

NP-Complete Problems

NP-Complete Problems

There is a big class of algorithmic problems called **NP-complete** problems such that:

- they belong to **NPTIME**, i.e., they can be solved in polynomial time by a **nondeterministic** algorithm
- these problems can be solved by exponential time algorithms
- no polynomial time algorithm is known for any of these problems
- on the other hand, for any of these problems it is not proved that there cannot exist a polynomial time algorithm for the given problem
- every NP-complete problem can be polynomially reduced to any other NP-complete problem

Remark: This is not a definition of NP-complete problems. The precise definition will be described later.

Polynomial Reductions between Problems

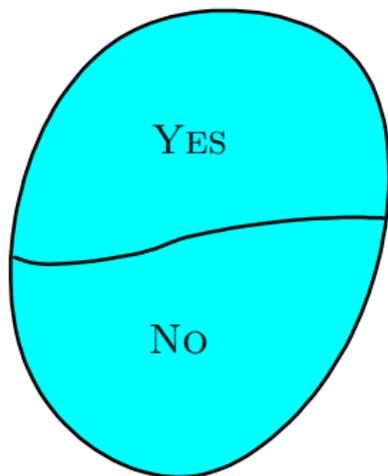
There is a **polynomial reduction** of problem P_1 to problem P_2 if there exists an algorithm Alg with a polynomial time complexity that reduces problem P_1 to problem P_2 .

Tento algoritmus Alg :

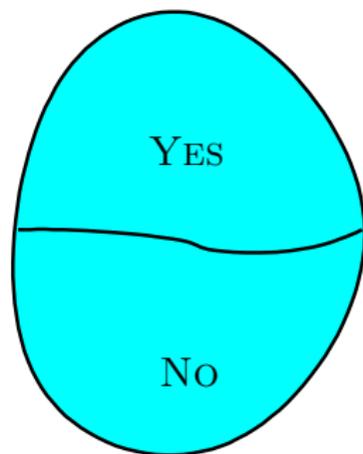
- It can get an arbitrary instance of problem P_1 as an input.
- For an instance of a problem P_1 obtained as an input (let us denote it as x) it produces an instance of a problem P_2 as an output.
- It holds i.e., the answer for the input x of problem P_1 is **YES** iff the answer for the input $Alg(x)$ of problem P_2 is **YES**.

Polynomial Reductions between Problems

Inputs of problem P_1



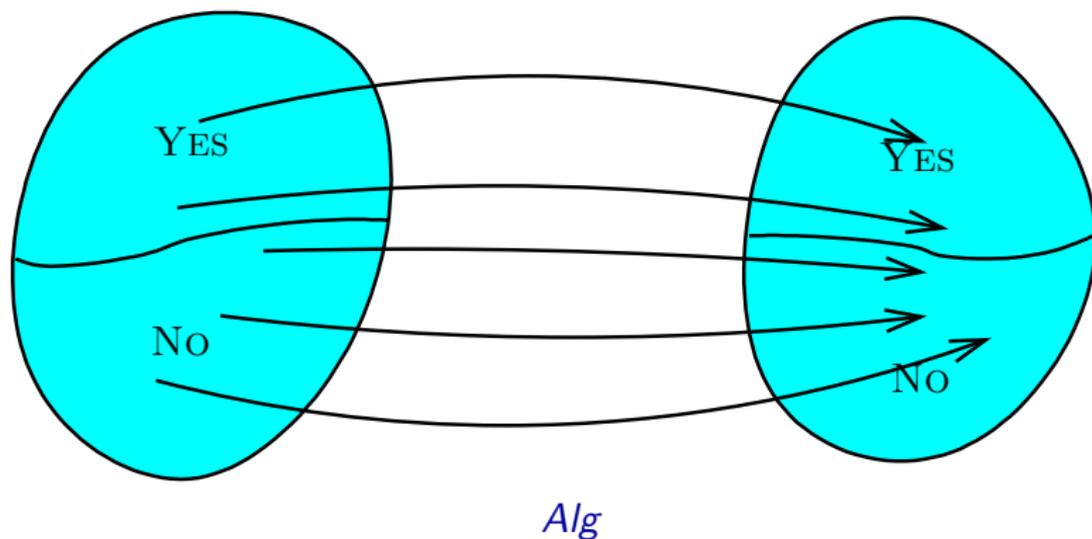
Inputs of problem P_2



Polynomial Reductions between Problems

Inputs of problem P_1

Inputs of problem P_2



Polynomial Reductions between Problems

Let us say that problem A can be reduced in polynomial time to problem B , i.e., there is a polynomial algorithm P realizing this reduction.

If problem B is in the class PTIME then problem A is also in the class PTIME .

A solution of problem A for an input x :

- Call P with input x and obtain a returned value $P(x)$.
- Call a polynomial time algorithm solving problem B with the input $P(x)$.

Write the returned value as the answer for A .

That means:

If A is not in PTIME then also B can not be in PTIME .

As an example, a polynomial time reduction from the 3-SAT problem to the independent set problem (IS) will be described.

Remark: Both 3-SAT and IS are examples of NP-complete problems.

Problem 3-SAT

3-SAT is a variant of the SAT problem where the possible inputs are restricted to formulas of a certain special form:

3-SAT

Input: Formula φ is a conjunctive normal form where every clause contains exactly 3 literals.

Question: Is φ satisfiable?

Problem 3-SAT

Recalling some notions:

- A **literal** is a formula of the form x or $\neg x$ where x is an atomic proposition.
- A **clause** is a disjunction of literals.

Examples: $x_1 \vee \neg x_2$ $\neg x_5 \vee x_8 \vee \neg x_{15} \vee \neg x_{23}$ x_6

- A formula is in a **conjunctive normal form (CNF)** if it is a conjunction of clauses.

Example: $(x_1 \vee \neg x_2) \wedge (\neg x_5 \vee x_8 \vee \neg x_{15} \vee \neg x_{23}) \wedge x_6$

So in the 3-SAT problem we require that a formula φ is in a CNF and moreover that every clause of φ contains exactly three literals.

Example:

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

Problem 3-SAT

The following formula is satisfiable:

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

It is true for example for valuation ν where

$$\nu(x_1) = 0$$

$$\nu(x_2) = 1$$

$$\nu(x_3) = 0$$

$$\nu(x_4) = 1$$

On the other hand, the following formula is not satisfiable:

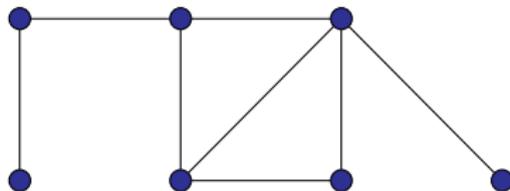
$$(x_1 \vee x_1 \vee x_1) \wedge (\neg x_1 \vee \neg x_1 \vee \neg x_1)$$

Independent Set (IS) Problem

Independent set (IS) problem

Input: An undirected graph G , a number k .

Question: Is there an independent set of size k in the graph G ?



