Algorithms and Complexity (AC), week 3

Marie Schmidt

(Based on slides by Gerhard Woeginger and Jesper Nederlof)

Landelijk Netwerk Mathematische Besliskunde

LNMB, Sep–Nov 2017

Our program for rest of week 3

- pseudo-polynomial time, strong NP-hardness & weak NP-hardness
- co-NP, co-NP versus NP
- An unsolvable problem

Subset Sum (SS)

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

Question: does there exist an index set $J \subseteq \{1, \ldots, n\}$ with $\sum_{j \in J} a_j = b$?

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

Subset Sum (SS)

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

Question: does there exist an index set $J \subseteq \{1, \dots, n\}$ with $\sum_{j \in J} a_j = b$?

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

Theorem

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

Proof:

Subset Sum (SS)

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

Question: does there exist an index set $J \subseteq \{1, \ldots, n\}$ with $\sum_{j \in J} a_j = b$?

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

Theorem

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

Proof: by reduction from Ex-Cov.

Subset Sum (SS)

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

Question: does there exist an index set $J \subseteq \{1, \ldots, n\}$ with $\sum_{j \in J} a_j = b$?

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

Theorem

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

Proof: by reduction from Ex-Cov.
Let $(X = \{x_1, \ldots, x_m\}, \{S_1, \ldots, S_n\})$ be an instance of Ex-Cov.
Define numbers a_j as $a_j := \sum_{i=1}^{m} c_{ij} \cdot d_i$ with $c_{ij} = 1$ if $x_i \in S_j$ and $d_i = (n+1)^{i-1}$.
Set $b := \sum_{i=1}^{m} (n+1)^{i-1}$.

Subset Sum (SS)

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

Question: does there exist an index set $J \subseteq \{1, \dots, n\}$ with $\sum_{j \in J} a_j = b$?

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

Theorem

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

Proof: by reduction from Ex-Cov.
Let $(X = \{x_1, \dots, x_m\}, \{S_1, \dots, S_n\})$ be an instance of Ex-Cov.
Define numbers a_j as $a_j := \sum_{i=1}^{m} c_{ij} \cdot d_i$ with $c_{ij} = 1$ if $x_i \in S_j$ and $d_i = (n+1)^{i-1}$.
Set $b := \sum_{i=1}^{m} (n+1)^{i-1}$.

Show:
J index set of a solution to Ex-Cov $\Leftrightarrow$ J index set of a solution to SS.

Subset Sum (SS)

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

Question: does there exist an index set $J \subseteq \{1, \ldots, n\}$ with $\sum_{j \in J} a_j = b$?

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

Theorem

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

Proof: by reduction from Ex-Cov.
Let $(X = \{x_1, \ldots, x_m\}, \{S_1, \ldots, S_n\})$ be an instance of Ex-Cov.
Define numbers a_j as $a_j := \sum_{i=1}^{m} c_{ij} \cdot d_i$ with $c_{ij} = 1$ if $x_i \in S_j$ and $d_i = (n+1)^{i-1}$.
Set $b := \sum_{i=1}^{m} (n+1)^{i-1}$.

Show:
J index set of a solution to Ex-Cov $\Leftrightarrow$ J index set of a solution to SS.
Also: argue why this is a *polynomial-time* transformation.

size(I) = instance size
= length (number of symbols) of reasonable encoding of instance I

number(I)
= value of the largest number occuring in instance I

size(I) = instance size
= length (number of symbols) of reasonable encoding of instance I

number(I)
= value of the largest number occuring in instance I

Example

In an SS instance $I = (A, b)$

- number(I)$= \max\{b, \max_{i=1}^{n} a_i\}$
- size(I)$= \Theta(\log b + \sum_{i=1}^{n} \log a_i)$.

An algorithm for SUBSET SUM

Define function $F[k, c]$:
$F[k, c]$=TRUE if and only if $\exists S \subseteq \{1, \ldots, k\} : \sum_{i \in S} a_i = c$

An algorithm for SUBSET SUM

Define function $F[k, c]$:
$F[k, c]$=TRUE if and only if $\exists S \subseteq \{1, \ldots, k\} : \sum_{i \in S} a_i = c$
We are interested in $F[n, b]$!

An algorithm for SUBSET SUM

Define function $F[k, c]$:
$F[k, c]$=TRUE if and only if $\exists S \subseteq \{1, \ldots, k\} : \sum_{i \in S} a_i = c$
We are interested in $F[n, b]$!

Dynamic programming algorithm to compute $F[n, b]$

Input: a set of positive integers $a_1, \ldots, a_n$; a bound b
Output: 'YES' if there is a subset I' of index set I with $\sum_{i \in I'} a_i = b$, 'NO' otherwise

An algorithm for SUBSET SUM

Define function $F[k, c]$:
$F[k, c]$=TRUE if and only if $\exists S \subseteq \{1, \dots, k\} : \sum_{i \in S} a_i = c$
We are interested in $F[n, b]$!

Dynamic programming algorithm to compute $F[n, b]$

Input: a set of positive integers $a_1, \dots, a_n$; a bound b
Output: 'YES' if there is a subset I' of index set I with $\sum_{i \in I'} a_i = b$, 'NO' otherwise

$F[0, 0] := TRUE$, $F[0, c] := NO$ for all $c = 1, \dots, b$

An algorithm for SUBSET SUM

Define function $F[k, c]$:
$F[k, c]$=TRUE if and only if $\exists S \subseteq \{1, \ldots, k\} : \sum_{i \in S} a_i = c$
We are interested in $F[n, b]$!

