Computational Complexity; slides 5, HT 2019
nondeterminism

Prof. Paul W. Goldberg (Dept. of Computer Science, University of Oxford)

HT 2019

Nondeterministic Turing Machines

Definition.

A non-deterministic 1-tape Turing machine is a 6-tuple $(Q, \Sigma, \Gamma, \Delta, q_0, F)$ where

- Q is a finite set of states
- Σ is a finite alphabet of symbols
- $\Gamma \supseteq \Sigma \cup \{\Box\}$ is a finite alphabet of symbols
- $\Delta \subseteq (Q \setminus F) \times \Gamma \times Q \times \Gamma \times \{-1, 0, 1\}$ transition **relation**
- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is a set of final states

As before, we assume $\Sigma := \{0, 1\}$ and $\Gamma := \Sigma \cup \{\Box\}$.

The computation of a non-deterministic Turing machine $\mathcal{M} = (Q, \Sigma, \Gamma, \Delta, q_0, F)$ on input w is a "computation tree" analogy with NFA, (N)PDA

Non-Deterministic Turing Acceptor

Non-deterministic Turing acceptor: $(Q, \Sigma, \Gamma, \Delta, q_0, F_a, F_r)$

Computation path:

Any path from the start configuration to a stop configuration in the configuration tree.

accepting path: the stop configuration is in an accepting state. (also called an accepting run)

rejecting path otherwise

Language accepted by an NTM $\mathcal{M}$:

$\mathcal{L}(\mathcal{M}) := \{w \in \Sigma^* :$ there **exists** an accepting path of $\mathcal{M}$ on $w\}$

Simulation

The following models can all be (poly-time) simulated by 1-tape NTMs:

- k-tape non-deterministic Turing machines
- Two way infinite multi-tape NTMs
- Non-det. Random access Turing machines
- ...

All these simulations run in polynomial time.
Can simulate with deterministic TM, but not in poly-time

From P to NP

NP: languages accepted by NTM in polynomially-many steps; equivalently, problems whose yes-instances are accepted by (poly-time) NTM

- e.g. 3-Sat and 3-Colourability, TSP, SAT, etc
- No polynomial time algorithms for these problems are known
- but are in NP

"Guess and test": generic NP algorithm. As for P, no need to think in terms of TMs

Non-Deterministic Complexity Classes

Important Non-Deterministic Complexity Classes:

- Time classes:
  - $\mathrm{NP}\textsc{time}$ a.k.a. $\mathrm{NP} := \bigcup_{d\in\mathbb{N}} \mathrm{N}\textsc{Time}(n^d)$
  - $\mathrm{NE}\textsc{xptime} := \bigcup_{d\in\mathbb{N}} \mathrm{N}\textsc{Time}(2^{n^d})$
- Space classes:
  - $\mathrm{NL}\textsc{ogspace} := \bigcup_{d\in\mathbb{N}} \mathrm{N}\textsc{space}(d \log n)$
  - $\mathrm{NP}\textsc{space} := \bigcup_{d\in\mathbb{N}} \mathrm{N}\textsc{space}(n^d)$
  - $\mathrm{NE}\textsc{xpspace} := \bigcup_{d\in\mathbb{N}} \mathrm{N}\textsc{space}(2^{n^d})$

where $\mathrm{N}\textsc{Time}(\mathrm{T})$ (etc.) means what you think it means. Note that all accepting/non-accepting computations of a $\mathrm{N}\textsc{Time}(\mathrm{T})$ TM should have length at most T

Certificates

Every yes-instance of such problems has a short and easily checkable certificate that proves it is a yes-instance.

- SAT – a satisfying assignment
- k-COLOURABILITY – a k-colouring
- HAMILTONIAN CIRCUIT – a Hamiltonian circuit
- TRAVELLING SALESMAN (version with a "distance budget") – a round trip (i.e. permutation)

Verifiers

Definition.