$k = 4$

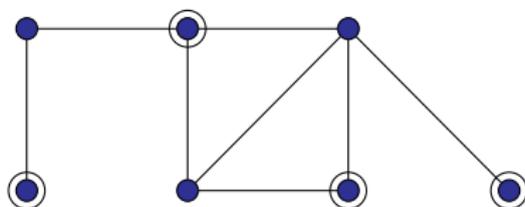
Remark: An **independent set** in a graph is a subset of nodes of the graph such that no pair of nodes from this set is connected by an edge.

Independent Set (IS) Problem

Independent set (IS) problem

Input: An undirected graph G , a number k .

Question: Is there an independent set of size k in the graph G ?

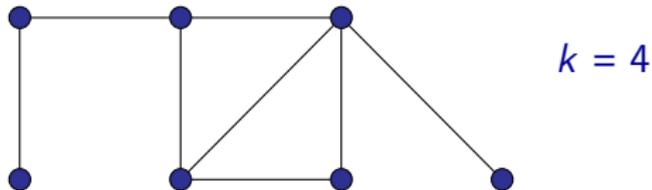


$k = 4$

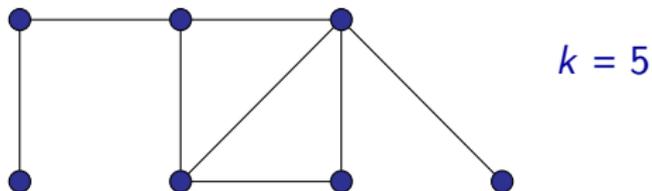
Remark: An **independent set** in a graph is a subset of nodes of the graph such that no pair of nodes from this set is connected by an edge.

Independent Set (IS) Problem

An example of an instance where the answer is **YES**:



An example of an instance where the answer is **NO**:



A Reduction from 3-SAT to IS

We describe a (polynomial-time) algorithm with the following properties:

- **Input:** An arbitrary instance of 3-SAT, i.e., a formula φ in a conjunctive normal form where every clause contains exactly three literals.
- **Output:** An instance of IS, i.e., an undirected graph G and a number k .
- Moreover, the following will be ensured for an arbitrary input (i.e., for an arbitrary formula φ in the above mentioned form):

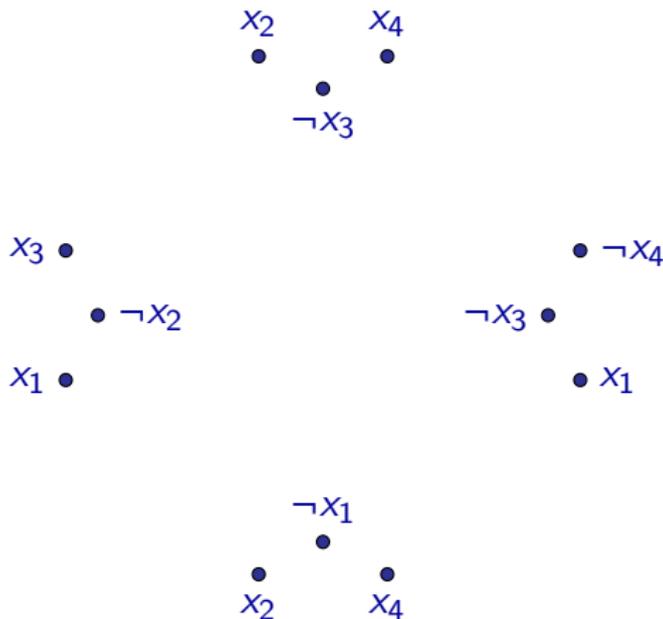
There will be an independent set of size k in graph G iff formula φ will be satisfiable.

A Reduction from 3-SAT to IS

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

A Reduction from 3-SAT to IS

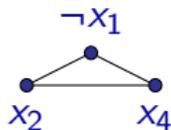
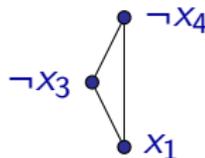
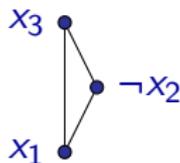
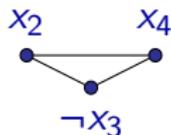
$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$



For each occurrence of a literal we add a node to the graph.

A Reduction from 3-SAT to IS

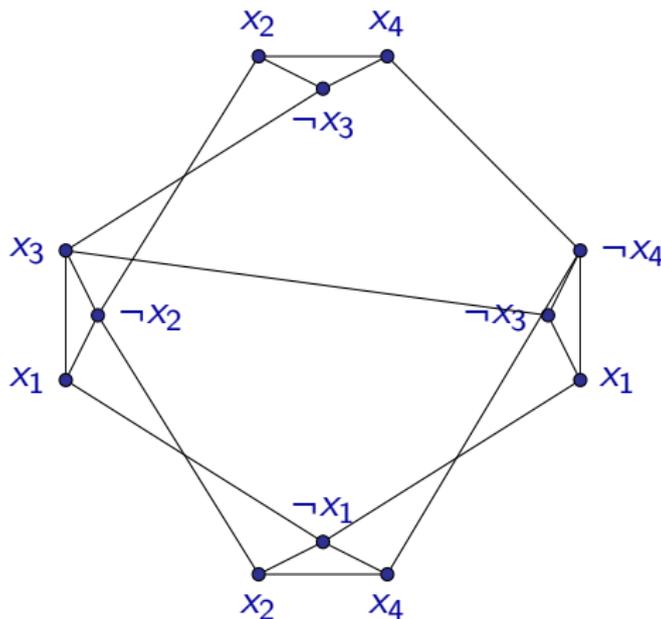
$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$



We connect with edges the nodes corresponding to occurrences of literals belonging to the same clause.

A Reduction from 3-SAT to IS

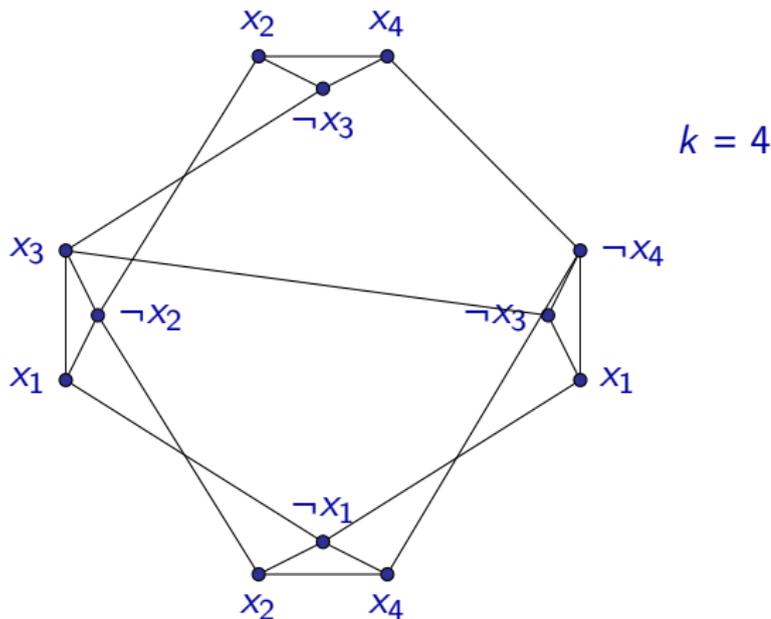
$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$



For each pair of nodes corresponding to literals x_i and $\neg x_i$ we add an edge between them.

