Complexity I

Martin Hoefer

(based on material by Walter Unger)

- Some problems may not be solved in polynomial time.

Example (1): Rubik's Dodecahedron

Challenge

Solve the dodecahedron.

This problem can be solved by computer: Try all possible sequences of moves.

But:

- Tons of possibilities
- Huge computation time

Example (2): Rush Hour

Challenge

Get the red car out of the traffic jam.

This problem can be solved by computer: Try all possible sequences of moves.

But:

- Tons of possibilities
- Huge computation time

Example (3): Traveling Salesman

Challenge

Find a short round-trip through the largest German cities.

The depicted route is the shortest one (among 43.589.145.600 routes).

The problem can be solved by checking all possible cases (slow and in-efficient). Nobody knows a fast algorithm for this problem ...

Easy versus Hard Problems

- Some algorithmic problems are easy to solve, within micro-seconds, even for large inputs (for instance: minimum spanning tree; shortest path; network flow)
- Other algorithmic problems are hard to solve, and take hours or days or weeks of computation time, even for small inputs

Central question

How do we tell the hard problems from the easy ones?

Basic Concepts (1)

Discrete problem:

- Optimization problem (with goal min/max)
- Decision problem (with answer YES/NO)

Example: Optimization problem

Instance: a graph $G = (V, E)$
Goal: find a clique of maximum size in G

Example: Decision problem

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

Basic Concepts (2)

Instance:

- specification of problem data

Example: Instance of decision version of clique

- $V = \{1, 2, 3, 4, 5\}$;
- $E = \{[1, 2],\ [1, 3],\ [4, 5],\ [2, 3],\ [3, 5]\}$;
- $k = 3$

Basic Concepts (3)

Problem size:

- length (number of symbols) of reasonable encoding of instance

Example

- Graph: adjacency list; adjacency matrix
- Set: list of elements; bit vector
- Number: decimal; binary; hex; unary

We do not really care whether
an n-vertex graph is encoded with $4n^2 + 3n$ or with $7n^2 + 2$ symbols.

Recall: big-Oh notation; big-Omega; big-Theta

$4n^2 + 3n \in \Theta(n^2)$ and $7n^2 + 2 \in \Theta(n^2)$

$4n^2 + 3n \in O(n^2)$; $4n^2 + 3n \in O(n^3)$; $7n^2 + 2 \in \Omega(n \log n)$

Basic Concepts (4)

Algorithm:

- an unambiguous recipe for solving a discrete problem

Our definition in this course: **C++ program** (Church-Turing thesis)

Time complexity of an algorithm:

- number of elementary steps made by C++ program

The time complexity is measured as a function of the instance size:

- $T_A(I)$ = number of steps that algorithm A makes on instance I
- $T(n)$ = maximum number of steps that algorithm A makes on any instance I of size $O(n)$

Polynomial versus Exponential (1)

Polynomial growth rate:

- $O(\text{poly}(n))$ for some polynomial poly

Examples: $O(n)$; $O(n \log n)$; $O(n^3)$; $O(n^{100})$

Exponential growth rate:

- everything that grows faster than polynomial

Examples: $2^{\sqrt{n}}$; 2^n; 3^n; $n!$; 2^{2^n}; n^n

Intuition:

Polynomial = desirable, good, harmless, fast, short, small

Exponential = undesirable, bad, evil, slow, wasteful, horrible

Polynomial versus Exponential (2)

Question

What's the number of subsets of an n-element set?
Does this number grow polynomially with n?
Does this number grow exponentially with n?

Polynomial versus Exponential (2)

Question

What's the number of subsets of an n-element set?
Does this number grow polynomially with n?
Does this number grow exponentially with n?

2^n

Polynomial versus Exponential (2)

Question

What's the number of subsets of an n-element set?
Does this number grow polynomially with n?
Does this number grow exponentially with n?

2^n

Question

What's the number of permutations of n elements?
Does this number grow polynomially with n?
Does this number grow exponentially with n?

Polynomial versus Exponential (2)

Question

What's the number of subsets of an n-element set?
Does this number grow polynomially with n?
Does this number grow exponentially with n?

2^n

Question

What's the number of permutations of n elements?
Does this number grow polynomially with n?
Does this number grow exponentially with n?

Stirling's formula: $n! \; = \; \sqrt{2\pi n} \cdot (n/e)^n$
Ergo: $n! \in O(n^n)$

Polynomial versus Exponential (3)

Question

What's the number of 2-element subsets of an n-element set?

Polynomial versus Exponential (3)

Question

What's the number of 2-element subsets of an n-element set?

$$\binom{n}{2} \;=\; \frac{1}{2}n(n-1) \;\in\; O(n^2)$$

Polynomial versus Exponential (3)

Question

What's the number of 2-element subsets of an n-element set?

$$\binom{n}{2} = \frac{1}{2}n(n-1) \in O(n^2)$$

Question

What's the number of 3-element subsets of an n-element set?

Polynomial versus Exponential (3)

Question

What's the number of 2-element subsets of an n-element set?

$$\binom{n}{2} = \frac{1}{2}n(n-1) \in O(n^2)$$

Question

What's the number of 3-element subsets of an n-element set?

$$\binom{n}{3} = \frac{1}{6}n(n-1)(n-2) \in O(n^3)$$

Polynomial versus Exponential (3)

Question

What's the number of 2-element subsets of an n-element set?

$$\binom{n}{2} = \frac{1}{2}n(n-1) \in O(n^2)$$

Question

What's the number of 3-element subsets of an n-element set?

$$\binom{n}{3} = \frac{1}{6}n(n-1)(n-2) \in O(n^3)$$

Question

What's the number of k-element subsets of an n-element set?

Polynomial versus Exponential (3)

Question

What's the number of 2-element subsets of an n-element set?

$$\binom{n}{2} = \frac{1}{2}n(n-1) \in O(n^2)$$

Question

What's the number of 3-element subsets of an n-element set?

$$\binom{n}{3} = \frac{1}{6}n(n-1)(n-2) \in O(n^3)$$

Question

What's the number of k-element subsets of an n-element set?

