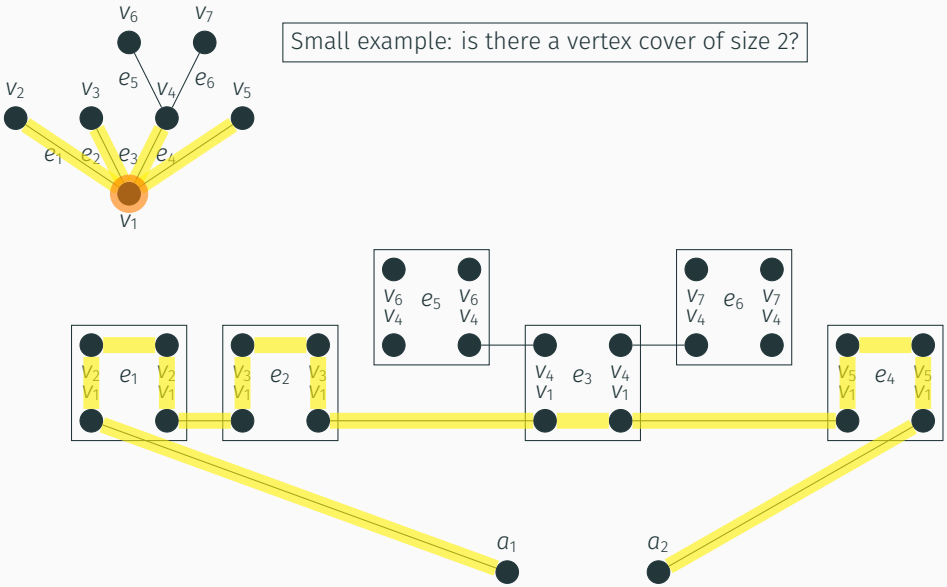
The idea is that if we select a vertex v in the VC graph, we can create a path through all the gadgets of edges connected to v.

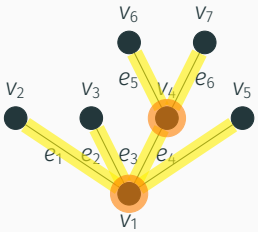
Since the gadgets have a constant size, this can be done in polynomial time. ✓



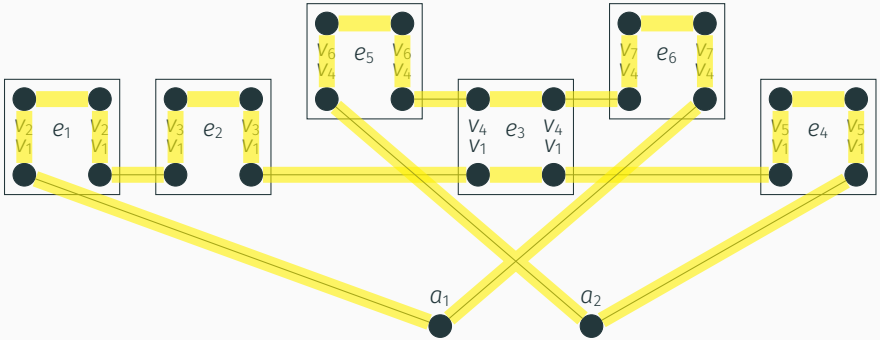
Small example: is there a vertex cover of size 2?

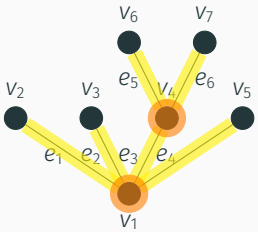



v_6
v_7
e_5
e_6
v_2
v_3
v_4
v_5
e_1
e_2
e_3
e_4
v_1
Small example: is there a vertex cover of size 2?
v_6 v_4
e_5
v_6 v_4
v_7 v_4
e_6
v_7 v_4
v_2 v_1
e_1
v_2 v_1
v_3 v_1
e_2
v_3 v_1
v_4 v_1
e_3
v_4 v_1
v_5 v_1
e_4
v_5 v_1
a_1
a_2