1. A Turing acceptor $\mathcal{M}$ which halts on all inputs is called a verifier for language $\mathcal{L}$ if

   $$\mathcal{L} = \{w : \mathcal{M} \text{ accepts } \langle w, c\rangle \text{ for some string } c\}$$

   The string c is called a certificate (or witness) for w.

2. A polynomial time verifier for $\mathcal{L}$ is a polynomially time bounded Turing acceptor $\mathcal{M}$ such that

   $$\mathcal{L} = \{w : \mathcal{M} \text{ accepts } \langle w, c\rangle \text{ for some string } c \text{ with } |c| \leq p(|w|)\}$$

   for some fixed polynomial $p(n)$.

All problems for the previous slide have verifiers that run in polynomial time.

Equivalent def of NP

The class of languages that have polynomial-time verifiers

Examples.

- SATISFIABILITY is in NP

  For any formula that can be satisfied, the satisfying assignment can be used as a certificate.

  It can be verified in polynomial time that the assignment satisfies the formula.

- k-COLOURABILITY is in NP

  For any graph that can be coloured, the colouring can be used as a certificate.

  It can be verified in polynomial time that the colouring is a proper colouring.

More Examples of Problems in NP

COMPOSITE (non-prime) NUMBER

Input: A positive integer $n > 1$

Problem: Are there integers $u, v > 1$ such that $u \cdot v = n$?

SUBSET SUM

Input: A collection of positive integers $S := \{a_1, \ldots, a_k\}$ and a target integer t.

Problem: Is there a subset $T \subseteq S$ such that $\sum_{a_i \in T} a_i = t$?

A Problem (probably) not in NP

No Hamiltonian Cycle

Input: A graph G

Problem: Is it true that G has no Hamiltonian cycle?

Note. Whereas it is easy to certify that a graph has a Hamiltonian cycle, there does not seem to be a certificate that it has not.

But we may just not be clever enough to find one.

co-NP

co-NP problem: complement of an NP problem
In a co-NP problem, no-instances have (concise) certificates
Believed that NP is <u>not</u> equal to co-NP

The following result justifies guess and test approach to establishing membership of NP:

NP as languages having concise certificates

Theorem. NP as just defined, is languages having concise certificates

Proof. Suppose $\mathcal{L} \in$ NP.

Hence, there is an NTM $\mathcal{M}$ such that

$$w \in \mathcal{L} \iff \text{there is an accepting run of } \mathcal{M} \text{ of length } \leq n^k$$

for some k. This path can be used as a certificate for w

(Clearly, a DTM can check in polynomial time that a candidate

for a certificate is a valid accepting computation path.)

NP as languages having concise certificates

Theorem. NP as just defined, is languages having concise certificates

Proof. Suppose $\mathcal{L} \in$ NP.

Hence, there is an NTM $\mathcal{M}$ such that

$$w \in \mathcal{L} \iff \text{there is an accepting run of } \mathcal{M} \text{ of length } \leq n^k$$

for some k. This path can be used as a certificate for w

(Clearly, a DTM can check in polynomial time that a candidate
for a certificate is a valid accepting computation path.)

Conversely: If $\mathcal{L}$ has a polynomial-time verifier $\mathcal{M}$, say of length at most n^k,

then we can construct an NTM $\mathcal{M}^*$ deciding $\mathcal{L}$ as follows:

1. $\mathcal{M}^*$ guesses a string of length $\leq n^k$
2. $\mathcal{M}^*$ checks in deterministic polynomial-time if this is a certificate.

Deterministic vs. Non-Deterministic Time

Clearly, P$\subseteq$NP.

Question: The question $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ is among the most important open problems in computer science and mathematics.