The binomial coefficient $\binom{n}{k} = \frac{n(n-1)(n-2)\cdots(n-k+1)}{k!} \in O(n^k)$

Complexity Class P

Definition

A decision problem X lies in the complexity class P,
if X is solved by a C++ program with polynomial time complexity

- **P** stands for **P**olynomial Time

Example

The following problems are in P:

- Computing the median of a list of integers
- Computing the greatest common divisor of two integers
- Checking whether a given graph is planar
- Computing a minimum spanning tree for an edge-weighted graph
- Solving a linear program
- Testing whether a given integer is a prime

Complexity Class NP

Definition

A decision problem X lies in the complexity class NP,
if the YES-instances of X possess certificates of polynomial length
that can be verified in polynomial time

- Certificate: short piece of text; short proof; supporting evidence
- **NP** stands for **N**on-deterministic **P**olynomial

Complexity Class NP

Definition

A decision problem X lies in the complexity class NP,
if the YES-instances of X possess certificates of polynomial length
that can be verified in polynomial time

- Certificate: short piece of text; short proof; supporting evidence
- **NP** stands for **N**on-deterministic **P**olynomial

Example

A certificate for the decision version of clique:
Subset $C \subseteq V$ of size k that induces a clique

Exercise: Satisfiability

Satisfiability (SAT)

Instance:
A logical formula Φ in CNF over logical variable set $X = \{x_1, \ldots, x_n\}$

Question: Does there exist a truth setting for X that satisfies Φ?

Examples

$$\varphi_1 = (x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$$
$$\varphi_1 = (x + y + z)\,(\overline{x} + \overline{y} + \overline{z})$$

$$\varphi_2 = (x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$
$$\varphi_2 = (x + y)\,(\overline{x} + y)\,(x + \overline{y})\,(\overline{x} + \overline{y})$$

Question

What's a good NP-certificate for SAT?

Exercise: Independent Set / Vertex Cover

Problem: Independent Set (INDEP-SET)

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

Question: Does G contain an independent set of size $\geq k$?

Problem: Vertex Cover (VC)

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

Question: Does G contain a vertex cover of size $\leq k$?

- Independent set $S \subseteq V$: does not span any edges
- Vertex cover $S \subseteq V$: touches all edges in the graph

Question

What's a good NP-certificate for INDEP-SET / VC?

Exercise: Hamiltonian Cycle / TSP

Hamiltonian cycle (Ham-Cycle)

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?

Question

What's a good NP-certificate for Ham-Cycle?
What's a good NP-certificate for TSP?

Exercise: Exact Cover

Exact cover (Ex-Cover)

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?

- Partition: every element of X lies in exactly one part

Question

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

Exercise: Subset-Sum

Subset-Sum

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

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

Question

What's a good NP-certificate for Subset-Sum?

What have we seen so far?

Complexity class P

The class P contains all decision problems
that can be solved efficiently on a computer.

Intuitively: P contains the problems that we understand well and that we can settle within a reasonable amount of computation time

What have we seen so far?

Complexity class P

The class *P* contains all decision problems
that can be solved efficiently on a computer.

Intuitively: *P* contains the problems that we understand well and that we can settle within a reasonable amount of computation time

Complexity class NP

The class *NP* contains all decision problems
for which there **exists** a short solution,
and whose short solution can efficiently be verified
(under the assumption that this short solution is shown to us)

Intuitively: *NP* contains more or less all natural problems that ask us to specify a concrete solution

The big open question of computer science

P=NP ?

If the solution to some problem is easy to check,
does this mean that the solution is also easy to detect?

Consequences

In case P=NP:

- Many difficult problems from economics and industry can be solved quickly
- Perfect time tables, production plans, transportation plans, etc
- Mathematics reaches a new level: If a theorem allows a short proof, then we are also able to detect this proof
- Modern cryptography collapses

Consequences

In case P=NP:

- Many difficult problems from economics and industry can be solved quickly
- Perfect time tables, production plans, transportation plans, etc
- Mathematics reaches a new level: If a theorem allows a short proof, then we are also able to detect this proof
- Modern cryptography collapses

In case P$\neq$NP:

- Difficult problems from economics and industry can only be attacked with lots of computation time and expert knowledge
- We should not expect perfect solutions for hard problems with lots of data
- Mathematics and cryptography will not change much

One Million Dollar

In the year 2000, the **Clay Mathematics Institute (CMI)** has offered one million dollar prize money for the solution of each of the following seven problems:

- P versus NP problem
- Hodge conjecture
- Poincaré conjecture ✓ (Grigori Perelman, 2006)
- Riemann hypothesis
- Yang-Mills existence and mass gap
- Navier-Stokes existence and smoothness
- Birch and Swinnerton-Dyer conjecture

Finding solution vs Deciding existence of solution

Some arbitrary decision problem in NP

Instance: A discrete object X.

Question: Does this object X possess a solution Y?

Dilemma:

- The decision problem only formulates the question, **whether** such a solution Y **does exist**
- But in real life we would also like to determine the solution object Y precisely, and to work with it

Way out:

- A fast algorithm for the decision problem often yields (through repeated applications) a fast algorithm for the computation of an explicit solution object

Example: SAT (1)

Problem: Satisfiability (SAT)

Instance: Formula Φ in CNF over $X = \{x_1, \ldots, x_n\}$

Question: Does there exist a truth setting for X that satisfies Φ?

- Suppose that in φ some variable is fixed as $x := 1$.
  Then all clauses with the literal x are satisfied by this,
  and in all clauses with the literal $\bar{x}$ this literal simply disappears.
- We derive a shorter CNF-formula $\varphi[x = 1]$.
- In an analogous way $\varphi[x = 0]$ results by fixing $x := 0$.

Example

For $\varphi = (x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (\neg y \vee z) \wedge (u \vee z)$

we have $\varphi[y = 1] = (\neg x \vee \neg z) \wedge (z) \wedge (u \vee z)$

and $\varphi[z = 0] = (x \vee y) \wedge (\neg y) \wedge (u)$

Example: SAT (2)

We consider SAT instances with n variables and m clauses.

Theorem

Suppose the algorithm A decides SAT instances in $T(n, m)$ time. Then there exists an algorithm B, that for satisfiable SAT instances
constructs in $n \cdot T(n, m)$ time a satisfying truth setting.

Example: SAT (2)

We consider SAT instances with n variables and m clauses.

Theorem

Suppose the algorithm A decides SAT instances in $T(n, m)$ time. Then there exists an algorithm B, that for satisfiable SAT instances
constructs in $n \cdot T(n, m)$ time a satisfying truth setting.