Dynamic programming algorithm to compute $F[n, b]$

Input: a set of positive integers $a_1, \ldots, a_n$; a bound b
Output: 'YES' if there is a subset I' of index set I with $\sum_{i \in I'} a_i = b$, 'NO' otherwise

$F[0, 0] := TRUE$, $F[0, c] := NO$ for all $c = 1, \ldots, b$
for $k = 1, \ldots, n$ **do**
  for $c = 1, \ldots, b$ **do**
    $F[i, c] = F[i-1, c] \vee F[i-1, c-a_i]$
  end for
end for

An algorithm for SUBSET SUM

Define function $F[k, c]$:
$F[k, c]$=TRUE if and only if $\exists S \subseteq \{1, \ldots, k\} : \sum_{i \in S} a_i = c$
We are interested in $F[n, b]$!

Dynamic programming algorithm to compute $F[n, b]$

Input: a set of positive integers $a_1, \ldots, a_n$; a bound b
Output: 'YES' if there is a subset I' of index set I with $\sum_{i \in I'} a_i = b$, 'NO' otherwise

$F[0, 0] := TRUE$, $F[0, c] := NO$ for all $c = 1, \ldots, b$
for $k = 1, \ldots, n$ **do**
  for $c = 1, \ldots, b$ **do**
    $F[i, c] = F[i-1, c] \vee F[i-1, c-a_i]$
  end for
end for
return $F[n, b]$

An algorithm for SUBSET SUM

Define function $F[k, c]$:
$F[k, c]$=TRUE if and only if $\exists S \subseteq \{1, \ldots, k\} : \sum_{i \in S} a_i = c$
We are interested in $F[n, b]$!

Dynamic programming algorithm to compute $F[n, b]$

Input: a set of positive integers $a_1, \ldots, a_n$; a bound b
Output: 'YES' if there is a subset I' of index set I with $\sum_{i \in I'} a_i = b$, 'NO' otherwise

$F[0, 0] := TRUE$, $F[0, c] := NO$ for all $c = 1, \ldots, b$
for $k = 1, \ldots, n$ **do**
  for $c = 1, \ldots, b$ **do**
    $F[i, c] = F[i-1, c] \vee F[i-1, c-a_i]$
  end for
end for
return $F[n, b]$

Running time of this algorithm?

Pseudo-polynomial time

Definition

A decision problem X is solvable in pseudo-polynomial time,
if there exists an algorithm that solves instances I of X
in time polynomially bounded in size(I) and number(I).

Pseudo-polynomial time

Definition

A decision problem X is solvable in pseudo-polynomial time,
if there exists an algorithm that solves instances I of X
in time polynomially bounded in size(I) and number(I).

Which of the decision problems we studied so far is solvable in pseudo-polynomial time?

- SAT?
- 3-SAT?
- CLIQUE?
- IS?
- VC?
- Ex-Cov?
- SUBSET SUM
- PARTITION?
- HC?
- TSP?

Pseudo-polynomial time

Definition

A decision problem X is solvable in pseudo-polynomial time,
if there exists an algorithm that solves instances I of X
in time polynomially bounded in size(I) and number(I).

Observation: number(I) is only relevant for problems that involve numbers (distances, costs, weights, lengths, penalties, profits, time intervals, etc)

Which of the decision problems we studied so far is solvable in pseudo-polynomial time?

- SAT?
- 3-SAT?
- CLIQUE?
- IS?
- VC?
- Ex-Cov?
- SUBSET SUM
- PARTITION?
- HC?
- TSP?

Pseudo-polynomial time

Definition

A decision problem X is solvable in pseudo-polynomial time,
if there exists an algorithm that solves instances I of X
in time polynomially bounded in size(I) and number(I).

Observation: number(I) is only relevant for problems that involve numbers (distances, costs, weights, lengths, penalties, profits, time intervals, etc)

Which of the decision problems we studied so far is solvable in pseudo-polynomial time?

- SAT
- 3-SAT
- CLIQUE
- IS
- VC
- Ex-Cov
- SUBSET SUM
- PARTITION?
- HC
- TSP?

Pseudo-polynomial time

Definition

A decision problem X is solvable in pseudo-polynomial time,
if there exists an algorithm that solves instances I of X
in time polynomially bounded in size(I) and number(I).

Observation: number(I) is only relevant for problems that involve numbers (distances, costs, weights, lengths, penalties, profits, time intervals, etc)

Which of the decision problems we studied so far is solvable in pseudo-polynomial time?

- SAT
- 3-SAT
- CLIQUE
- IS
- VC
- Ex-Cov
- SUBSET SUM
- PARTITION?
- HC
- TSP

Pseudo-polynomial time

Definition

A decision problem X is solvable in pseudo-polynomial time,
if there exists an algorithm that solves instances I of X
in time polynomially bounded in size(I) and number(I).

Observation: number(I) is only relevant for problems that involve numbers (distances, costs, weights, lengths, penalties, profits, time intervals, etc)

Which of the decision problems we studied so far is solvable in pseudo-polynomial time?

- SAT
- 3-SAT
- CLIQUE
- IS
- VC
- Ex-Cov
- SUBSET SUM
- PARTITION (exercise)
- HC
- TSP

Strong NP-hardness

Definition

A decision problem X is strongly NP-hard,
if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$
such that
restriction of X to instances I with $\text{number(I)} \leq p(\text{size(I)})$ is NP-hard.

Strong NP-hardness

Definition

A decision problem X is strongly NP-hard,
if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$
such that
restriction of X to instances I with number(I) $\leq p(\text{size(I)})$ is NP-hard.

- SAT, CLIQUE, IS, VC, HC, TSP are strongly NP-hard
- unary NP-hard = strongly NP-hard
- weak NP-hard = NP-hard, but may be solvable in pseudo-polynomial time

Strong NP-hardness

Definition

A decision problem X is strongly NP-hard,
if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$
such that
restriction of X to instances I with number(I) $\leq p$(size(I)) is NP-hard.

- SAT, CLIQUE, IS, VC, HC, TSP are strongly NP-hard
- unary NP-hard = strongly NP-hard
- weak NP-hard = NP-hard, but may be solvable in pseudo-polynomial time

Theorem

If decision problem X
is strongly NP-hard and solvable in pseudo-polynomial time
then P=NP.