A Reduction from 3-SAT to IS

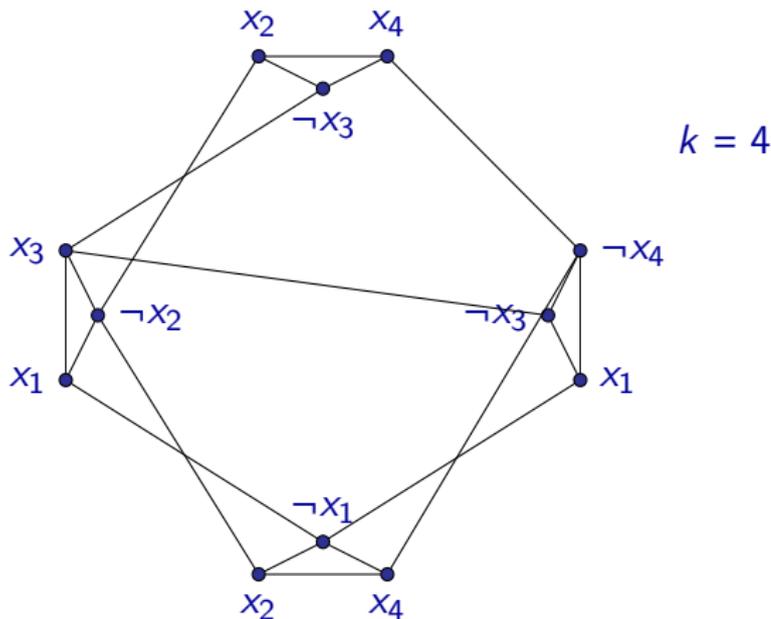
$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$



We put k to be equal to the number of clauses.

A Reduction from 3-SAT to IS

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



The constructed graph and number k are the output of the algorithm.

A Reduction from 3-SAT to IS

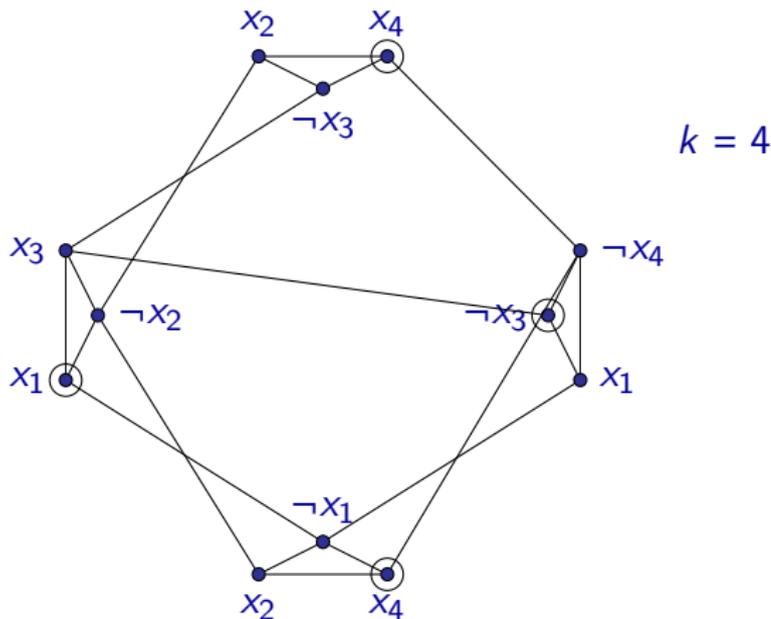
$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

$$\nu(x_1) = 1$$

$$\nu(x_2) = 1$$

$$\nu(x_3) = 0$$

$$\nu(x_4) = 1$$



We select one literal that has a value **1** in the valuation ν , and we put the corresponding node into the independent set.

A Reduction from 3-SAT to IS

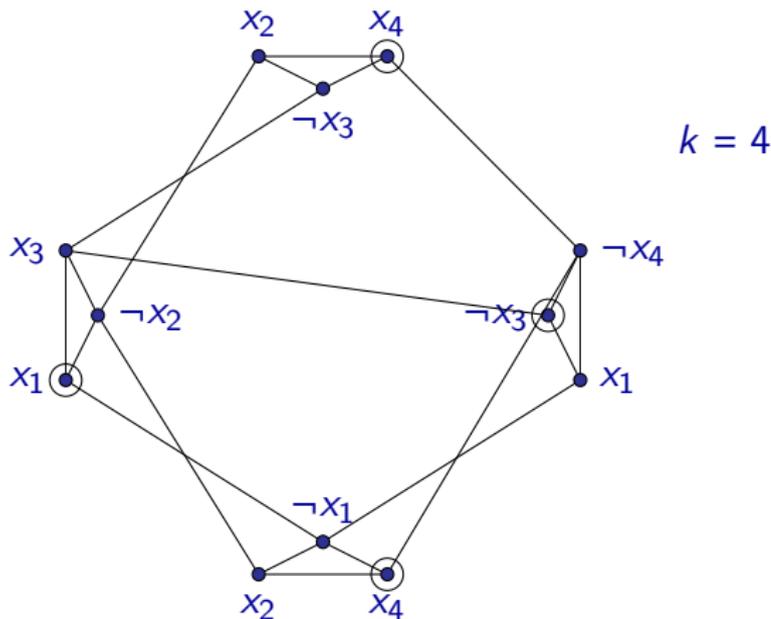
$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

$$\nu(x_1) = 1$$

$$\nu(x_2) = 1$$

$$\nu(x_3) = 0$$

$$\nu(x_4) = 1$$



We can easily verify that the selected nodes form an independent set.

A Reduction from 3-SAT to IS

The selected nodes form an independent set because:

- One node has been selected from each triple of nodes corresponding to one clause.
- Nodes denoted x_i and $\neg x_i$ could not be selected together. (Exactly one of them has the value **1** in the given valuation ν .)

A Reduction from 3-SAT to IS

On the other hand, if there is an independent set of size k in graph G , then it surely has the following properties:

- At most one node is selected from each triple of nodes corresponding to one clause.

But because there are k clauses and k nodes are selected, exactly one node must be selected from each triple.

- Nodes denoted x_i and $\neg x_i$ cannot be selected together.

We can choose a valuation according to the selected nodes, since it follows from the previous discussion that it must exist.

(Arbitrary values can be assigned to the remaining variables.)

For the given valuation, the formula φ has surely the value **1**, since in each clause there is at least one literal with value **1**.