Proof:

- We fix step by step the truth values of $x_1, x_2, \dots, x_n$.
- FOR $i = 1, 2, \dots, n$ DO
  If $\varphi[x_i = 1]$ is satisfiable, then set $x_i := 1$ and $\varphi := \varphi[x_i = 1]$
  Else set $x_i := 0$ and $\varphi := \varphi[x_i = 0]$
- At the end, the fixed truth values $x_1, x_2, \dots, x_n$ yield a satisfying truth setting for φ

Example: CLIQUE

Problem: CLIQUE

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

Question: Does G contain a clique on $\geq k$ vertices?

- If we remove from G a vertex v and all edges that are incident to v, we get a smaller graph $G - v$
- If $G - v$ contains a k-clique, vertex v is irrelevant
- If $G - v$ does not contain a k-clique, then the neighborhood $N[v]$ of vertex v in $G - v$ contains a $(k - 1)$-clique

Example: CLIQUE

Problem: CLIQUE

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

Question: Does G contain a clique on $\geq k$ vertices?

- If we remove from G a vertex v and all edges that are incident to v, we get a smaller graph $G - v$
- If $G - v$ contains a k-clique, vertex v is irrelevant
- If $G - v$ does not contain a k-clique, then the neighborhood $N[v]$ of vertex v in $G - v$ contains a $(k - 1)$-clique

Theorem

Suppose an algorithm A decides the CLIQUE problem in $T(n)$ time.
Then there exists an algorithm B, that for YES-instances constructs in $n \cdot T(n)$ time a k-clique.

Example: Hamiltonian Cycle

Problem: Hamiltonian cycle (Ham-Cycle)

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

Question: Does G contain a Hamiltonian cycle?

- If we remove from G some edge e, we get a smaller graph $G - e$
- If $G - e$ has a Hamiltonian cycle, edge e is irrelevant
- If $G - e$ does not have a Hamiltonian cycle, e is not irrelevant

Theorem

Suppose an algorithm A decides the Ham-Cycle problem in $T(n)$ time. Then there exists an algorithm B, that for YES-instances constructs in $|E| \cdot T(n)$ time a Hamiltonian cycle.

Optimization Problems

Definition: Optimization problem

The input of an optimization problem specifies (usually: implicitly) a set $\mathcal{F}$ of **feasible solutions** together with an **objective function** $f : \mathcal{F} \to \mathbb{N}$ (that measures costs, weights, profits).

The goal is to compute an optimal solution in $\mathcal{F}$.
A **minimization problem** aims at minimizing costs, and
a **maximization problem** aims at maximizing profits.

Optimization Problems

Definition: Optimization problem

The input of an optimization problem specifies (usually: implicitly) a set $\mathcal{F}$ of **feasible solutions** together with an **objective function** $f : \mathcal{F} \to \mathbb{N}$ (that measures costs, weights, profits).

The goal is to compute an optimal solution in $\mathcal{F}$.
A **minimization problem** aims at minimizing costs, and
a **maximization problem** aims at maximizing profits.

Dilemma:

- The classes P and NP only consist of **decision problems**
- But: Many real-world problems are **optimization problems**

Way out:

- We re-formulate the optimization problem into a "very similar" decision problem

Example: Traveling Salesman (1)

- An instance of the Traveling Salesman Problem consists of cities $1, \dots, n$ together with all distances $d(i,j)$ for $1 \le i \neq j \le n$
- The goal is to find a shortest round-trip (Hamilton cycle; tour) through all the cities

Optimization version of TSP

Instance: Integers $d(i,j)$ for $1 \le i \neq j \le n$

Feasible solution: Permutation π of $1, \dots, n$

Goal: Minimize $d(\pi) := \sum_{i=1}^{n-1} d(\pi(i), \pi(i+1)) + d(\pi(n), \pi(1))$

Example: Traveling Salesman (1)

- An instance of the Traveling Salesman Problem consists of cities $1, \ldots, n$ together with all distances $d(i,j)$ for $1 \leq i \neq j \leq n$
- The goal is to find a shortest round-trip (Hamilton cycle; tour) through all the cities

Optimization version of TSP

Instance: Integers $d(i,j)$ for $1 \leq i \neq j \leq n$

Feasible solution: Permutation π of $1, \ldots, n$

Goal: Minimize $d(\pi) := \sum_{i=1}^{n-1} d(\pi(i), \pi(i+1)) + d(\pi(n), \pi(1))$

Decision version of TSP

- The instance additionally contains a bound B
- Question: Does there exist a feasible solution of length $d(\pi) \leq B$?

Optimization versus Decision

For an optimization problem with a set $\mathcal{F}$ of feasible solutions and a weight function $f : \mathcal{F} \to \mathbb{N}$ we define the corresponding decision problem:

Instance: As in the optimization problem, plus a bound $B \in \mathbb{N}$

Question: Does there exist a feasible solution $x \in \mathcal{F}$
with $f(x) \geq B$ (for maximization problems) respectively
with $f(x) \leq B$ (for minimization problems)?

Optimization versus Decision

For an optimization problem with a set $\mathcal{F}$ of feasible solutions and a weight function $f : \mathcal{F} \to \mathbb{N}$ we define the corresponding decision problem:

Instance: As in the optimization problem, plus a bound $B \in \mathbb{N}$

Question: Does there exist a feasible solution $x \in \mathcal{F}$
with $f(x) \geq B$ (for maximization problems) respectively
with $f(x) \leq B$ (for minimization problems)?

- With the help of an algorithm for the optimization problem, we can easily solve the corresponding decision problem. **(How?)**

Optimization versus Decision

For an optimization problem with a set $\mathcal{F}$ of feasible solutions and a weight function $f : \mathcal{F} \to \mathbb{N}$ we define the corresponding decision problem:

Instance: As in the optimization problem, plus a bound $B \in \mathbb{N}$

Question: Does there exist a feasible solution $x \in \mathcal{F}$
with $f(x) \geq B$ (for maximization problems) respectively
with $f(x) \leq B$ (for minimization problems)?

- With the help of an algorithm for the optimization problem, we can easily solve the corresponding decision problem. **(How?)**
- With the help of an algorithm for the decision problem, we can **often** solve the corresponding optimization problem.
- We will illustrate this for the Travelling Salesman Problem.

Example: Traveling Salesman (2)

Instance: Integers $d(i,j)$ for $1 \leq i \neq j \leq n$ and B
Feasible: Permutation π of $1,\ldots,n$

Optimization: Find minimal tour length $d(\pi)$
Decision: Does there exist a permutation π with $d(\pi) \leq B$?