Strong NP-hardness

THREE PARTITION

Instance: positive integers $a_1, \ldots, a_{3n}$ with $\sum_{i=1}^{3n} a_i = nA$
Question: does there exists a partition of the index set $\{1, \ldots, 3n\}$
into n three-element subsets $T_1, \ldots, T_n$ such that
every three-element set T satisfies $\sum_{i \in T} a_i = A$

Strong NP-hardness

THREE PARTITION

Instance: positive integers $a_1, \ldots, a_{3n}$ with $\sum_{i=1}^{3n} a_i = nA$
Question: does there exists a partition of the index set $\{1, \ldots, 3n\}$
into n three-element subsets $T_1, \ldots, T_n$ such that
every three-element set T satisfies $\sum_{i \in T} a_i = A$

Theorem

THREE PARTITION is strongly NP-complete.

Strong NP-hardness

THREE PARTITION

Instance: positive integers $a_1, \dots, a_{3n}$ with $\sum_{i=1}^{3n} a_i = nA$
Question: does there exists a partition of the index set $\{1, \dots, 3n\}$
into n three-element subsets $T_1, \dots, T_n$ such that
every three-element set T satisfies $\sum_{i \in T} a_i = A$

Theorem

THREE PARTITION is strongly NP-complete.

Proof: proof in Garey-Johnson shows that
$SAT \leq_p \leq 3DM \leq_p 4 - PARTITION \leq_p 3 - PARTITION$
Where the instance I constructed in the proof of $3DM \leq_p 4 - PARTITION$ has $number(I) \leq 2^{16}|A|^4$.

Recall:

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

Recall:

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

A decision problem X is NP-complete,
if $X \in NP$ and all problems $Y \in NP$ can be reduced to it.

Recall:

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

A decision problem X is NP-complete,
if $X \in NP$ and all problems $Y \in NP$ can be reduced to it.

Now we define:

Definition

A decision problem X lies in the complexity class coNP,
if the NO-instances of X possess certificates of polynomial length
that can be verified in polynomial time

Recall:

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

A decision problem X is NP-complete,
if $X \in NP$ and all problems $Y \in NP$ can be reduced to it.

Now we define:

Definition

A decision problem X lies in the complexity class coNP,
if the NO-instances of X possess certificates of polynomial length
that can be verified in polynomial time

A decision problem X is $coNP$-complete,
if $X \in coNP$ and all problems $Y \in coNP$ can be reduced to it.

Non-HAMILTONICITY

Instance: an undirected graph $G = (V, E)$
Question: is G not Hamiltonian?

Un-Satisfiability (UNSAT)

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

Question: Is there no truth assignment for X that simultaneously satisfies all clauses in C?

TAUTOLOGY

Instance: a set of logical variables $X := \{x_1, \ldots, x_n\}$ and a formula Φ in CNF over X

Question: are all truth settings for X satisfying for Φ?

Non-HAMILTONICITY

Instance: an undirected graph $G = (V, E)$
Question: is G not Hamiltonian?

Un-Satisfiability (UNSAT)

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

Question: Is there no truth assignment for X that simultaneously satisfies all clauses in C?

TAUTOLOGY

Instance: a set of logical variables $X := \{x_1, \ldots, x_n\}$ and a formula Φ in CNF over X

Question: are all truth settings for X satisfying for Φ?

Theorem

Non-HAMILTONICITY, UNSAT and TAUTOLOGY are coNP-complete.

Non-HAMILTONICITY

Instance: an undirected graph $G = (V, E)$
Question: is G not Hamiltonian?

Un-Satisfiability (UNSAT)

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

Question: Is there no truth assignment for X that simultaneously satisfies all clauses in C?

TAUTOLOGY

Instance: a set of logical variables $X := \{x_1, \ldots, x_n\}$ and a formula Φ in CNF over X

Question: are all truth settings for X satisfying for Φ?

Lemma

If X is NP-complete, $\bar{X}$ is coNP-complete.

$\Rightarrow$ NP-completeness of Non-HAMILTONICITY & UNSAT

Excursion: Logical formulas

Let X be a set of logical *variables*.

- *Truth assignment*: $t : X \to \{true, false\}$
- *Literals:* We call x and $\neg x$ literals corresponding to variable $x \in X$. $t(\neg x) = true$ $\Leftrightarrow t(x) = false$
- *(disjunctive) clause* over X: disjunction of literals, e.g., $(x_1 \vee \neg x_2 \vee \ldots \vee x_k)$.

Excursion: Logical formulas

Let X be a set of logical *variables*.

- *Truth assignment*: $t : X \to \{true, false\}$
- *Literals:* We call x and $\neg x$ literals corresponding to variable $x \in X$. $t(\neg x) = true \Leftrightarrow t(x) = false$
- *(disjunctive) clause* over X: disjunction of literals, e.g., $(x_1 \vee \neg x_2 \vee \ldots \vee x_k)$.
- *conjunctive clause* over X: conjunction of literals, e.g., $(\neg x_1 \wedge x_2 \wedge \ldots \wedge x_j)$.

Excursion: Logical formulas

Let X be a set of logical *variables*.

- *Truth assignment*: $t : X \to \{true, false\}$
- *Literals:* We call x and $\neg x$ literals corresponding to variable $x \in X$. $t(\neg x) = true \Leftrightarrow t(x) = false$
- *(disjunctive) clause* over X: disjunction of literals, e.g., $(x_1 \vee \neg x_2 \vee \ldots \vee x_k)$.
- *conjunctive clause* over X: conjunction of literals, e.g., $(\neg x_1 \wedge x_2 \wedge \ldots \wedge x_j)$.
- logical formula in X:
  (general) logical expression in variables from X, e.g.,
  $[(x1 \vee \neg x_2) \wedge (x_1 \vee x_3)] \vee \neg(x_1 \vee x_2)$

Excursion: Logical formulas

Let X be a set of logical *variables*.