A Reduction from 3-SAT to IS

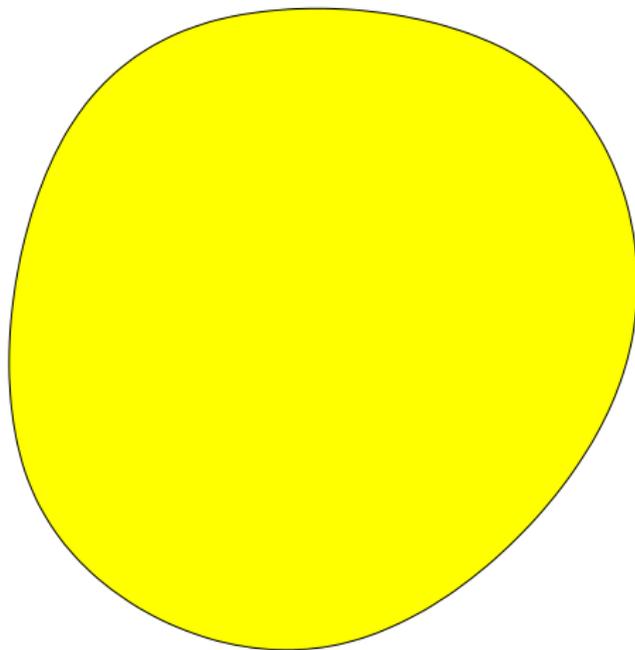
It is obvious that the running time of the described algorithm polynomial:
Graph G and number k can be constructed for a formula φ in time $\mathcal{O}(n^2)$,
where n is the size of formula φ .

We have also seen that there is an independent set of size k in the
constructed graph G iff the formula φ is satisfiable.

The described algorithm shows that 3-SAT can be reduced in polynomial
time to IS.

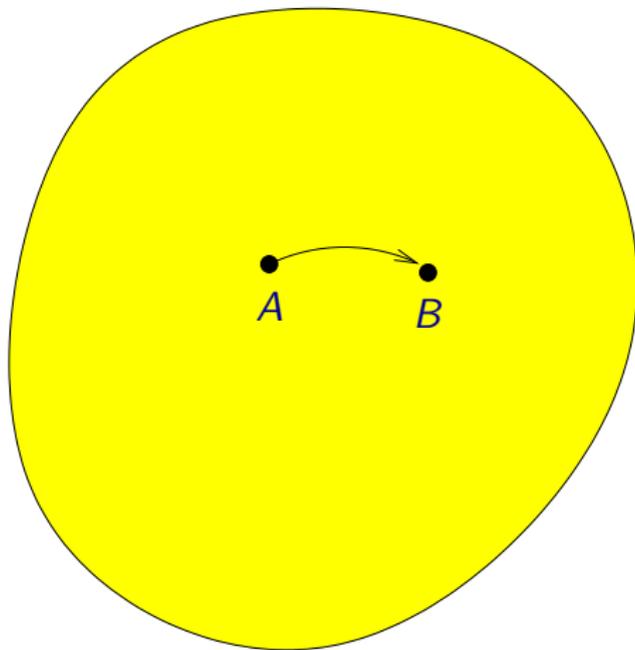
NP-Complete Problems

Let us consider a set of all decision problems.



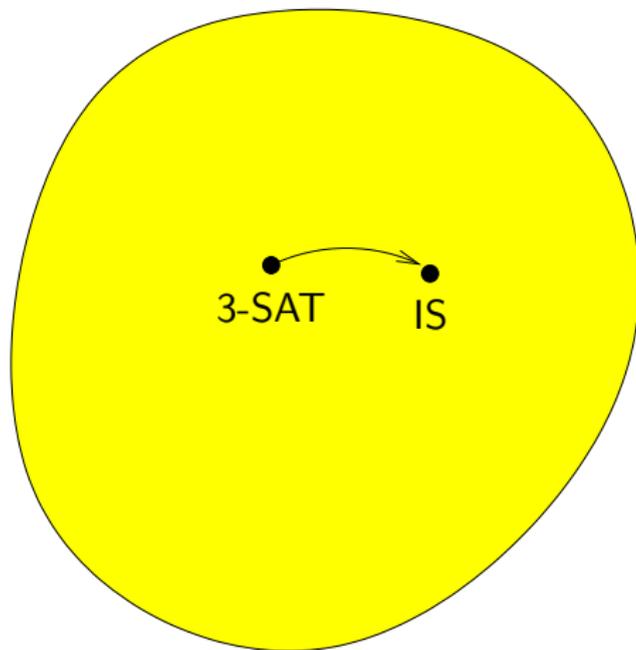
NP-Complete Problems

By an arrow we denote that a problem A can be reduced in polynomial time to a problem B .



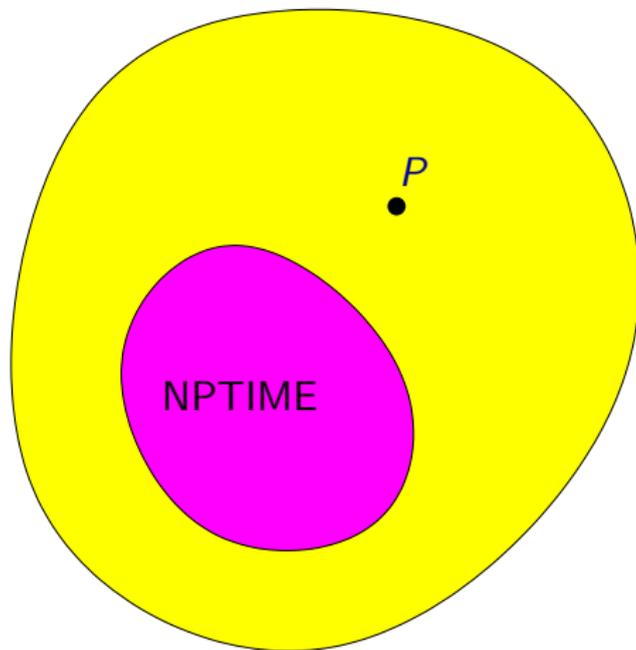
NP-Complete Problems

For example 3-SAT can be reduced in polynomial time to IS.



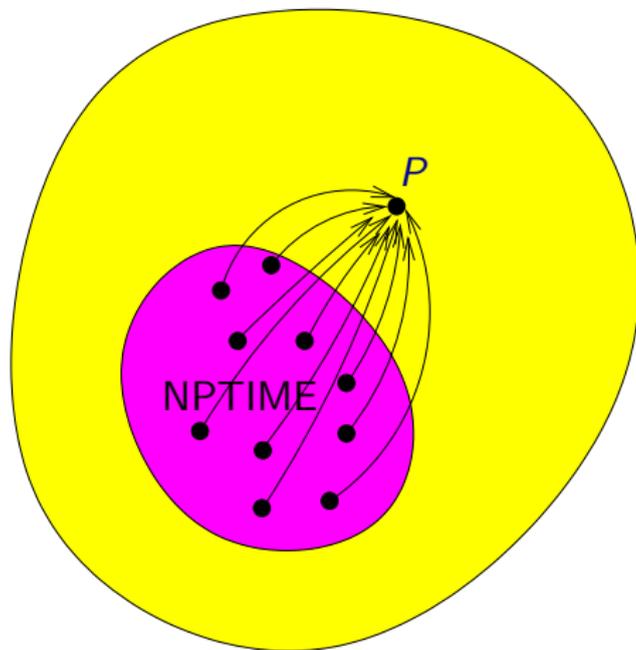
NP-Complete Problems

Let us consider now the class **NPTIME** and a problem P .