Example: Traveling Salesman (2)

Instance: Integers $d(i,j)$ for $1 \leq i \neq j \leq n$ and B
Feasible: Permutation π of $1, \ldots, n$

Optimization: Find minimal tour length $d(\pi)$
Decision: Does there exist a permutation π with $d(\pi) \leq B$?

Theorem

If the decision problem of TSP is solvable in polynomial time, then also the optimization problem of TSP is solvable in polynomial time.

Proof idea:
With the help of a polynomial time algorithm A for the decision problem we will construct a polynomial time algorithm B for computing the optimal objective value for the optimization problem.

Example: Traveling Salesman (3)

Algorithm B

We perform a binary search (bisection search) with the following parameters:

- The minimal length is 0.
- The maximal length is $L := \sum_{i=1}^{n} \sum_{j=i+1}^{n} d(i,j)$.
- We find the optimal objective value by binary search over the range $\{0, \ldots, L\}$.
- In each iteration, we apply the polynomial time algorithm A (for the decision problem) to tell us in which half we have to search on.

The number of iterations in the binary search is $\lceil \log(L + 1) \rceil$.

Example: Traveling Salesman (4)

Investigation of the instance size:

- The coding size of $a \in \mathbb{N}$ is $\kappa(a) := \lceil \log(a+1) \rceil$.
- The function κ is subadditive:
  For all $a, b \in \mathbb{N}$ we have $\kappa(a+b) \leq \kappa(a) + \kappa(b)$.

Example: Traveling Salesman (4)

Investigation of the instance size:

- The coding size of $a \in \mathbb{N}$ is $\kappa(a) := \lceil \log(a+1) \rceil$.
- The function κ is subadditive:
  For all $a, b \in \mathbb{N}$ we have $\kappa(a+b) \leq \kappa(a) + \kappa(b)$.
- The instance size $|I|$ of the Traveling Salesman Problem is at least

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} \kappa(d(i,j)) \leq \kappa\left(\sum_{i=1}^{n} \sum_{j=i+1}^{n} d(i,j) \right) = \kappa(L) = \lceil \log(L+1) \rceil$$

Example: Traveling Salesman (4)

Investigation of the instance size:

- The coding size of $a \in \mathbb{N}$ is $\kappa(a) := \lceil \log(a+1) \rceil$.
- The function κ is subadditive:
  For all $a, b \in \mathbb{N}$ we have $\kappa(a+b) \leq \kappa(a) + \kappa(b)$.
- The instance size $|I|$ of the Traveling Salesman Problem is at least

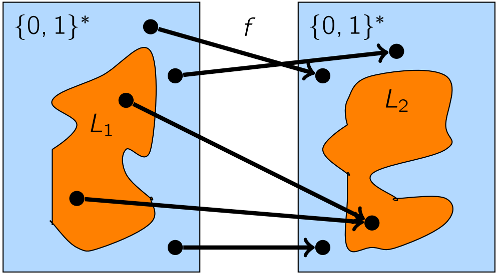
$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} \kappa(d(i,j)) \leq \kappa\left(\sum_{i=1}^{n} \sum_{j=i+1}^{n} d(i,j)\right) = \kappa(L) = \lceil \log(L+1) \rceil$$

- Algorithm B essentially consists of $\lceil \log(L+1) \rceil \leq |I|$ calls of the polynomial time Algorithm A.
- Hence the overall running time of algorithm B is polynomially bounded in the instance size.

Polynomial-Time Reductions (1)

Definition

Let L_1 and L_2 be languages (problems) over Σ_1 respectively Σ_2.
Then L_1 is **polynomially reducible** to L_2 (with the notation $L_1 \leq_p L_2$),
if there exists a polynomially computable function $f : \Sigma_1^* \to \Sigma_2^*$
so that for all $x \in \Sigma_1^*$ we have: $x \in L_1 \Leftrightarrow f(x) \in L_2$.



Polynomial-Time Reductions (2a)

Theorem

If $L_1 \leq_p L_2$ and if $L_2 \in P$, then $L_1 \in P$.

Polynomial-Time Reductions (2a)

Theorem

If $L_1 \leq_p L_2$ and if $L_2 \in P$, then $L_1 \in P$.

Proof

- The reduction f has polynomial run time $p(\cdot)$
- Algorithm A_2 decides L_2 in polynomial time $q(\cdot)$

Polynomial-Time Reductions (2a)

Theorem

If $L_1 \leq_p L_2$ and if $L_2 \in P$, then $L_1 \in P$.

Proof

- The reduction f has polynomial run time $p(\cdot)$
- Algorithm A_2 decides L_2 in polynomial time $q(\cdot)$

We construct a new algorithm A_1 that decides L_1:

Step 1: Compute $f(x)$
Step 2: Simulate algorithm A_2 with input $f(x)$
Step 3: Accept x, if and only if A_2 accepts $f(x)$

Polynomial-Time Reductions (2a)

Theorem

If $L_1 \leq_p L_2$ and if $L_2 \in P$, then $L_1 \in P$.

Proof

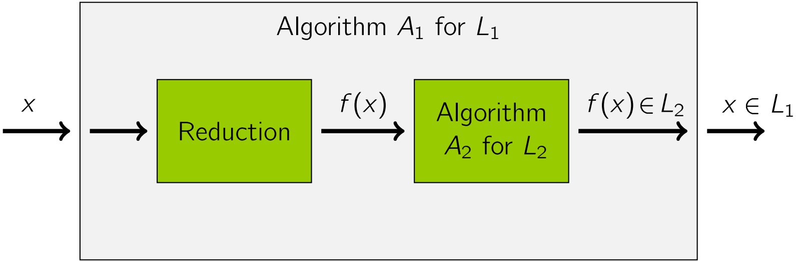
- The reduction f has polynomial run time $p(\cdot)$
- Algorithm A_2 decides L_2 in polynomial time $q(\cdot)$

We construct a new algorithm A_1 that decides L_1:

Step 1: Compute $f(x)$
Step 2: Simulate algorithm A_2 with input $f(x)$
Step 3: Accept x, if and only if A_2 accepts $f(x)$

Step 1 has run time $p(|x|)$ and
Step 2 has run time $q(|f(x)|) \leq q(p(|x|) + |x|)$

Polynomial-Time Reductions (2b)



Polynomial-Time Reductions (2c)

On the last two slides we have shown:

Theorem

If $L_1 \leq_p L_2$ and if $L_2 \in P$, then $L_1 \in P$.