- *Truth assignment*: $t : X \to \{true, false\}$
- *Literals:* We call x and $\neg x$ literals corresponding to variable $x \in X$. $t(\neg x) = true$ $\Leftrightarrow t(x) = false$
- *(disjunctive) clause* over X: disjunction of literals, e.g., $(x_1 \vee \neg x_2 \vee \ldots \vee x_k)$.
- *conjunctive clause* over X: conjunction of literals, e.g., $(\neg x_1 \wedge x_2 \wedge \ldots \wedge x_j)$.
- logical formula in X:
  (general) logical expression in variables from X, e.g.,
  $[(x1 \vee \neg x_2) \wedge (x_1 \vee x_3)] \vee \neg(x_1 \vee x_2)$
- logical formula in conjunctive normal form (CNF):
  conjunction of disjunctive clauses, e.g. $(x1 \vee \neg x_2) \wedge (x_1 \vee x_3)$
- logical formula in disjunctive normal form (DNF):
  disjunction of conjunctive clauses, e.g. $(x1 \wedge \neg x_2) \vee (x_1 \wedge x_3)$

Excursion: Logical formulas

Let X be a set of logical *variables*.

- *Truth assignment*: $t : X \to \{true, false\}$
- *Literals:* We call x and $\neg x$ literals corresponding to variable $x \in X$. $t(\neg x) = true \Leftrightarrow t(x) = false$
- *(disjunctive) clause* over X: disjunction of literals, e.g., $(x_1 \vee \neg x_2 \vee \ldots \vee x_k)$.
- *conjunctive clause* over X: conjunction of literals, e.g., $(\neg x_1 \wedge x_2 \wedge \ldots \wedge x_j)$.
- logical formula in X:
  (general) logical expression in variables from X, e.g.,
  $[(x1 \vee \neg x_2) \wedge (x_1 \vee x_3)] \vee \neg(x_1 \vee x_2)$
- logical formula in conjunctive normal form (CNF):
  conjunction of disjunctive clauses, e.g. $(x1 \vee \neg x_2) \wedge (x_1 \vee x_3)$
- logical formula in disjunctive normal form (DNF):
  disjunction of conjunctive clauses, e.g. $(x1 \wedge \neg x_2) \vee (x_1 \wedge x_3)$

Satisfiability (SAT) - as we use it / CNF-SAT

Instance: set of logical variables $X := \{x_1, \ldots, x_n\}$, logical formula Φ in CNF
Question: does there exist a truth assignment for X that satisfies Φ?

NP-complete (Cook-Levin)

Excursion: Logical formulas

Let X be a set of logical *variables*.

- *Truth assignment*: $t : X \to \{true, false\}$
- *Literals:* We call x and $\neg x$ literals corresponding to variable $x \in X$. $t(\neg x) = true \Leftrightarrow t(x) = false$
- *(disjunctive) clause* over X: disjunction of literals, e.g., $(x_1 \vee \neg x_2 \vee \ldots \vee x_k)$.
- *conjunctive clause* over X: conjunction of literals, e.g., $(\neg x_1 \wedge x_2 \wedge \ldots \wedge x_j)$.
- logical formula in X:
  (general) logical expression in variables from X, e.g.,
  $[(x1 \vee \neg x_2) \wedge (x_1 \vee x_3)] \vee \neg(x_1 \vee x_2)$
- logical formula in conjunctive normal form (CNF):
  conjunction of disjunctive clauses, e.g. $(x1 \vee \neg x_2) \wedge (x_1 \vee x_3)$
- logical formula in disjunctive normal form (DNF):
  disjunction of conjunctive clauses, e.g. $(x1 \wedge \neg x_2) \vee (x_1 \wedge x_3)$

Satisfiability (SAT) - more general

Instance: set of logical variables $X := \{x_1, \ldots, x_n\}$, (any) logical formula Φ
Question: does there exist a truth assignment for X that satisfies Φ?

NP-complete (in NP & generalization of CNF-SAT)

Excursion: Logical formulas

Let X be a set of logical *variables*.

- *Truth assignment*: $t : X \to \{true, false\}$
- *Literals:* We call x and $\neg x$ literals corresponding to variable $x \in X$. $t(\neg x) = true \Leftrightarrow t(x) = false$
- *(disjunctive) clause* over X: disjunction of literals, e.g., $(x_1 \vee \neg x_2 \vee \ldots \vee x_k)$.
- *conjunctive clause* over X: conjunction of literals, e.g., $(\neg x_1 \wedge x_2 \wedge \ldots \wedge x_j)$.
- logical formula in X:
  (general) logical expression in variables from X, e.g.,
  $[(x1 \vee \neg x_2) \wedge (x_1 \vee x_3)] \vee \neg(x_1 \vee x_2)$
- logical formula in conjunctive normal form (CNF):
  conjunction of disjunctive clauses, e.g. $(x1 \vee \neg x_2) \wedge (x_1 \vee x_3)$
- logical formula in disjunctive normal form (DNF):
  disjunction of conjunctive clauses, e.g. $(x1 \wedge \neg x_2) \vee (x_1 \wedge x_3)$

DNF-SAT

Instance: set of logical variables $X := \{x_1, \ldots, x_n\}$, logical formula Φ in DNF
Question: does there exist a truth assignment for X that satisfies Φ?

In P.

Excursion: Logical formulas

Can we transform any logical formula into CNF?

- commutative, associative, distributive: $(x_1 \wedge x_2) \vee x_3 = (x_1 \vee x_3) \wedge (x_2 \vee x_3)$
- $\neg\neg l = l$
- $\neg(l_1 \wedge l_1) = \neg l_1 \vee \neg l_2$ (De Morgan's law)
- $\neg(x \vee y) = \neg x \wedge \neg y$ (De Morgan's law)

Excursion: Logical formulas

Can we transform any logical formula into CNF?