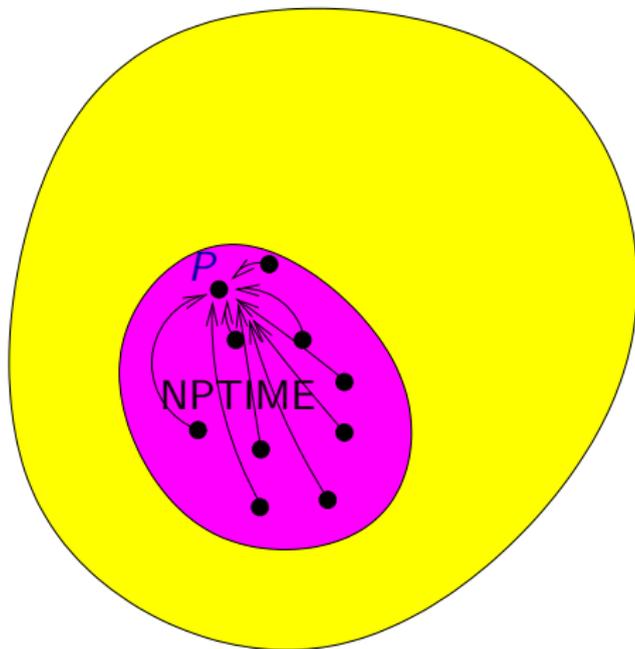
NP-Complete Problems

A problem P is **NP-hard** if every problem from **NPTIME** can be reduced in polynomial time to P .



NP-Complete Problems

A problem P is **NP-complete** if it is NP-hard and it belongs to the class NPTIME.



NP-Complete Problems

If we have found a polynomial time algorithm for some NP-hard problem P , then we would have polynomial time algorithms for all problems P' from NPTIME :

- At first we would apply an algorithm for the reduction from P' to P on an input of a problem P' .
- Then we would use a polynomial algorithm for P on the constructed instance of P and returned its result as the answer for the original instance of P' .

In such case, $\text{PTIME} = \text{NPTIME}$ would hold, since for every problem from NPTIME there would be a polynomial-time (deterministic) algorithm.

On the other hand, if there is at least one problem from **NPTIME** for which a polynomial-time algorithm does not exist, then it means that for none of **NP**-hard problems there is a polynomial-time algorithm.

It is an open question whether the first or the second possibility holds.

NP-Complete Problems

It is not difficult to see that:

If a problem A can be reduced in a polynomial time to a problem B and problem B can be reduced in a polynomial time to a problem C , then problem A can be reduced in a polynomial time to problem C .

So if we know about some problem P that it is NP-hard and that P can be reduced in a polynomial time to a problem P' , then we know that the problem P' is also NP-hard.

NP-Complete Problems

Theorem (Cook, 1971)

Problem SAT is NP-complete.

It can be shown that SAT can be reduced in a polynomial time to 3-SAT and we have seen that 3-SAT can be reduced in a polynomial time to IS.

This means that problems 3-SAT and IS are NP-hard.

It is not difficult to show that 3-SAT and IS belong to the class NPTIME.

Problems 3-SAT and IS are NP-complete.

Examples of Some NP-Complete Problems

The following previously mentioned problems are NP-complete:

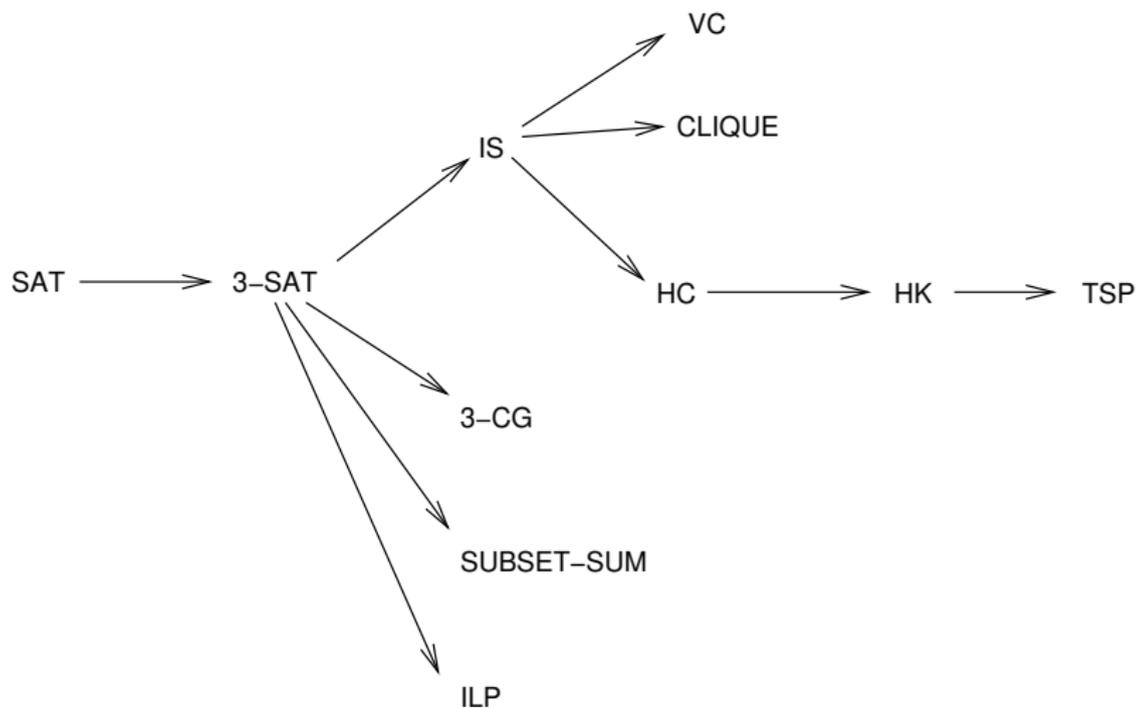
- SAT (boolean satisfiability problem)
- 3-SAT
- IS — independent set problem

On the following slides, examples of some other NP-complete problems are described:

- CG — graph coloring (remark: it is NP-complete even in the special case where we have 3 colors)
- VC — vertex cover
- CLIQUE — clique problem
- HC — Hamiltonian cycle
- HK — Hamiltonian circuit
- TSP — traveling salesman problem
- SUBSET-SUM
- ILP — integer linear programming