- It is equivalent to determining whether or not the existence of a short solution guarantees an efficient way of finding it.
- Most people are convinced that $\mathsf{P} \neq \mathsf{NP}$

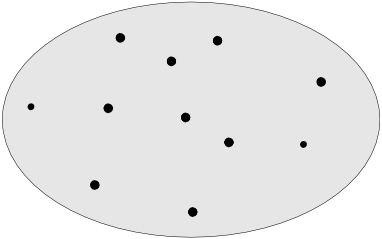
  But after 30 years of effort there is still no proof.
- Resolving the question (either way) would win a prize of $1 million – see `http://www.claymath.org/millennium-problems/`

poly-time reductions amongst NP problems

- Some problems in NP will have polynomial-time many-one reductions to others.
- This partitions the complexity class into equivalence classes via polynomial-time reductions:

  Each class contains problems that are pairwise inter-reducible.
- Equivalence classes are partially ordered by the reduction relation.
- Problems in the maximal class are called complete

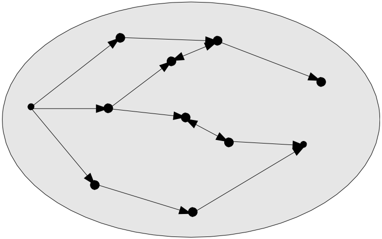
NP:



- Some problems in NP will have polynomial-time many-one reductions to others.
- This partitions the complexity class into equivalence classes via polynomial-time reductions:

  Each class contains problems that are pairwise inter-reducible.
- Equivalence classes are partially ordered by the reduction relation.
- Problems in the maximal class are called complete

NP:

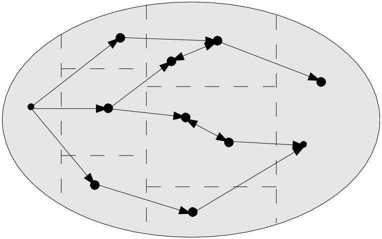


poly-time reductions amongst NP problems

- Some problems in NP will have polynomial-time many-one reductions to others.
- This partitions the complexity class into equivalence classes via polynomial-time reductions:

  Each class contains problems that are pairwise inter-reducible.
- Equivalence classes are partially ordered by the reduction relation.
- Problems in the maximal class are called complete

NP:



NP-Hardness and NP-Completeness

Definition.

1. A language $\mathcal{H}$ is NP-hard, if $\mathcal{L} \leq_p \mathcal{H}$ for every language $\mathcal{L} \in \mathrm{NP}$.
2. A language $\mathcal{C}$ is NP-complete, if $\mathcal{C}$ is NP-hard and $\mathcal{C} \in \mathrm{NP}$.

NP-Completeness:

- NP-complete problems are the hardest problems in NP.
- They are all equally difficult – an efficient solution to one would solve them all.

Lemma. If $\mathcal{L}$ is NP-hard and $\mathcal{L} \leq_p \mathcal{L}'$, then $\mathcal{L}'$ is NP-hard as well.

Proving NP-Completeness

NP-completeness: To show that $\mathcal{L}$ is NP-complete, we must show that every language in NPcan be reduced to $\mathcal{L}$ in polynomial time.

However: Once we have one NP-complete language $\mathcal{C}$, we can show that another language $\mathcal{L}'$ is NP-complete just by showing that

- $\mathcal{C} \leq_p \mathcal{L}'$
- $\mathcal{L}' \in \mathrm{NP}$

Hence: The problem is to find the first one ...

2 problems involving propositional logic

1. Given a formula φ on variables $x_1, \ldots x_n$, and values for those variables, derive the value of φ — easy!
2. Search for values for $x_1, \ldots, x_n$ that make φ evaluate to TRUE — naive algorithm is exponential: 2^n vectors of truth assignments.

Cook's Theorem (1971) or, Cook-Levin Theorem

The second of these, called SAT, is **NP**-complete.

Stephen Cook, Leonid Levin

The challenge of solving boolean formulae

There's a HUGE theory literature on the computational challenge of solving various classes of syntactically restricted classes of boolean formulae, also circuits.
Likewise much has been written about their relative *expressive power*
SAT-solver: software that solves input instances of SAT — OK, so it's worst-case exponential, but aim to solve instances that arise in practice.