- commutative, associative, distributive: $(x_1 \wedge x_2) \vee x_3 = (x_1 \vee x_3) \wedge (x_2 \vee x_3)$
- $\neg\neg l = l$
- $\neg(l_1 \wedge l_1) = \neg l_1 \vee \neg l_2$ (De Morgan's law)
- $\neg(x \vee y) = \neg x \wedge \neg y$ (De Morgan's law)

Excursion: Logical formulas

Can we transform any logical formula into CNF?

- commutative, associative, distributive: $(x_1 \wedge x_2) \vee x_3 = (x_1 \vee x_3) \wedge (x_2 \vee x_3)$
- $\neg\neg l = l$
- $\neg(l_1 \wedge l_1) = \neg l_1 \vee \neg l_2$ (De Morgan's law)
- $\neg(x \vee y) = \neg x \wedge \neg y$ (De Morgan's law)

Example:

$$\begin{aligned} & [(x1 \wedge \neg x_2) \vee (x_1 \wedge x_3)] \wedge \neg(x_1 \vee x_2) \\ = & [(x1 \wedge \neg x_2) \vee x_1] \wedge [(x1 \wedge \neg x_2) \vee x_3] \wedge (\neg x_1 \wedge \neg x_2) \\ = & x_1 \wedge (x1 \vee x_3) \wedge (\neg x_2 \vee x_3) \wedge \neg x_1 \wedge \neg x_2 \end{aligned}$$

Excursion: Logical formulas

Can we transform any logical formula into CNF?

- commutative, associative, distributive: $(x_1 \wedge x_2) \vee x_3 = (x_1 \vee x_3) \wedge (x_2 \vee x_3)$
- $\neg\neg l = l$
- $\neg(l_1 \wedge l_1) = \neg l_1 \vee \neg l_2$ (De Morgan's law)
- $\neg(x \vee y) = \neg x \wedge \neg y$ (De Morgan's law)

Example 2:

$$\begin{aligned}\Phi &= (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n) \\ = &(x_1 \vee x_2 \vee \ldots \vee x_n) \wedge (y_1 \vee x_2 \vee \ldots \vee x_n) \\ &\wedge(x_1 \vee y_2 \vee \ldots \vee x_n) \wedge (y_1 \vee y_2 \vee \ldots \vee x_n) \\ &\wedge \ldots \wedge (y_1 \vee y_2 \vee \ldots \vee y_n)\end{aligned}$$

Excursion: Logical formulas

Can we transform any logical formula into CNF?

- commutative, associative, distributive: $(x_1 \wedge x_2) \vee x_3 = (x_1 \vee x_3) \wedge (x_2 \vee x_3)$
- $\neg\neg l = l$
- $\neg(l_1 \wedge l_1) = \neg l_1 \vee \neg l_2$ (De Morgan's law)
- $\neg(x \vee y) = \neg x \wedge \neg y$ (De Morgan's law)

Example 2:

$$\begin{aligned} \Phi &= (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n) \\ &= (x_1 \vee x_2 \vee \ldots \vee x_n) \wedge (y_1 \vee x_2 \vee \ldots \vee x_n) \\ &\wedge (x_1 \vee y_2 \vee \ldots \vee x_n) \wedge (y_1 \vee y_2 \vee \ldots \vee x_n) \\ &\wedge \ldots \wedge (y_1 \vee y_2 \vee \ldots \vee y_n) \end{aligned}$$

Naive approach leads to formula of exponential length here!

Excursion: Logical formulas

Can we transform any logical formula into CNF?

- commutative, associative, distributive: $(x_1 \wedge x_2) \vee x_3 = (x_1 \vee x_3) \wedge (x_2 \vee x_3)$
- $\neg\neg l = l$
- $\neg(l_1 \wedge l_1) = \neg l_1 \vee \neg l_2$ (De Morgan's law)
- $\neg(x \vee y) = \neg x \wedge \neg y$ (De Morgan's law)

Example 2:

$$\begin{aligned} \Phi &= (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n) \\ &= (x_1 \vee x_2 \vee \ldots \vee x_n) \wedge (y_1 \vee x_2 \vee \ldots \vee x_n) \\ &\wedge (x_1 \vee y_2 \vee \ldots \vee x_n) \wedge (y_1 \vee y_2 \vee \ldots \vee x_n) \\ &\wedge \ldots \wedge (y_1 \vee y_2 \vee \ldots \vee y_n) \end{aligned}$$

But: 'more general SAT' is in NP, and CNF-SAT is NP-complete:

Excursion: Logical formulas

Can we transform any logical formula into CNF?

- commutative, associative, distributive: $(x_1 \wedge x_2) \vee x_3 = (x_1 \vee x_3) \wedge (x_2 \vee x_3)$
- $\neg\neg l = l$
- $\neg(l_1 \wedge l_1) = \neg l_1 \vee \neg l_2$ (De Morgan's law)
- $\neg(x \vee y) = \neg x \wedge \neg y$ (De Morgan's law)

Example 2:

$$\begin{aligned} \Phi &= (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n) \\ &= (x_1 \vee x_2 \vee \ldots \vee x_n) \wedge (y_1 \vee x_2 \vee \ldots \vee x_n) \\ &\wedge (x_1 \vee y_2 \vee \ldots \vee x_n) \wedge (y_1 \vee y_2 \vee \ldots \vee x_n) \\ &\wedge \ldots \wedge (y_1 \vee y_2 \vee \ldots \vee y_n) \end{aligned}$$

But: 'more general SAT' is in NP, and CNF-SAT is NP-complete: there **must be** a way of writing Φ as a CNF formula!

Excursion: Logical formulas

Can we transform any logical formula into CNF?