Examples of Some NP-Complete Problems

By a polynomial reductions from problems that are already known to be NP-complete, NP-completeness of many other problems can be shown:



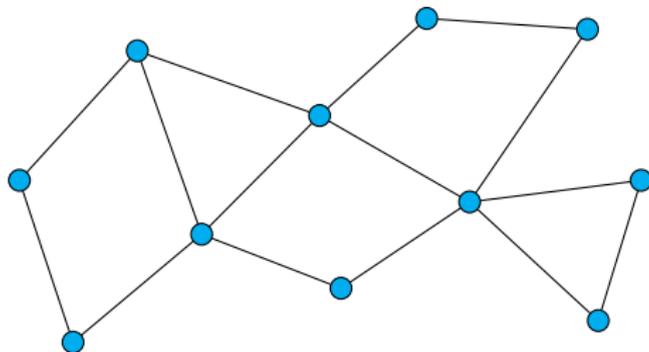
Graph Coloring

Graph coloring

Input: An undirected graph G , a natural number k .

Question: Is it possible to color nodes of the graph G using k colors in such a way that there is no pair of nodes where both nodes are colored with the same color and connected with an edge?

Example: $k = 3$



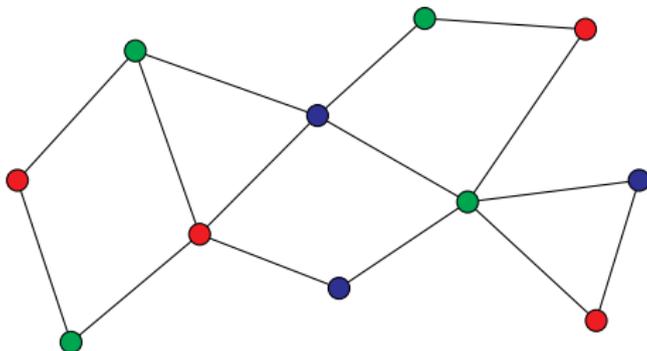
Graph Coloring

Graph coloring

Input: An undirected graph G , a natural number k .

Question: Is it possible to color nodes of the graph G using k colors in such a way that there is no pair of nodes where both nodes are colored with the same color and connected with an edge?

Example: $k = 3$



Answer: YES

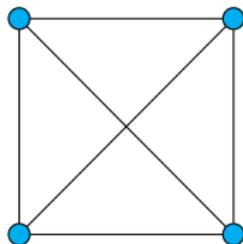
Graph Coloring

Graph coloring

Input: An undirected graph G , a natural number k .

Question: Is it possible to color nodes of the graph G using k colors in such a way that there is no pair of nodes where both nodes are colored with the same color and connected with an edge?

Example: $k = 3$



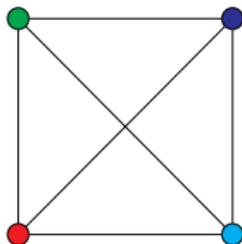
Graph Coloring

Graph coloring

Input: An undirected graph G , a natural number k .

Question: Is it possible to color nodes of the graph G using k colors in such a way that there is no pair of nodes where both nodes are colored with the same color and connected with an edge?

Example: $k = 3$



Answer: No

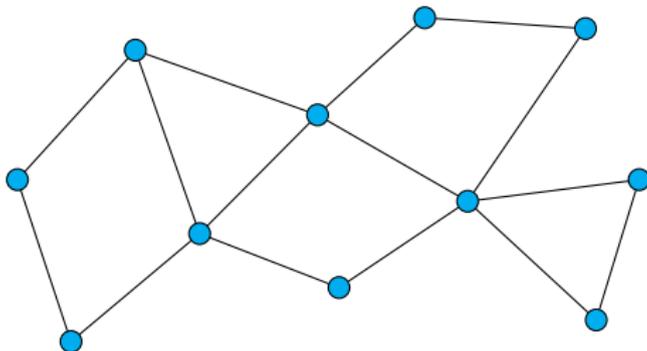
VC – Vertex Cover

VC – vertex cover

Input: An undirected graph G and a natural number k .

Question: Is there some subset of nodes of G of size k such that every edge has at least one of its nodes in this subset?

Example: $k = 6$



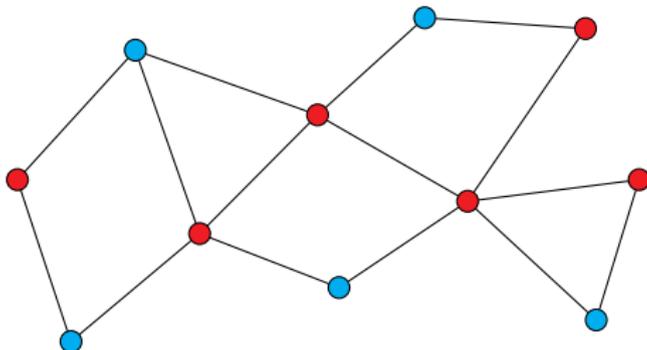
VC – Vertex Cover

VC – vertex cover

Input: An undirected graph G and a natural number k .

Question: Is there some subset of nodes of G of size k such that every edge has at least one of its nodes in this subset?

Example: $k = 6$



Answer: YES

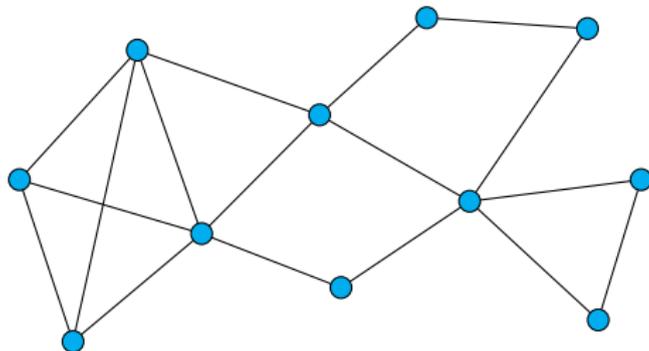
CLIQUE

CLIQUE

Input: An undirected graph G and a natural number k .

Question: Is there some subset of nodes of G of size k such that every two nodes from this subset are connected by an edge?

Example: $k = 4$



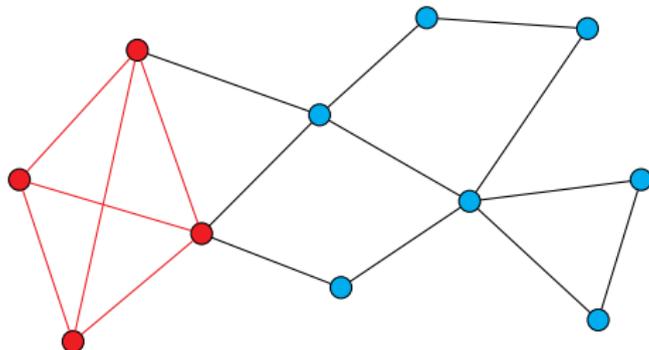
CLIQUE

CLIQUE

Input: An undirected graph G and a natural number k .