- "truth table" approach: clearly exponential
- DPLL algorithm; resolution: worst-case exponential, often fast in practice

Reducing an **NP** problem to SAT

Goal: fixing non-deterministic TM M, integer k, given w create in poly-time a propositional formula $\textbf{CodesAcceptRun}_M(w)$ that is satisfied by assignments that code an n^k length accepting run of M on w (where $n = |w|$)

Idea: introduce propositional variables

- $HasSymbol_{i,j}(a)$: "at time i, tape has letter a at location j"
- $HasHead_{i,j}(q)$: "at time i, TM is in location j, state q"

We'll assume M has "stay put" transitions for which it can change tape contents; R and L moves don't change tape. Assume also that to accept, M goes to LHS of tape and prints special symbol.

M has a "configuration table"

		Tape space j		
Time i	1	2	$\cdots$	n^k
1	(q_0, w_1)	w_2	$\cdots$	
2	w_1'	(q_1, w_2)		
$\vdots$				
$\vdots$				
n^k				

This corresponds to a run where
$HasSymbol_{1,1}(w_1)$
$HasHead_{1,1}(q_0)$
$HasSymbol_{1,2}(w_2)$
$HasSymbol_{2,1}(w_1')$
$HasSymbol_{2,2}(w_2)$
$HasHead_{2,2}(q_1)$
...are true
(Others, e.g. $HasHead_{1,2}(q_0)$ are false)

Idea: the search for "correct" non-determinstic choices for M shall correspond to search for satisfying assignment for $\mathbf{CodesAcceptRun}_M(w)$.
$\mathbf{CodesAcceptRun}_M(w)$ shall be a conjunction of *clauses*.

Getting started

To write the formula **CodesAcceptRun**$_M(w)$, let's start by writing:

$$HasSymbol_{1,j}(w_j)$$

for each $j = 1, ..., |w|$, where w_j is the j-th letter of input w, also

$$\neg HasSymbol_{1,j}(a)$$

for any a where a is *not* the j-th letter of w.

Similarly

$$HasHead_{1,1}(q_0)$$

says M is in state q_0 at time 1, location 1. Add a bunch of negated "HasHead" variables.

TM head "sanity clauses"

Include the following:

$$HasHead_{i,j}(q) \Rightarrow \neg HasHead_{i,j'}(q')$$

...for all states q, q', for all i, j, j' with $j \neq j'$.

Leftward moving state. If M has transition rule $(q, a) \rightarrow \{(q_1, a, L), (q_2, a, L)\}$ then we write:

$$HasHead_{i,j}(q) \Rightarrow [HasHead_{i+1,j-1}(q_1) \vee HasHead_{i+1,j-1}(q_2)]$$

Write the above for all $i, j \in \{1, 2, 3, \ldots, n^k\}$.

Tape space

Time

	1	$\cdots$	$j-1$	j	$\cdots$	n^k
1						
i			w_2	(q, a)		
$i+1$			(q_1, w_2)	a		
$\vdots$						
n^k						

Moving head clauses: Rightward-moving State or Leftward-moving State

For every rightward or leftward state q, for every a we add the clause:

$$HasSymbol_{i,j}(a) \wedge HasHead_{i,j}(q) \Rightarrow HasSymbol_{i+1,j}(a)$$

Meaning: if the head is at place j at step i and we are in a rightward- or leftward moving state, symbol in place j at step $i+1$ is the same.