- commutative, associative, distributive: $(x_1 \wedge x_2) \vee x_3 = (x_1 \vee x_3) \wedge (x_2 \vee x_3)$
- $\neg\neg l = l$
- $\neg(l_1 \wedge l_1) = \neg l_1 \vee \neg l_2$ (De Morgan's law)
- $\neg(x \vee y) = \neg x \wedge \neg y$ (De Morgan's law)

Example 2:

$$\begin{aligned} \Phi &= (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n) \\ &= (x_1 \vee x_2 \vee \ldots \vee x_n) \wedge (y_1 \vee x_2 \vee \ldots \vee x_n) \\ &\wedge (x_1 \vee y_2 \vee \ldots \vee x_n) \wedge (y_1 \vee y_2 \vee \ldots \vee x_n) \\ &\wedge \ldots \wedge (y_1 \vee y_2 \vee \ldots \vee y_n) \end{aligned}$$

But: 'more general SAT' is in NP, and CNF-SAT is NP-complete:
there **must be** a way of writing Φ as a CNF formula!
Idea: Write $(x_i \wedge y_i) = (\neg x_i \vee \neg y_i \vee z_i) \wedge (x_i \vee \neg z_i) \wedge (y_i \vee \neg z_i)$
We then obtain a clause Φ' in $X' = X \cup \{z_1, \ldots, z_n\}$ of polynomial length.

Excursion: Logical formulas

Can we transform any logical formula into CNF?

- commutative, associative, distributive: $(x_1 \wedge x_2) \vee x_3 = (x_1 \vee x_3) \wedge (x_2 \vee x_3)$
- $\neg\neg l = l$
- $\neg(l_1 \wedge l_1) = \neg l_1 \vee \neg l_2$ (De Morgan's law)
- $\neg(x \vee y) = \neg x \wedge \neg y$ (De Morgan's law)

Example 2:

$$\begin{aligned}\Phi &= (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n)\\ &= (x_1 \vee x_2 \vee \ldots \vee x_n) \wedge (y_1 \vee x_2 \vee \ldots \vee x_n)\\ &\wedge (x_1 \vee y_2 \vee \ldots \vee x_n) \wedge (y_1 \vee y_2 \vee \ldots \vee x_n)\\ &\qquad \wedge \ldots \wedge (y_1 \vee y_2 \vee \ldots \vee y_n)\end{aligned}$$

But: 'more general SAT' is in NP, and CNF-SAT is NP-complete: there **must be** a way of writing Φ as a CNF formula!
Idea: Write $(x_i \wedge y_i) = (\neg x_i \vee \neg y_i \vee z_i) \wedge (x_i \vee \neg z_i) \wedge (y_i \vee \neg z_i)$
We then obtain a clause Φ' in $X' = X \cup \{z_1, \ldots, z_n\}$ of polynomial length.
For general approach to transform logical formulas to CNF, see, e.g., wikipedia: Tseytin transformation

Back to TAUTOLOGY

TAUTOLOGY

Instance: a set of logical variables $X := \{x_1, \ldots, x_n\}$ and DNF-formula Φ over X
Question: are all truth settings for X satisfying assignments for C?

Theorem

TAUTOLOGY is coNP-complete.

Proof:

Back to TAUTOLOGY

TAUTOLOGY

Instance: a set of logical variables $X := \{x_1, \ldots, x_n\}$ and DNF-formula Φ over X
Question: are all truth settings for X satisfying assignments for C?

Theorem

TAUTOLOGY is coNP-complete.

Proof: We show: CNF-SAT $\leq_p \overline{TAUTOLOGY}$.

Back to TAUTOLOGY

TAUTOLOGY

Instance: a set of logical variables $X := \{x_1, \ldots, x_n\}$ and DNF-formula Φ over X
Question: are all truth settings for X satisfying assignments for C?

Theorem

TAUTOLOGY is coNP-complete.

Proof: We show: CNF-SAT $\leq_p$ $\overline{TAUTOLOGY}$.

$\overline{TAUTOLOGY}$

Instance: a set of logical variables $X := \{x_1, \ldots, x_n\}$ and a DNF-formula Φ over X
Question: is there a truth setting for X satisfying for $\neg\Phi$?

Back to TAUTOLOGY

TAUTOLOGY

Instance: a set of logical variables $X := \{x_1, \ldots, x_n\}$ and DNF-formula Φ over X
Question: are all truth settings for X satisfying assignments for C?

Theorem

TAUTOLOGY is coNP-complete.

Proof: We show: CNF-SAT $\leq_p \overline{TAUTOLOGY}$.

$\overline{TAUTOLOGY}$

Instance: a set of logical variables $X := \{x_1, \ldots, x_n\}$ and a DNF-formula Φ over X
Question: is there a truth setting for X satisfying for $\neg\Phi$?

Let X' be a set of logical variables and Φ' a CNF-formula on X.

Back to TAUTOLOGY

TAUTOLOGY

Instance: a set of logical variables $X := \{x_1, \ldots, x_n\}$ and DNF-formula Φ over X
Question: are all truth settings for X satisfying assignments for C?

Theorem

TAUTOLOGY is coNP-complete.

Proof: We show: CNF-SAT $\leq_p \overline{TAUTOLOGY}$.

$\overline{TAUTOLOGY}$

Instance: a set of logical variables $X := \{x_1, \ldots, x_n\}$ and a DNF-formula Φ over X
Question: is there a truth setting for X satisfying for $\neg\Phi$?

Let X' be a set of logical variables and Φ' a CNF-formula on X.
Then $\Phi := \neg\Phi'$ is a DNF-formula on X (De-Morgan's law).

Back to TAUTOLOGY

TAUTOLOGY

Instance: a set of logical variables $X := \{x_1, \ldots, x_n\}$ and DNF-formula Φ over X
Question: are all truth settings for X satisfying assignments for C?

Theorem

TAUTOLOGY is coNP-complete.

Proof: We show: CNF-SAT $\leq_p \overline{TAUTOLOGY}$.

$\overline{TAUTOLOGY}$

Instance: a set of logical variables $X := \{x_1, \ldots, x_n\}$ and a DNF-formula Φ over X
Question: is there a truth setting for X satisfying for $\neg\Phi$?