Intuition for $L_1 \leq_p L_2$:

- If L_2 is easy, then also L_1 is easy
- If L_1 is difficult, then also L_2 is difficult

Polynomial-Time Reductions (3)

Lemma

Reducibility is a transitive relation:
$L_1 \leq_p L_2$ and $L_2 \leq_p L_3$ implies $L_1 \leq_p L_3$

Proof: by putting the two tranformations into series

COLORING $\leq_p$ SAT

Problem: COLORING

Instance: An undirected graph $G = (V, E)$; a number $k \in \mathbb{N}$

Question: Does there exist a coloring $c : V \to \{1, \ldots, k\}$ of the vertices with k colors, so that adjacent vertices receive distinct colors?
In other words, we would like to have $\forall e = \{u, v\} \in E : \; c(u) \neq c(v)$

Problem: Satisfiability (SAT)

Instance: A logical formula Φ in CNF over $X = \{x_1, \ldots, x_n\}$

Question: Does there exist a truth setting for X that satisfies Φ?

Theorem

COLORING $\leq_p$ SAT

COLORING $\leq_p$ SAT: The Reduction

The Boolean variables

For every vertex $v \in V$ and for every color $i \in \{1, \ldots, k\}$ we introduce a Boolean variable x_v^i.

COLORING $\leq_p$ SAT: The Reduction

The Boolean variables

For every vertex $v \in V$ and for every color $i \in \{1, \ldots, k\}$
we introduce a Boolean variable x_v^i.

The clauses

For every vertex $v \in V$
we introduce the clause $(x_v^1 + x_v^2 + \ldots + x_v^k)$

For every edge $\{u, v\} \in E$ and for every color $i \in \{1, \ldots, k\}$
we introduce the clause $(\bar{x}_u^i + \bar{x}_v^i)$

COLORING $\leq_p$ SAT: The Reduction

The Boolean variables

For every vertex $v \in V$ and for every color $i \in \{1, \ldots, k\}$
we introduce a Boolean variable x_v^i.

The clauses

For every vertex $v \in V$
we introduce the clause $(x_v^1 + x_v^2 + \ldots + x_v^k)$

For every edge $\{u, v\} \in E$ and for every color $i \in \{1, \ldots, k\}$
we introduce the clause $(\bar{x}_u^i + \bar{x}_v^i)$

- Number of variables = $k\,|V|$
- Number of clauses = $|V| + k\,|E|$
- Total size of formula = $k\,|V| + 2k\,|E| \;\in\; O(k|V|^2)$

COLORING $\leq_p$ SAT: Correctness (1)

Graph G has k-coloring $\Rightarrow$ formula φ is satisfiable

COLORING $\leq_p$ SAT: Correctness (1)

Graph G has k-coloring $\Rightarrow$ formula φ is satisfiable

- Let c be a k-coloring of G

COLORING $\leq_p$ SAT: Correctness (1)

Graph G has k-coloring $\Rightarrow$ formula φ is satisfiable

- Let c be a k-coloring of G
- For every vertex $v \in V$ with $c(v) = i$ we set $x_v^i = 1$.
  All other variables are set to 0.

COLORING $\leq_p$ SAT: Correctness (1)

Graph G has k-coloring $\Rightarrow$ formula φ is satisfiable

- Let c be a k-coloring of G
- For every vertex $v \in V$ with $c(v) = i$ we set $x_v^i = 1$.
  All other variables are set to 0.
- For every vertex $v \in V$ the clause $(x_v^1 + x_v^2 + \ldots + x_v^k)$ is satisfied

COLORING $\leq_p$ SAT: Correctness (1)

Graph G has k-coloring $\Rightarrow$ formula φ is satisfiable

- Let c be a k-coloring of G
- For every vertex $v \in V$ with $c(v) = i$ we set $x_v^i = 1$.
  All other variables are set to 0.
- For every vertex $v \in V$ the clause $(x_v^1 + x_v^2 + \ldots + x_v^k)$ is satisfied
- For $\{u, v\} \in E$ and $i \in \{1, \ldots, k\}$ the clause $(\bar{x}_u^i + \bar{x}_v^i)$ is satisfied
  (Otherwise both vertices u and v have the same color i.)

COLORING $\leq_p$ SAT: Correctness (1)

Graph G has k-coloring $\Rightarrow$ formula φ is satisfiable

- Let c be a k-coloring of G
- For every vertex $v \in V$ with $c(v) = i$ we set $x_v^i = 1$.
  All other variables are set to 0.
- For every vertex $v \in V$ the clause $(x_v^1 + x_v^2 + \ldots + x_v^k)$ is satisfied
- For $\{u, v\} \in E$ and $i \in \{1, \ldots, k\}$ the clause $(\bar{x}_u^i + \bar{x}_v^i)$ is satisfied
  (Otherwise both vertices u and v have the same color i.)
- Hence: This truth assignment satisfies formula φ

COLORING $\leq_p$ SAT: Correctness (2)

Formula φ is satisfiable $\Rightarrow$ graph G has k-coloring

COLORING $\leq_p$ SAT: Correctness (2)

Formula φ is satisfiable $\Rightarrow$ graph G has k-coloring

- Consider a satisfying truth assignment for φ

COLORING $\leq_p$ SAT: Correctness (2)

Formula φ is satisfiable $\Rightarrow$ graph G has k-coloring

- Consider a satisfying truth assignment for φ
- Because of clause $(x_v^1 + x_v^2 + \ldots + x_v^k)$, for every vertex v there is at least one color i with $x_v^i = 1$

COLORING $\leq_p$ SAT: Correctness (2)

Formula φ is satisfiable $\Rightarrow$ graph G has k-coloring

- Consider a satisfying truth assignment for φ
- Because of clause $(x_v^1 + x_v^2 + \ldots + x_v^k)$, for every vertex v there is at least one color i with $x_v^i = 1$
- For every vertex we pick one such color

COLORING $\leq_p$ SAT: Correctness (2)

Formula φ is satisfiable $\Rightarrow$ graph G has k-coloring

- Consider a satisfying truth assignment for φ
- Because of clause $(x_v^1 + x_v^2 + \ldots + x_v^k)$, for every vertex v there is at least one color i with $x_v^i = 1$
- For every vertex we pick one such color
- We claim: $c(u) \neq c(v)$ holds for every edge $\{u, v\} \in E$