Tape space

	1	$\cdots$	j		$\cdots$	n^k
1						
Time i			(q,a)	w_2	$\cdots$	
$i+1$			a	(q_1,w_2)	$\cdots$	
$\vdots$						
n^k						

Moving head clauses: stay-same state

For every stay-and-write state q, if we have transition $(q, w_0) \rightarrow \{(q_1, w_1, Stay), (q_2, w_1, Stay)\}$ then we add:

$$HasSymbol_{i,j}(w_0) \wedge HasHead_{i,j}(q) \Rightarrow HasSymbol_{i+1,j}(w_1)$$

(new symbol is written – use "stay determinism" assumption of M_A here!) And also:

$$HasHead_{i,j}(q) \Rightarrow [HasHead_{i+1,j}(q_1) \vee HasHead_{i+1,j}(q_2)]$$

(head does not move, although state may change)

	1	$\cdots$	j	$\cdots$	n^k
1					
i			(q, w_0)	$\cdots$	
$i+1$			(q_1, w_1)	$\cdots$	
$\vdots$					
n^k					

Clauses stating that if the head is not close to place j at time i, then symbol in place j is unchanged in the next time.
For any state q and symbol w_3, any $i \leq n_k$ and number h in a certain range we have

$$HasHead_{i,j}(q) \wedge HasSymbol_{i,j+h}(w_3) \Rightarrow HasSymbol_{i+1,j+h}(w_3)$$

If q is a rightward-moving state, do this for $n^k - j \geq h \geq 2$ **and** $-(j-1) \leq h < 0$
If q is a leftward-moving state do this for $n^k - j \geq h \geq 1$ **and** $-(j-1) \leq h < -1$

If q is a stay put state, do this for $h \neq 0$

	1	$\cdots$	j	$\cdots$	n^k
1					
i			(q, w_0)	$\cdots$ w_3	
$i+1$			(q_1, w_1)	$\cdots$ w_3	
$\vdots$					

Final configuration clause: let's assume that whenever M accepts, it accepts at LHS of tape and prints special symbol $\Box$ there

$$HasSymbol_{n^k,1}(\Box) \wedge HasHead_{n^k,1}(q_{accept})$$

At time n^k, head is at the beginning and state is accepting with special termination symbol

	1	$\cdots$			$\cdots$	n^k
1	q_0	w_1	w_2	$\cdots$		
$\vdots$						
n^k	$(q_{accept}, \Box)$					

Proof of the construction (overview, not details)

We started with M, w, constructed formula **CodesAcceptRun**$_M(w)$. Two items to establish:

- **CodesAcceptRun**$_M(w)$ is constructed in polynomial time
- **CodesAcceptRun**$_M(w)$ is satisfiable iff M accepts w

For the first item, as I pointed out, many clauses were added, but polynomially-many. (large polynomial blow-up may be counter-intuitive)

For the second, the main point is that an accepting run gives rise to a satisfying assignment of the formula (and vice versa) is a direct way, according to our understanding of what the **HasHead** and **HasSymbol** variables mean, for runs of M.

NP-Completeness Proofs

To prove that a problem $\mathcal{X}$ is NP-complete, we now just have to perform two steps:

1. Show that $\mathcal{X} \in \mathsf{NP}$ usually easy
2. Find a known NP-complete problem $\mathcal{X}'$ and reduce $\mathcal{X}' \leq_p \mathcal{X}$. the FUN part

Thousands of problem have now been shown to be NP-complete (See Garey and Johnson for an early survey); Karp 1972, "reducibility among combinatorial problems" kicked-off this work

NP-Completeness Proofs

To prove that a problem $\mathcal{X}$ is NP-complete, we now just have to perform two steps:

1. Show that $\mathcal{X} \in$ NP   usually easy
2. Find a known NP-complete problem $\mathcal{X}'$ and reduce $\mathcal{X}' \leq_p \mathcal{X}$.
   the FUN part

Thousands of problem have now been shown to be NP-complete (See Garey and Johnson for an early survey); Karp 1972, "reducibility among combinatorial problems" kicked-off this work

Coming up next: some examples. I pointed out earlier that CNF-SAT$\leq_p$3-SAT (BTW, goes back to Cook's paper) 3-SAT is a more convenient starting-point of reductions.