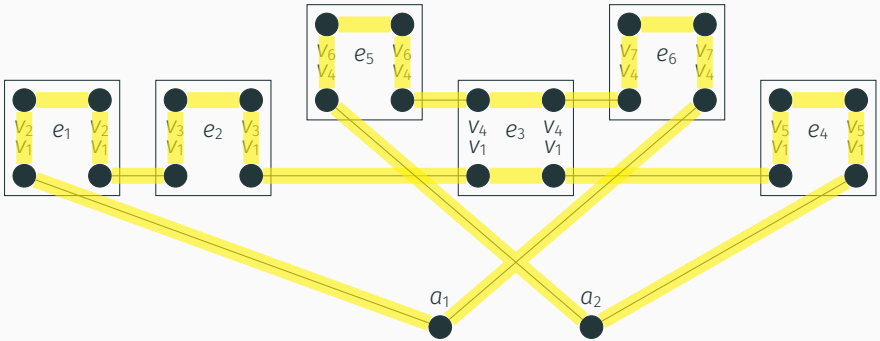
Small example: is there a vertex cover of size 2?





Small example: is there a vertex cover of size 2?

YES!



NP-hardness: Hamiltonian cycle / TSP

Show: G' has a Hamiltonian cycle $C \Rightarrow G$ has a vertex cover of size k:

Show: G' has a Hamiltonian cycle $C \Rightarrow G$ has a vertex cover of size k:
Consider a subpath of the cycle that starts at an a_i and ends at an a_j ($j \neq i$) without passing a node a_k:

Show: G' has a Hamiltonian cycle $C \Rightarrow G$ has a vertex cover of size k:
Consider a subpath of the cycle that starts at an a_i and ends at an a_j ($j \neq i$) without passing a node a_k:
This subpath passes through a set of cover-testing components all associated with a specific v.

Show: G' has a Hamiltonian cycle $C \Rightarrow G$ has a vertex cover of size k:
Consider a subpath of the cycle that starts at an a_i and ends at an a_j ($j \neq i$) without passing a node a_k:
This subpath passes through a set of cover-testing components all associated with a specific v.
Therefore, the k vertices $\{a_1, a_2, \ldots, a_k\}$ divide the Hamiltonian circuit into k paths, each path i corresponding to a vertex v: the one that corresponds to the node $(v, e_{v[1]}, 1)$ visited right after a_i.

Show: G' has a Hamiltonian cycle $C \Rightarrow G$ has a vertex cover of size k:
Consider a subpath of the cycle that starts at an a_i and ends at an a_j ($j \neq i$) without passing a node a_k:
This subpath passes through a set of cover-testing components all associated with a specific v.
Therefore, the k vertices $\{a_1, a_2, \ldots, a_k\}$ divide the Hamiltonian circuit into k paths, each path i corresponding to a vertex v: the one that corresponds to the node $(v, e_{v[1},1)$ visited right after a_i.
These vertices form a vertex cover:

Show: G' has a Hamiltonian cycle $C \Rightarrow G$ has a vertex cover of size k:
Consider a subpath of the cycle that starts at an a_i and ends at an a_j ($j \neq i$) without passing a node a_k:
This subpath passes through a set of cover-testing components all associated with a specific v.
Therefore, the k vertices $\{a_1, a_2, \ldots, a_k\}$ divide the Hamiltonian circuit into k paths, each path i corresponding to a vertex v: the one that corresponds to the node $(v, e_{v[1}, 1)$ visited right after a_i.
These vertices form a vertex cover:
If there was an edge that was not covered in E, then its 'cover-testing component' would not be visited. ✓

NP-hardness: Hamiltonian cycle / TSP

Show: G' has a Hamiltonian cycle $C \Leftarrow G$ has a vertex cover of size k:

NP-hardness: Hamiltonian cycle / TSP

Show: G' has a Hamiltonian cycle $C \Leftarrow G$ has a vertex cover of size k:

Let $V^* = \{v_1, v_2, \ldots, v_k\}$ be a vertex cover (wlog $|V^*| = k$).