Let X' be a set of logical variables and Φ' a CNF-formula on X.
Then $\Phi := \neg\Phi'$ is a DNF-formula on X (De-Morgan's law).
Thus (X, Φ) is an instance of $\overline{TAUTOLOGY}$

Back to TAUTOLOGY

TAUTOLOGY

Instance: a set of logical variables $X := \{x_1, \ldots, x_n\}$ and DNF-formula Φ over X
Question: are all truth settings for X satisfying assignments for C?

Theorem

TAUTOLOGY is coNP-complete.

Proof: We show: CNF-SAT $\leq_p \overline{TAUTOLOGY}$.

$\overline{TAUTOLOGY}$

Instance: a set of logical variables $X := \{x_1, \ldots, x_n\}$ and a DNF-formula Φ over X
Question: is there a truth setting for X satisfying for $\neg\Phi$?

Let X' be a set of logical variables and Φ' a CNF-formula on X.
Then $\Phi := \neg\Phi'$ is a DNF-formula on X (De-Morgan's law).
Thus (X, Φ) is an instance of $\overline{TAUTOLOGY}$ which is satisfiable if and only in (X', Φ') is satisfiable.

NP versus coNP (3)

Problems in $NP \cap coNP$ have

- good certificates for YES-instances
- good certificates for NO-instances

NP versus coNP (3)

Problems in $NP \cap coNP$ have
- good certificates for YES-instances
- good certificates for NO-instances

Example

Linear Programming (LP):
Instance: a matrix A; vectors c and b; a bound t
Question: does there exist a real vector x with $Ax \leq b$ and $cx \leq t$?

- LP lies in NP
- LP lies in coNP (LP-duality)

- MaxFlow in NP
- MaxFlow in coNP

The Soviet railway system problem

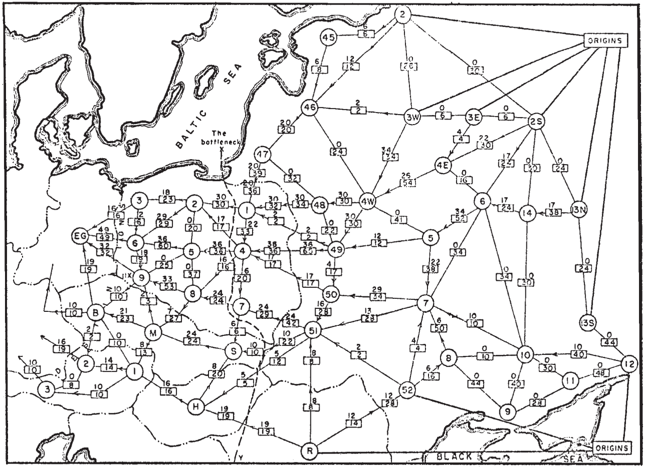

Fig. 2. From Harris and Ross [11]: Schematic diagram of the railway network of the Western Soviet Union and Eastern European countries, with a maximum flow of value 163,000 tons from Russia to Eastern Europe, and a cut of capacity 163,000 tons indicated as "The bottleneck"

NP versus coNP (4)

- FACT: If $P = coNP$ then $P = NP$ (P closed under complementation)

NP versus coNP (4)

- FACT: If $P = coNP$ then $P = NP$ (P closed under complementation)
- FACT: $P \subseteq NP \cap coNP$

NP versus coNP (4)

- FACT: If $P = coNP$ then $P = NP$ (P closed under complementation)
- FACT: $P \subseteq NP \cap coNP$
- Some people think that $P \neq NP \cap coNP$
- Some people think that $P = NP \cap coNP$

- Most people think that $NP \neq coNP$

NP versus coNP (4)

- FACT: If $P = coNP$ then $P = NP$ (P closed under complementation)
- FACT: $P \subseteq NP \cap coNP$
- Some people think that $P \neq NP \cap coNP$
- Some people think that $P = NP \cap coNP$

- Most people think that $NP \neq coNP$

Theorem

If coNP contains some NP-complete problem X, then NP=coNP.

NP versus coNP (4)

- FACT: If $P = coNP$ then $P = NP$ (P closed under complementation)
- FACT: $P \subseteq NP \cap coNP$
- Some people think that $P \neq NP \cap coNP$
- Some people think that $P = NP \cap coNP$

- Most people think that $NP \neq coNP$

Theorem

If coNP contains some NP-complete problem X, then NP=coNP.

Hence:

- X being NP-complete is indication for $X \notin coNP$

NP versus coNP (4)

- FACT: If $P = coNP$ then $P = NP$ (P closed under complementation)
- FACT: $P \subseteq NP \cap coNP$
- Some people think that $P \neq NP \cap coNP$
- Some people think that $P = NP \cap coNP$

- Most people think that $NP \neq coNP$

Theorem

If coNP contains some NP-complete problem X, then NP=coNP.

Hence:

- X being NP-complete is indication for $X \notin coNP$
- X being coNP-complete is indication for $X \notin NP$

NP versus coNP (4)

- FACT: If $P = coNP$ then $P = NP$ (P closed under complementation)
- FACT: $P \subseteq NP \cap coNP$
- Some people think that $P \neq NP \cap coNP$
- Some people think that $P = NP \cap coNP$

- Most people think that $NP \neq coNP$

Theorem

If coNP contains some NP-complete problem X, then NP=coNP.

Hence:

- X being NP-complete is indication for $X \notin coNP$
- X being coNP-complete is indication for $X \notin NP$
- $X \in NP \cap coNP$ is indication for X not being (co)NP-complete

NP versus coNP (5)

Example

Factoring (LP):
Instance: integers y, l, u (given in binary).
Question: Is there an integer x that divides y and satisfies $l \leq x \leq u$?

in P? strongly NP-complete? weakly NP-complete? in NP? in co-NP?