3-SAT$\leq_p$INTEGER PROGRAMMING (simple but important)
3-SAT$\leq_p$IND SET$\leq_p$CLIQUE
3-SAT$\leq_p$DIRECTED HAMILTONIAM PATH
3-SAT$\leq_p$SUBSET SUM$\leq_p$KNAPSACK

NP-Completeness of INTEGER PROGRAMMING, CLIQUE

IP: Input: a set of linear constraints, Question: can we satisfy them with integer values?
3-SAT$\leq_p$IP (I will do this on board)

(Recall:) CLIQUE: Given G, k, does G contain a clique of order $\geq k$?

Theorem

CLIQUE is NP*-complete.*

SAT $\leq_p$ CLIQUE
I will do this on the board. It's convenient to reduce from 3-SAT to IND SET and from there to CLIQUE.

Directed Hamiltonian Path

Input: G: directed graph.

Problem: Is there a directed path in G containing every vertex exactly once?

Theorem. DIRECTED HAMILTONIAN PATH is NP-complete

Directed Hamiltonian Path

Input: G: directed graph.

Problem: Is there a directed path in G containing every vertex exactly once?

Theorem. DIRECTED HAMILTONIAN PATH is NP-complete

Proof.

1. DIRECTED HAMILTONIAN PATH $\in$ NP.

   Take the path to be the certificate.

NP-Completeness of DIRECTED HAMILTONIAN PATH

Directed Hamiltonian Path

Input: G: directed graph.

Problem: Is there a directed path in G containing every vertex exactly once?

Theorem. DIRECTED HAMILTONIAN PATH is NP-complete

Proof.

1. DIRECTED HAMILTONIAN PATH $\in$ NP.

   Take the path to be the certificate.

2. DIRECTED HAMILTONIAN PATH is NP-hard.

   3-SATISFIABILITY $\leq_p$ DIRECTED HAMILTONIAN PATH

Digression: How to design reductions

Show that problem $\mathcal{X}$ (Dir. Hamiltonian Path) is NP-hard.

Which problem to reduce to $\mathcal{X}$:

- Arguably, the most important part is to decide where to start from; e.g. which problem to reduce to Directed Hamiltonian Path — something graph-theoretic?
- Considerations:
  - Is there an NP-complete problem similar to $\mathcal{X}$?
    (E.g. CLIQUE and Independent Set)
  - It is not always beneficial to choose a problem of the same type
    (E.g. reducing a graph problem to a graph problem)
    - For instance, CLIQUE, Independent Set are "local" problems (is there a set of vertices inducing some structure)
    - Hamiltonian Path is a global problem
      (find a structure (the Ham. path) containing all vertices)

Digression: How to design reductions

Show that problem $\mathcal{X}$ (Dir. Hamiltonian Path) is NP-hard.

Which problem to reduce to $\mathcal{X}$:

- Arguably, the most important part is to decide where to start from; e.g. which problem to reduce to Directed Hamiltonian Path — something graph-theoretic?
- Considerations:
  - Is there an NP-complete problem similar to $\mathcal{X}$?
    (E.g. CLIQUE and Independent Set)
  - It is not always beneficial to choose a problem of the same type
    (E.g. reducing a graph problem to a graph problem)
    - For instance, CLIQUE, Independent Set are "local" problems (is there a set of vertices inducing some structure)
    - Hamiltonian Path is a global problem
      (find a structure (the Ham. path) containing all vertices)

How to design the reduction:

- Does your problem come from an optimisation problem?
  If so: a maximisation problem? a minimisation problem?

NP-Completeness of SUBSET SUM

SUBSET SUM

Input: A collection of positive integers $S := \{a_1, \ldots, a_k\}$ and a target integer t.

Problem: Is there a subset $T \subseteq S$ such that $\sum_{a_i \in T} a_i = t$?