COLORING $\leq_p$ SAT: Correctness (2)

Formula φ is satisfiable $\Rightarrow$ graph G has k-coloring

- Consider a satisfying truth assignment for φ
- Because of clause $(x_v^1 + x_v^2 + \ldots + x_v^k)$, for every vertex v there is at least one color i with $x_v^i = 1$
- For every vertex we pick one such color
- We claim: $c(u) \neq c(v)$ holds for every edge $\{u, v\} \in E$
- Proof: If $c(u) = c(v) = i$, then $x_u^i = x_v^i = 1$. But then the clause $(\bar{x}_u^i + \bar{x}_v^i)$ would be violated

COLORING $\leq_p$ SAT: Consequences

Our reduction COLORING $\leq_p$ SAT implies the following:

Corollary

If SAT possesses a polynomial algorithm,
then also COLORING possesses a polynomial algorithm.

Corollary

If COLORING cannot be solved in polynomial time,
then also SAT cannot be solved in polynomial time.

Vertex Cover $\leq_p$ SAT

Problem: Vertex Cover (VC)

Instance: An undirected graph $G = (V, E)$; an integer $k \in \mathbb{N}$

Question: Does G allow a vertex cover with $\leq k$ vertices?

Vertex Cover $S \subseteq V$ contains (at least) one end-vertex of every edge

Problem: Satisfiability (SAT)

Instance: A logical formula Φ in CNF over $X = \{x_1, \ldots, x_n\}$

Question: Does there exist a truth setting for X that satisfies Φ?

Theorem

Vertex Cover $\leq_p$ SAT

Vertex Cover $\leq_p$ SAT: The Reduction (1st try)

The Boolean variables

For every vertex $v \in V$
we introduce a Boolean variable x_v.

Vertex Cover $\leq_p$ SAT: The Reduction (1st try)

The Boolean variables

For every vertex $v \in V$
we introduce a Boolean variable x_v.

The clauses

For every edge $\{u, v\} \in E$
we introduce the clause $(x_u + x_v)$

For every $(k+1)$-element subset $S \subseteq V$
we introduce the clause $\bigvee_{v \in S} \bar{x}_v$

Vertex Cover $\leq_p$ SAT: The Reduction (1st try)

The Boolean variables

For every vertex $v \in V$
we introduce a Boolean variable x_v.

The clauses

For every edge $\{u, v\} \in E$
we introduce the clause $(x_u + x_v)$

For every $(k+1)$-element subset $S \subseteq V$
we introduce the clause $\bigvee_{v \in S} \bar{x}_v$

- Number of variables = $|V|$
- Number of clauses = $\approx |V|^k$
- Total size of formula $\approx k\,|V|^k$ $\leftarrow$ **!!!#!!&!!!!!!!**

Vertex Cover $\leq_p$ SAT: The Reduction (2nd try)

The Boolean variables

For every vertex $v \in V$ and for every $i \in \{1, \ldots, k\}$
we introduce a Boolean variable x_v^i.
(Meaning: v is the i-th vertex in the cover)

Vertex Cover $\leq_p$ SAT: The Reduction (2nd try)

The Boolean variables

For every vertex $v \in V$ and for every $i \in \{1, \ldots, k\}$
we introduce a Boolean variable x_v^i.
(Meaning: v is the i-th vertex in the cover)

The clauses

For every edge $\{u, v\} \in E$
we introduce the clause $(x_u^1 + x_u^2 + \cdots + x_u^k \; + \; x_v^1 + x_v^2 + \cdots + x_v^k)$

Vertex Cover $\leq_p$ SAT: The Reduction (2nd try)

The Boolean variables

For every vertex $v \in V$ and for every $i \in \{1, \dots, k\}$
we introduce a Boolean variable x_v^i.
(Meaning: v is the i-th vertex in the cover)

The clauses

For every edge $\{u, v\} \in E$
we introduce the clause $(x_u^1 + x_u^2 + \cdots + x_u^k \; + \; x_v^1 + x_v^2 + \cdots + x_v^k)$

For every pair of vertices $u, v \in V$ with $u \neq v$ and for all $i \in \{1, \dots, k\}$
we introduce the clause $(\bar{x}_u^i + \bar{x}_v^i)$

Vertex Cover $\leq_p$ SAT: The Reduction (2nd try)

The Boolean variables

For every vertex $v \in V$ and for every $i \in \{1, \ldots, k\}$
we introduce a Boolean variable x_v^i.
(Meaning: v is the i-th vertex in the cover)

The clauses

For every edge $\{u, v\} \in E$
we introduce the clause $(x_u^1 + x_u^2 + \cdots + x_u^k \ + \ x_v^1 + x_v^2 + \cdots + x_v^k)$

For every pair of vertices $u, v \in V$ with $u \neq v$ and for all $i \in \{1, \ldots, k\}$
we introduce the clause $(\bar{x}_u^i + \bar{x}_v^i)$

- Number of variables = ???
- Number of clauses = ???
- Total size of formula = ???

Exercise

Problem: Independent Set (INDEP-SET)

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

Question: Does G contain an independent set of size $\geq k$?

Hamiltonian cycle (Ham-Cycle)

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

Question: Does G contain a Hamiltonian cycle?
(a simple cycle that visits every vertex exactly once)

Exercise

(a) Prove: INDEP-SET $\leq_p$ SAT

(b) Prove: Ham-Cycle $\leq_p$ SAT

Exercise

Problem: EvenPath

Instance: an undirected graph $G = (V, E)$; two vertices $s, t \in V$
Question: does there exist a simple path from s to t
that uses an **even** number of edges?

Problem: OddPath

Instance: an undirected graph $G' = (V', E')$; two vertices $s', t' \in V'$
Question: does there exist a simple path from s' to t'
that uses an **odd** number of edges?

Exercise

(a) Prove: EvenPath $\leq_p$ OddPath.
(b) Prove: OddPath $\leq_p$ EvenPath.

NP-Hardness and NP-Completeness (1)

Definition

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

NP-Hardness and NP-Completeness (1)

Definition

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

Theorem

If problem L is NP-hard, we have: $L \in P \;\Rightarrow\; P = NP$

Proof: A polynomial time algorithm for L together with the reduction $L' \leq_p L$ yields a polynomial time algorithm for every $L' \in NP$.