Question: Is there some subset of nodes of G of size k such that every two nodes from this subset are connected by an edge?

Example: $k = 4$



Answer: YES

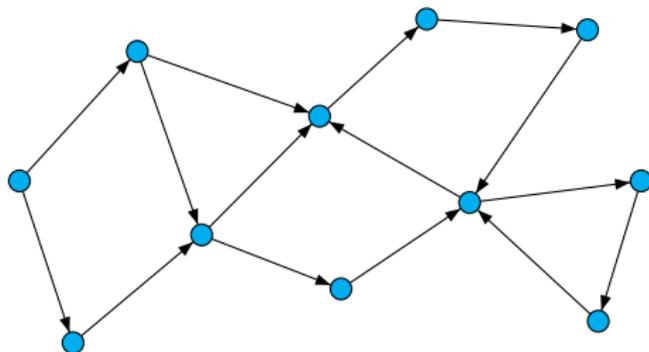
Hamiltonian Cycle

HC – Hamiltonian cycle

Input: A directed graph G .

Question: Is there a Hamiltonian cycle in G (i.e., a directed cycle going through each node exactly once)?

Example:



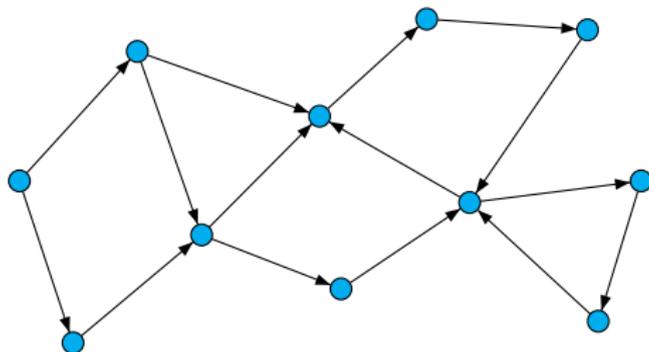
Hamiltonian Cycle

HC – Hamiltonian cycle

Input: A directed graph G .

Question: Is there a Hamiltonian cycle in G (i.e., a directed cycle going through each node exactly once)?

Example:



Answer: No

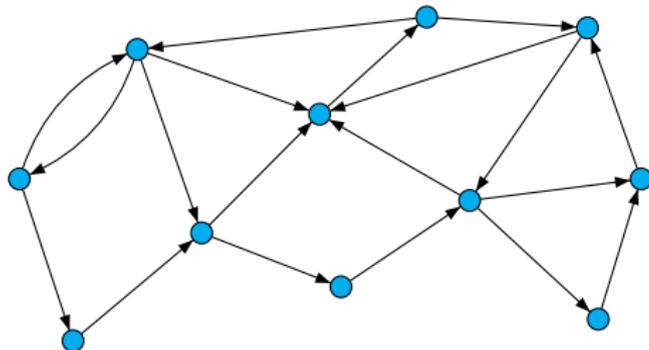
Hamiltonian Cycle

HC – Hamiltonian cycle

Input: A directed graph G .

Question: Is there a Hamiltonian cycle in G (i.e., a directed cycle going through each node exactly once)?

Example:



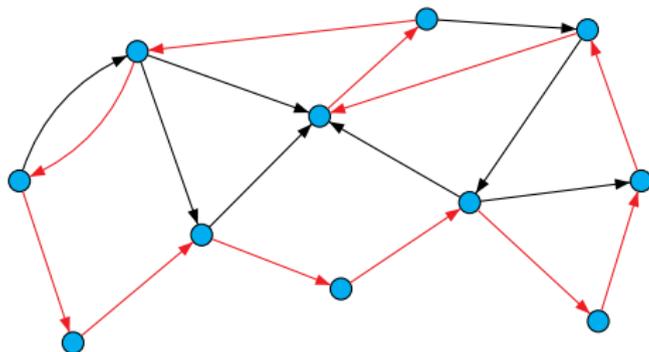
Hamiltonian Cycle

HC – Hamiltonian cycle

Input: A directed graph G .

Question: Is there a Hamiltonian cycle in G (i.e., a directed cycle going through each node exactly once)?

Example:



Answer: YES

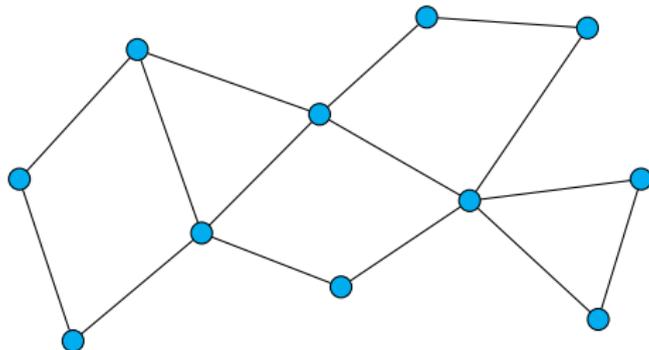
Hamiltonian Circuit

HK – Hamiltonian circuit

Input: An undirected graph G .

Question: Is there a Hamiltonian circuit in G (i.e., an undirected cycle going through each node exactly once)?

Example:



Answer: No

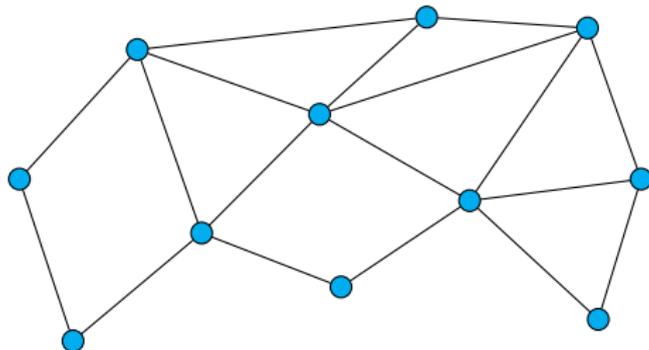
Hamiltonian Circuit

HK – Hamiltonian circuit

Input: An undirected graph G .

Question: Is there a Hamiltonian circuit in G (i.e., an undirected cycle going through each node exactly once)?

Example:



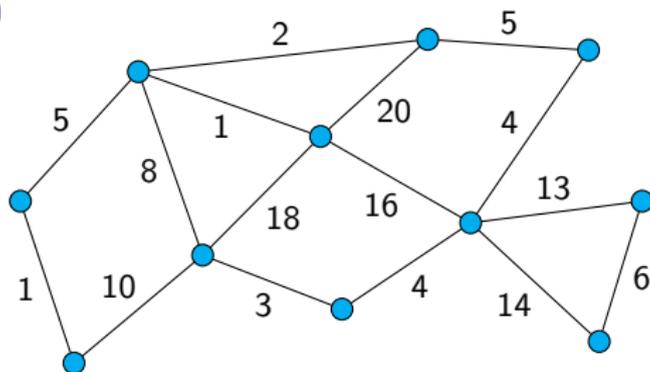
Traveling Salesman Problem

TSP - traveling salesman problem

Input: An undirected graph G with edges labelled with natural numbers and a number k .