Theorem. SUBSET SUM is NP-complete

Proof.

1. SUBSET SUM $\in \mathrm{NP}$.

   Take T to be the certificate.

2. SUBSET SUM is NP-hard.

   SAT $\leq_p$ SUBSET SUM (example next slide)

Example

$$(X_1 \vee X_2 \vee X_3) \wedge (\neg X_1 \vee \neg X_4) \wedge (X_4 \vee X_5 \vee \neg X_2 \vee \neg X_3)$$

		X_1	X_2	X_3	X_4	X_5	C_1	C_2	C_3
t_1	=	1	0	0	0	0	1	0	0
f_1	=	1	0	0	0	0	0	1	0
t_2	=		1	0	0	0	1	0	0
f_2	=		1	0	0	0	0	0	1
t_3	=			1	0	0	1	0	0
f_3	=			1	0	0	0	0	1
t_4	=				1	0	0	0	1
f_4	=				1	0	0	1	0
t_5	=					1	0	0	1
f_5	=					1	0	0	0
$m_{1,1}$	=						1	0	0
$m_{1,2}$	=						1	0	0
$m_{2,1}$	=						0	1	0
$m_{3,1}$	=						0	0	1
$m_{3,2}$	=						0	0	1
$m_{3,3}$	=						0	0	1
t	=	1	1	1	1	1	3	2	4

Given: $\varphi := C_1 \wedge \cdots \wedge C_k$ in conjunctive normal form.

(w.l.o.g. at most 9 literals per clause)

Let $X_1, \ldots, X_n$ be the variables in φ. For each X_i let

$$t_i := a_1 \ldots a_n c_1 \ldots c_k \quad \textit{where} \quad a_j := \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \qquad c_j := \begin{cases} 1 & X_i \text{ occurs in } C_j \\ 0 & \text{otherwise} \end{cases}$$

$$f_i := a_1 \ldots a_n c_1 \ldots c_k \quad \textit{where} \quad a_j := \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \qquad c_j := \begin{cases} 1 & \neg X_i \text{ occurs in } C_j \\ 0 & \text{otherwise} \end{cases}$$

Example

$$(X_1 \vee X_2 \vee X_3) \wedge (\neg X_1 \vee \neg X_4) \wedge (X_4 \vee X_5 \vee \neg X_2 \vee \neg X_3)$$

		X_1	X_2	X_3	X_4	X_5	C_1	C_2	C_3
t_1	=	1	0	0	0	0	1	0	0
f_1	=	1	0	0	0	0	0	1	0
t_2	=		1	0	0	0	1	0	0
f_2	=		1	0	0	0	0	0	1
t_3	=			1	0	0	1	0	0
f_3	=			1	0	0	0	0	1
t_4	=				1	0	0	0	1
f_4	=				1	0	0	1	0
t_5	=					1	0	0	1
f_5	=					1	0	0	0
$m_{1,1}$	=						1	0	0
$m_{1,2}$	=						1	0	0
$m_{2,1}$	=						0	1	0
$m_{3,1}$	=						0	0	1
$m_{3,2}$	=						0	0	1
$m_{3,3}$	=						0	0	1
t	=	1	1	1	1	1	3	2	4

SAT $\leq_p$ SUBSET SUM

Further, for each clause C_i take $r := |C_i| - 1$ integers $m_{i,1}, \ldots, m_{i,r}$

where $m_{i,j} := c_i \ldots c_k$ with $c_j := \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases}$

Definition of S: Let

$$S := \{t_i, f_i : 1 \leq i \leq n\} \cup \{m_{i,j} : 1 \leq i \leq k, \quad 1 \leq j \leq |C_i| - 1\}$$

Target: Finally, choose as target

$$t := a_1 \ldots a_n c_1 \ldots c_k \text{ where } a_i := 1 \text{ and } c_i := |C_i|$$

Claim: There is $T \subseteq S$ with $\sum_{a_i \in T} a_i = t$ iff φ is satisfiable.