Consequence:
NP-hard problems cannot be solved in polynomial time, unless P=NP.

NP-Hardness and NP-Completeness (2)

Definition

A decision problem L is **NP-complete**,

- if $L \in$ NP, and
- if L is NP-hard.

The class of NP-complete problems is denoted by **NPC**.

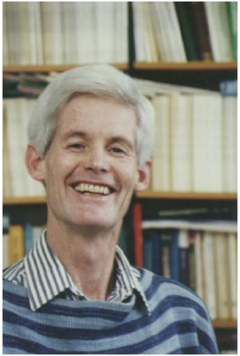
Intuition:

- NP-complete problems are the hardest problems in NP
- Recall: NP is huge and contains tons of important problems
- Unless P=NP, NP-complete problems cannot be solved in poly-time
- NP-complete problems are considered to be intractable

Stephen Arthur Cook OC (1939)

Wikipedia: Steve Cook is an American-Canadian computer scientist and mathematician who has made major contributions to the fields of complexity theory and proof complexity.

His seminal paper *"The complexity of theorem proving procedures"* (presented at the 1971 Symposium on the Theory of Computing) laid the foundations for the theory of NP-Completeness.
The ensuing exploration of the boundaries and nature of the class of NP-complete problems has become one of the most active and important research areas in computer science.



Leonid Anatolievich Levin (1948)

Wikipedia: Leonid Levin is a Soviet-American computer scientist. He obtained his master's degree at Moscow University in 1970 where he studied under Andrej Kolmogorov.

Leonid Levin and Stephen Cook independently discovered the existence of NP-complete problems. Levin is known for his work in randomness in computing, average-case complexity, algorithmic probability, theory of computation, and information theory.

Theorem of Cook & Levin

The starting point for all our NP-completeness proofs is the satisfiability problem SAT.

Problem: Satisfiability (SAT)

Instance: A logical formula Φ in CNF over $X = \{x_1, \ldots, x_n\}$

Question: Does there exist a truth setting for X that satisfies Φ?

Theorem (Cook & Levin)

SAT is NP-complete.

Proof: Long & technical. Omitted.

Hence: If $P \neq NP$, then SAT cannot be solved in polynomial time.

A Cooking Recipe for NP-Completeness Proofs (1)

- The NP-completeness of SAT is established by a very long and very technical "master-reduction" from all problems in NP to SAT
- For proving the NP-completeness of other problems, we could of course construct for every new problem another long and technical master-reduction
- But there is a much easier approach to establish NP-completeness via the NP-completeness of SAT

A Cooking Recipe for NP-Completeness Proofs (1)

- The NP-completeness of SAT is established by a very long and very technical "master-reduction" from all problems in NP to SAT
- For proving the NP-completeness of other problems, we could of course construct for every new problem another long and technical master-reduction
- But there is a much easier approach to establish NP-completeness via the NP-completeness of SAT

Theorem

For any NP-hard problem L^* we have: $L^* \leq_p L \;\Rightarrow\; L$ is NP-hard

A Cooking Recipe for NP-Completeness Proofs (1)

- The NP-completeness of SAT is established by a very long and very technical "master-reduction" from all problems in NP to SAT
- For proving the NP-completeness of other problems, we could of course construct for every new problem another long and technical master-reduction
- But there is a much easier approach to establish NP-completeness via the NP-completeness of SAT

Theorem

For any NP-hard problem L^* we have: $L^* \leq_p L \;\Rightarrow\; L$ is NP-hard

Proof:

- For all $L' \in NP$ we have $L' \leq_p L^*$ and $L^* \leq_p L$.
- Transitivity of $\leq_p$ implies $L' \leq_p L$ for all $L' \in NP$.

A Cooking Recipe for NP-Completeness Proofs (2)

Here is our cooking recipe:

1. Show that $L \in NP$.
2. Pick an NP-complete language L^*.
3. **(Reduction):**
   Construct a function f, that maps instances of L^* to instances of L.
4. **(Polynomial time):**
   Show that f can be computed in polynomial time.
5. **(Correctness):**
   Show that for $x \in \{0,1\}^*$ we have $x \in L^*$ if and only if $f(x) \in L$.

3-SAT: Definition

- A k-**clause** is a clause that consists of exactly k literals
- A CNF-formula φ is in k-**CNF**, if it consists of k-clauses

Example of a formula in 3-CNF

$$\varphi = \underbrace{(\bar{x}_1 \vee \bar{x}_2 \vee x_3)}_{\text{3 literals}} \wedge \underbrace{(\bar{x}_1 \vee x_2 \vee \bar{x}_3)}_{\text{3 literals}}$$

3-SAT: Definition

- A k-**clause** is a clause that consists of exactly k literals
- A CNF-formula φ is in k-**CNF**, if it consists of k-clauses

Example of a formula in 3-CNF

$$\varphi = \underbrace{(\bar{x}_1 \vee \bar{x}_2 \vee x_3)}_{\text{3 literals}} \wedge \underbrace{(\bar{x}_1 \vee x_2 \vee \bar{x}_3)}_{\text{3 literals}}$$

Problem: 3-SAT

Instance: A logical formula Φ in 3-CNF
Question: Does there exist a satisfying truth setting?

3-SAT is a special case of SAT and hence lies (exactly as SAT) in NP

3-SAT: NP-Completeness (Start)

Theorem

SAT $\leq_p$ 3-SAT

3-SAT: NP-Completeness (Start)

Theorem

SAT $\leq_p$ 3-SAT

Proof:

- Consider an arbitrary formula φ in CNF (instance of SAT)
- We will construct a formula φ' in 3-CNF that is equivalent to formula φ':

  φ is satisfiable $\Leftrightarrow$ φ' is satisfiable

3-SAT: NP-Completeness (Start)

Theorem

SAT $\leq_p$ 3-SAT

Proof:

- Consider an arbitrary formula φ in CNF (instance of SAT)
- We will construct a formula φ' in 3-CNF that is equivalent to formula φ':
  φ is satisfiable $\Leftrightarrow$ φ' is satisfiable

- A 1-clause or 2-clause becomes an equivalent 3-clause by duplicating one or two literals
- 3-clauses remain 3-clauses

3-SAT: NP-Completeness (Start)

Theorem

SAT $\leq_p$ 3-SAT

Proof:

- Consider an arbitrary formula φ in CNF (instance of SAT)
- We will construct a formula φ' in 3-CNF that is equivalent to formula φ':
  φ is satisfiable $\Leftrightarrow$ φ' is satisfiable

- A 1-clause or 2-clause becomes an equivalent 3-clause by duplicating one or two literals
- 3-clauses remain 3-clauses
- k-clauses with $k \geq 4$ are handled by repeatedly applying the following **clause-transformation**:
  The clause $c = (\ell_1 + \ell_2 + \ell_3 + \cdots + \ell_k)$ is replaced by the two new clauses $(\ell_1 + \cdots + \ell_{k-2} + a)$ and $(\bar{a} + \ell_{k-1} + \ell_k)$. Here a denotes a newly created auxiliary variable.