Question: Is there a closed tour going through all nodes of the graph G such that the sum of labels of edges on this tour is at most k ?

Example: $k = 70$



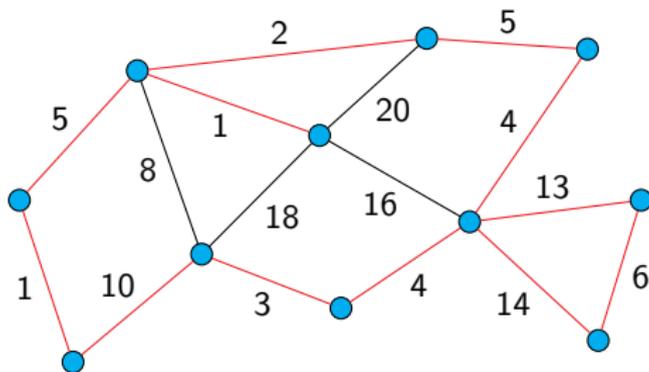
Traveling Salesman Problem

TSP - traveling salesman problem

Input: An undirected graph G with edges labelled with natural numbers and a number k .

Question: Is there a closed tour going through all nodes of the graph G such that the sum of labels of edges on this tour is at most k ?

Example: $k = 70$



Answer: YES, since there is a tour with the sum 69.

SUBSET-SUM

Problem SUBSET-SUM

Input: A sequence a_1, a_2, \dots, a_n of natural numbers and a natural number s .

Question: Is there a set $I \subseteq \{1, 2, \dots, n\}$ such that $\sum_{i \in I} a_i = s$?

In other words, the question is whether it is possible to select a subset with sum s of a given (multi)set of numbers.

Example: For the input consisting of numbers $3, 5, 2, 3, 7$ and number $s = 15$ the answer is **YES**, since $3 + 5 + 7 = 15$.

For the input consisting of numbers $3, 5, 2, 3, 7$ and number $s = 16$ the answer is **NO**, since no subset of these numbers has sum 16 .

Remark:

The order of numbers a_1, a_2, \dots, a_n in an input is not important.

Note that this is not exactly the same as if we have formulated the problem so that the input is a set $\{a_1, a_2, \dots, a_n\}$ and a number s — numbers cannot occur multiple times in a set but they can in a sequence.

Problem SUBSET-SUM is a special case of a **knapsack problem**:

Knapsack problem

Input: Sequence of pairs of natural numbers

$(a_1, b_1), (a_2, b_2), \dots, (a_n, b_n)$ and two natural numbers s and t .

Question: Is there a set $I \subseteq \{1, 2, \dots, n\}$ such that $\sum_{i \in I} a_i \leq s$ and $\sum_{i \in I} b_i \geq t$?

SUBSET-SUM

Informally, the knapsack problem can be formulated as follows:

We have n objects, where the i -th object weights a_i grams and its price is b_i dollars.

The question is whether there is a subset of these objects with total weight at most s grams (s is the capacity of the knapsack) and with total price at least t dollars.

Remark:

Here we have formulated this problem as a decision problem.

This problem is usually formulated as an optimization problem where the aim is to find such a set $I \subseteq \{1, 2, \dots, n\}$, where the value $\sum_{i \in I} b_i$ is maximal and where the condition $\sum_{i \in I} a_i \leq s$ is satisfied, i.e., where the capacity of the knapsack is not exceeded.

That SUBSET-SUM is a special case of the Knapsack problem can be seen from the following simple construction:

Let us say that $a_1, a_2, \dots, a_n, s_1$ is an instance of SUBSET-SUM.

It is obvious that for the instance of the knapsack problem where we have the sequence $(a_1, a_1), (a_2, a_2), \dots, (a_n, a_n), s = s_1$ and $t = s_1$, the answer is the same as for the original instance of SUBSET-SUM.

SUBSET-SUM

If we want to study the complexity of problems such as SUBSET-SUM or the knapsack problem, we must clarify what we consider as the size of an instance.

Probably the most natural it is to define the size of an instance as the total number of bits needed for its representation.

We must specify how natural numbers in the input are represented – if in binary (resp. in some other numeral system with a base at least 2 (e.g., decimal or hexadecimal) or in unary.

- If we consider the total number of bits when numbers are written in **binary** as the size of an input, no polynomial time algorithm is known for SUBSET-SUM.
- If we consider the total number of bits when numbers are written in **unary** as the size of an input, SUBSET-SUM can be solved by an algorithm whose time complexity is polynomial.

Problem ILP (integer linear programming)

Input: An integer matrix A and an integer vector b .

Question: Is there an integer vector x such that $Ax \leq b$?

An example of an instance of the problem:

$$A = \begin{pmatrix} 3 & -2 & 5 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix} \quad b = \begin{pmatrix} 8 \\ -3 \\ 5 \end{pmatrix}$$

So the question is if the following system of inequations has some integer solution:

$$\begin{aligned} 3x_1 - 2x_2 + 5x_3 &\leq 8 \\ x_1 + x_3 &\leq -3 \\ 2x_1 + x_2 &\leq 5 \end{aligned}$$

ILP – Integer Linear Programming

One of solutions of the system

$$\begin{aligned}3x_1 - 2x_2 + 5x_3 &\leq 8 \\x_1 + x_3 &\leq -3 \\2x_1 + x_2 &\leq 5\end{aligned}$$

is for example $x_1 = -4$, $x_2 = 1$, $x_3 = 1$, i.e.,

$$x = \begin{pmatrix} -4 \\ 1 \\ 1 \end{pmatrix}$$

because

$$\begin{aligned}3 \cdot (-4) - 2 \cdot 1 + 5 \cdot 1 &= -9 \leq 8 \\-4 + 1 &= -3 \leq -3 \\2 \cdot (-4) + 1 &= -7 \leq 5\end{aligned}$$

So the answer for this instance is **YES**.

Remark: A similar problem where the question for a given system of linear inequations is whether it has a solution in the set of **real** numbers, can be solved in a polynomial time.

A well-known book concerning NP-complete problems is

Michael R. Garey, David S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

This book also contains a catalog of NP-complete problems:

- It contains more than 300 problems.
- For each problem, its definition is described, references to literature are provided, and some notes concerning different variants of the problem, which of these variants are solvable in polynomial time, and which are NP-complete, etc.

- The catalog is divided into the following sections according to the areas, to which these problems belong:
 - Graph Theory
 - Network Design
 - Sets and Partitions
 - Storage and Retrieval
 - Sequencing and Scheduling
 - Mathematical Programming
 - Algebra and Number Theory
 - Games and Puzzles
 - Logic
 - Automata and Language Theory
 - Program Optimization
 - Miscellaneous
 - Open Problems