Note: basic arithmetic (division, multiplication) is in polynomial time. Primality testing is in P.

NP versus coNP (5)

Example

Factoring (LP):
Instance: integers y, l, u (given in binary).
Question: Is there an integer x that divides y and satisfies $l \leq x \leq u$?

in P? strongly NP-complete? weakly NP-complete? in NP? in co-NP?

Note: basic arithmetic (division, multiplication) is in polynomial time. Primality testing is in P.

Many cryptographic protocols are based on the difficulty of factoring large composite integers - an algorithm that efficiently factors an arbitrary integer would render these insecure.

An unsolvable decision problem (not discussed)

Problem: CheckTermination (also called ‘Halting Problem’)

Input: two text pieces `text1` and `text2`

An unsolvable decision problem (not discussed)

Problem: CheckTermination (also called ‘Halting Problem’)

Input: two text pieces `text1` and `text2`

Question: does the C++ program listed in `text1`
terminate on the input in `text2`?

An unsolvable decision problem (not discussed)

Problem: CheckTermination (also called 'Halting Problem')

Input: two text pieces `text1` and `text2`

Question: does the C++ program listed in `text1`
terminate on the input in `text2`?

- Suppose there exists an algorithm for CheckTermination

An unsolvable decision problem (not discussed)

Problem: CheckTermination (also called 'Halting Problem')

Input: two text pieces `text1` and `text2`

Question: does the C++ program listed in `text1` terminate on the input in `text2`?

- Suppose there exists an algorithm for CheckTermination
- Then there is a C++ program `CT(text1,text2)` implementing this

An unsolvable decision problem (not discussed)

Problem: CheckTermination (also called 'Halting Problem')

Input: two text pieces `text1` and `text2`

Question: does the C++ program listed in `text1` terminate on the input in `text2`?

- Suppose there exists an algorithm for CheckTermination
- Then there is a C++ program `CT(text1,text2)` implementing this

- We construct a new C++ program `wrong` that takes input `text3`

An unsolvable decision problem (not discussed)

Problem: CheckTermination (also called 'Halting Problem')

Input: two text pieces `text1` and `text2`

Question: does the C++ program listed in `text1` terminate on the input in `text2`?

- Suppose there exists an algorithm for CheckTermination
- Then there is a C++ program `CT(text1,text2)` implementing this

- We construct a new C++ program `wrong` that takes input `text3`
- First, `wrong` checks whether the C++ program listed in `text3`

An unsolvable decision problem (not discussed)

Problem: CheckTermination (also called 'Halting Problem')

Input: two text pieces `text1` and `text2`

Question: does the C++ program listed in `text1` terminate on the input in `text2`?

- Suppose there exists an algorithm for CheckTermination
- Then there is a C++ program `CT(text1,text2)` implementing this

- We construct a new C++ program `wrong` that takes input `text3`
- First, `wrong` checks whether the C++ program listed in `text3` terminates on the input in `text3` using `CT(text3,text3)`

An unsolvable decision problem (not discussed)

Problem: CheckTermination (also called 'Halting Problem')

Input: two text pieces `text1` and `text2`

Question: does the C++ program listed in `text1` terminate on the input in `text2`?

- Suppose there exists an algorithm for CheckTermination
- Then there is a C++ program `CT(text1,text2)` implementing this

- We construct a new C++ program `wrong` that takes input `text3`
- First, wrong checks whether the C++ program listed in `text3` terminates on the input in `text3` using `CT(text3,text3)`
- If `text3` does terminate, then wrong(text3) goes into an infinite loop

An unsolvable decision problem (not discussed)

Problem: CheckTermination (also called 'Halting Problem')

Input: two text pieces `text1` and `text2`

Question: does the C++ program listed in `text1` terminate on the input in `text2`?

- Suppose there exists an algorithm for CheckTermination
- Then there is a C++ program `CT(text1,text2)` implementing this

- We construct a new C++ program `wrong` that takes input `text3`
- First, wrong checks whether the C++ program listed in `text3` terminates on the input in `text3` using `CT(text3,text3)`
- If `text3` does terminate, then wrong(text3) goes into an infinite loop
- If `text3` does not terminate, then wrong(text3) stops

An unsolvable decision problem (not discussed)

Problem: CheckTermination (also called 'Halting Problem')

Input: two text pieces `text1` and `text2`

Question: does the C++ program listed in `text1` terminate on the input in `text2`?

- Suppose there exists an algorithm for CheckTermination
- Then there is a C++ program `CT(text1,text2)` implementing this

- We construct a new C++ program `wrong` that takes input `text3`
- First, `wrong` checks whether the C++ program listed in `text3` terminates on the input in `text3` using `CT(text3,text3)`
- If `text3` does terminate, then wrong(text3) goes into an infinite loop
- If `text3` does not terminate, then wrong(text3) stops

- What does CT(text4,text4) do if `text4` is the C++ code of `wrong`???

An unsolvable decision problem (not discussed)

Problem: CheckTermination (also called 'Halting Problem')

Input: two text pieces `text1` and `text2`

Question: does the C++ program listed in `text1` terminate on the input in `text2`?

- Suppose there exists an algorithm for CheckTermination
- Then there is a C++ program `CT(text1,text2)` implementing this

- We construct a new C++ program `wrong` that takes input `text3`
- First, `wrong` checks whether the C++ program listed in `text3` terminates on the input in `text3` using `CT(text3,text3)`
- If `text3` does terminate, then wrong(text3) goes into an infinite loop
- If `text3` does not terminate, then wrong(text3) stops

- What does CT(text4,text4) do if `text4` is the C++ code of `wrong`???
- Conclusion: There is no algorithm for CheckTermination
- Technique is called *diagonalization*. Also used to show there are decision problems that can be solved in $O(n^c)$, but not in $O(n^{c-1})$ time

Recommended Reading

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

- Chapter 26 (Maximum flow)
- Chapter 29 (Linear Programming, duality)