Clause-Transformation: Example

Clause-transformation for a 5-clause

- We start from the 5-clause $(x_1 + \bar{x}_2 + x_3 + x_4 + \bar{x}_5)$
- In the first transformation step we create a 4-clause and a 3-clause: $(x_1 + \bar{x}_2 + x_3 + a_1)\ (\bar{a}_1 + x_4 + \bar{x}_5)$

Clause-Transformation: Example

Clause-transformation for a 5-clause

- We start from the 5-clause $(x_1 + \bar{x}_2 + x_3 + x_4 + \bar{x}_5)$
- In the first transformation step we create a 4-clause and a 3-clause: $(x_1 + \bar{x}_2 + x_3 + a_1)\ (\bar{a}_1 + x_4 + \bar{x}_5)$
- Then we apply the transformation to the new 4-clause and get $(x_1 + \bar{x}_2 + a_2)\ (\bar{a}_2 + x_3 + a_1)\ (\bar{a}_1 + x_4 + \bar{x}_5)$. We terminate, as only 3-clauses remain.

Clause-Transformation: Correctness

Old clause: $c = (\ell_1 + \ell_2 + \ell_3 + \cdots + \ell_k)$
New clauses: $c' = (\ell_1 + \cdots + \ell_{k-2} + a)$ and $c'' = (\bar{a} + \ell_{k-1} + \ell_k)$

Clause-Transformation: Correctness

Old clause: $c = (\ell_1 + \ell_2 + \ell_3 + \cdots + \ell_k)$
New clauses: $c' = (\ell_1 + \cdots + \ell_{k-2} + a)$ and $c'' = (\bar{a} + \ell_{k-1} + \ell_k)$

(1) If a truth setting satisfies the two new clauses c' and c'', then it automatically also satisfies the old clause c:

- If $a = 0$, then $\ell_1 + \cdots + \ell_{k-2}$ is true
- If $a = 1$, then $\ell_{k-1} + \ell_k$ is true

Clause-Transformation: Correctness

Old clause: $c = (\ell_1 + \ell_2 + \ell_3 + \cdots + \ell_k)$
New clauses: $c' = (\ell_1 + \cdots + \ell_{k-2} + a)$ and $c'' = (\bar{a} + \ell_{k-1} + \ell_k)$

(1) If a truth setting satisfies the two new clauses c' and c'', then it automatically also satisfies the old clause c:

- If $a = 0$, then $\ell_1 + \cdots + \ell_{k-2}$ is true
- If $a = 1$, then $\ell_{k-1} + \ell_k$ is true

(2) If a truth setting satisfies the old clause c, then it can be extended to auxiliary variable a so that both new clauses c' and c'' are satisfied:

- The truth setting has at least one true literal in clause c
- If $\ell_1 + \cdots + \ell_{k-2}$ is true, then we set $a = 0$
- If $\ell_{k-1} + \ell_k$ is true, then we set $a = 1$

3-SAT: NP-Completeness (End)

- By applying the clause-transformation, we turn a k-clause into a $(k-1)$-clause and a 3-clause.
- After $k-3$ iterations a single old k-clause has turned into $k-2$ new 3-clauses.
- Hence $k \geq 4$ old literals turn into $3k-6$ new literals.
- This transformation is applied over and over again, until the formula only has 3-clauses.

3-SAT: NP-Completeness (End)

- By applying the clause-transformation, we turn a k-clause into a $(k-1)$-clause and a 3-clause.
- After $k-3$ iterations a single old k-clause has turned into $k-2$ new 3-clauses.
- Hence $k \geq 4$ old literals turn into $3k-6$ new literals.
- This transformation is applied over and over again, until the formula only has 3-clauses.

- If φ contains p literals, then φ' contains at most $3p$ literals.
- Therefore the run time of the reduction is polynomially bounded.

3-SAT: NP-Completeness (End)

- By applying the clause-transformation, we turn a k-clause into a $(k-1)$-clause and a 3-clause.
- After $k-3$ iterations a single old k-clause has turned into $k-2$ new 3-clauses.
- Hence $k \geq 4$ old literals turn into $3k-6$ new literals.
- This transformation is applied over and over again, until the formula only has 3-clauses.

- If φ contains p literals, then φ' contains at most $3p$ literals.
- Therefore the run time of the reduction is polynomially bounded.

Theorem

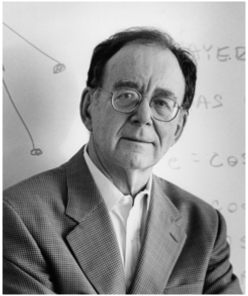
SAT $\leq_p$ 3-SAT

Richard Manning Karp (1935)

Wikipedia: Richard Karp is an American computer scientist, who has made many important discoveries in computer science, operations research, and in the area of combinatorial algorithms.

Karp introduced the now standard methodology for proving problems to be NP-complete which has led to the identification of many practical problems as being computationally difficult.

- Edmonds-Karp algorithm for max-flow
- Hopcroft-Karp algorithm for matching
- Rabin-Karp string search algorithm
- Karp-Lipton theorem



Karp's list with 21 NP-complete problems

In 1972 Richard Karp established the NP-completeness of 21 combinatorial and graph-theoretic problems.

SAT	3-SAT
INTEGER PROGRAMMING	COLORING
CLIQUE	CLIQUE COVER
INDEPENDENT-SET	EXACT COVER
VERTEX COVER	3-DIM MATCHING
SET COVER	STEINER TREE
FEEDBACK ARC SET	HITTING SET
FEEDBACK VERTEX SET	SUBSET-SUM
DIR HAM-CYCLE	JOB SEQUENCING
UND HAM-CYCLE	PARTITION
	MAX-CUT

Landscape with Karp's 20 reductions