NP-hardness: Hamiltonian cycle / TSP

Show: G' has a Hamiltonian cycle $C \Leftarrow G$ has a vertex cover of size k:

Let $V^* = \{v_1, v_2, \ldots, v_k\}$ be a vertex cover (wlog $|V^*| = k$).

We construct a cycle C as follows:

- Start at a selector vertex a_1 and add the the chain of gadgets corresponding to node v_1 to C.
- If both endpoints of an edge are contained in V^*, add only the vertices of the gadget on the side of v_1. Otherwise, add all the vertices of the gadget.
- Continue to the next selector vertex and repeat this process for the next v_i and a_i.
- Finally, complete the cycle by connecting back to a_1.

NP-hardness: Hamiltonian cycle / TSP

Show: G' has a Hamiltonian cycle $C \Leftarrow G$ has a vertex cover of size k:

Let $V^* = \{v_1, v_2, \ldots, v_k\}$ be a vertex cover (wlog $|V^*| = k$).

We construct a cycle C as follows:

- Start at a selector vertex a_1 and add the the chain of gadgets corresponding to node v_1 to C.
- If both endpoints of an edge are contained in V^*, add only the vertices of the gadget on the side of v_1. Otherwise, add all the vertices of the gadget.
- Continue to the next selector vertex and repeat this process for the next v_i and a_i.
- Finally, complete the cycle by connecting back to a_1.

In each subpath starting at selector vertex a_i, we visit all gadgets corresponding to the edges connected to v_i either fully or partially. If they are visited partially, the other half of the gadget will be visited in the subpath corresponding to the other endpoint of the gadget's edge.

Show: G' has a Hamiltonian cycle $C \Leftarrow G$ has a vertex cover of size k:

Let $V^* = \{v_1, v_2, \ldots, v_k\}$ be a vertex cover (wlog $|V^*| = k$).

We construct a cycle C as follows:

- Start at a selector vertex a_1 and add the the chain of gadgets corresponding to node v_1 to C.
- If both endpoints of an edge are contained in V^*, add only the vertices of the gadget on the side of v_1. Otherwise, add all the vertices of the gadget.
- Continue to the next selector vertex and repeat this process for the next v_i and a_i.
- Finally, complete the cycle by connecting back to a_1.

In each subpath starting at selector vertex a_i, we visit all gadgets corresponding to the edges connected to v_i either fully or partially. If they are visited partially, the other half of the gadget will be visited in the subpath corresponding to the other endpoint of the gadget's edge.

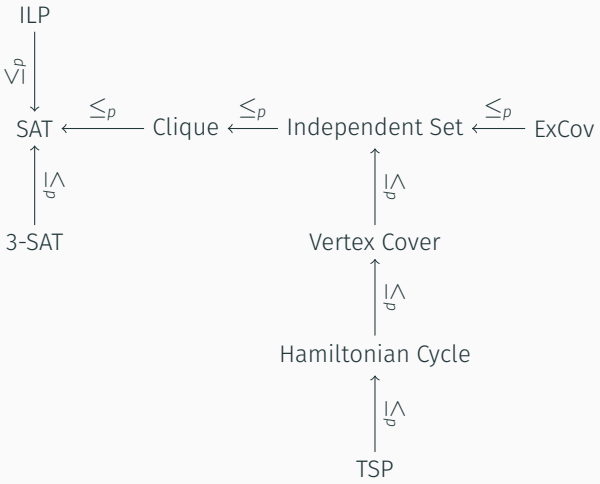
As V^* covers all edges, and each edge corresponds to a gadget, and all gadgets and selector nodes are visited, C must be a HC. ✓

Theorem

TSP is NP-complete.

Proof: already seen: in NP and TSP$\leq_p$ HC.

Reductions


ILP
$\leq_p$
SAT
$\leq_p$
Clique
$\leq_p$
Independent Set
$\leq_p$
ExCov
$\leq_p$
3-SAT
$\leq_p$
Vertex Cover
$\leq_p$
Hamiltonian Cycle
$\leq_p$
TSP

Questions?

Conclusion

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)

Next week: co-NP, strong NP-hardness and weak NP-hardness.