Example

$$(X_1 \vee X_2 \vee X_3) \wedge (\neg X_1 \vee \neg X_4) \wedge (X_4 \vee X_5 \vee \neg X_2 \vee \neg X_3)$$

		X_1	X_2	X_3	X_4	X_5	C_1	C_2	C_3
t_1	=	1	0	0	0	0	1	0	0
f_1	=	1	0	0	0	0	0	1	0
t_2	=		1	0	0	0	1	0	0
f_2	=		1	0	0	0	0	0	1
t_3	=			1	0	0	1	0	0
f_3	=			1	0	0	0	0	1
t_4	=				1	0	0	0	1
f_4	=				1	0	0	1	0
t_5	=					1	0	0	1
f_5	=					1	0	0	0
$m_{1,1}$	=						1	0	0
$m_{1,2}$	=						1	0	0
$m_{2,1}$	=						0	1	0
$m_{3,1}$	=						0	0	1
$m_{3,2}$	=						0	0	1
$m_{3,3}$	=						0	0	1
t	=	1	1	1	1	1	3	2	4

NP-Completeness of SUBSET SUM

Let $\varphi := \bigwedge C_i$ C_i: clauses

Show. If φ is satisfiable, then there is $T \subseteq S$ with $\sum_{s \in T} s = t$.

Let β be a satisfying assigment for φ

Set $$T_1 := \begin{array}{l} \{t_i : \beta(X_i) = 1 \quad 1 \leq i \leq m\} \cup \\ \{f_i : \beta(X_i) = 0 \quad 1 \leq i \leq m\} \end{array}$$

Further, for each clause C_i let r_i be the number of satisfied literals in C_i

(with resp. to β).

Set $T_2 := \{m_{i,j} : 1 \leq i \leq k, \quad 1 \leq j \leq |C_i| - r_i\}$

and define $T := T_1 \cup T_2$.

It follows: $\sum_{s \in T} s = t$

NP-Completeness of SUBSET SUM

Show. If there is $T \subseteq S$ with $\sum_{s \in T} s = t$, then φ is satisfiable.

Let $T \subseteq S$ s.th. $\sum_{s \in T} s = t$

Define $\beta(X_i) = \begin{cases} 1 & \text{if } t_i \in T \\ 0 & \text{if } f_i \in T \end{cases}$

This is well defined as for all i: $t_i \in T$ or $f_i \in T$ but not both.

Further, for each clause, there must be one literal set to 1 as for all i,

the $m_{i,j} : m_{i,j} \in S$ do not sum up to the number of literals in the clause.

KNAPSACK

Input: A set $I := \{1, \ldots, n\}$ of items
each of value v_i and weight w_i for $1 \leq i \leq n$
target value t weight limit ℓ

Problem: Is there $T \subseteq I$ such that

- $\sum_{i \in T} v_i \geq t$
- $\sum_{i \in T} w_i \leq \ell$

Theorem. KNAPSACK is NP-complete

NP-completeness of KNAPSACK

Knapsack

Input: A set $I := \{1, \ldots, n\}$ of items
each of value v_i and weight w_i for $1 \leq i \leq n$
target value t weight limit ℓ

Problem: Is there $T \subseteq I$ such that

- $\sum_{i \in T} v_i \geq t$
- $\sum_{i \in T} w_i \leq \ell$

Theorem. KNAPSACK is NP-complete

1. KNAPSACK $\in$ NP
   Take T as certificate.
2. KNAPSACK is NP-hard
   By reduction SUBSET SUM $\leq_p$ KNAPSACK

SUBSET SUM $\leq_p$ KNAPSACK

SUBSET SUM:

Given:	$S := \{a_1, \ldots, a_n\}$	collection of positive integers
	t	target integer

Problem: Is there a subset $T \subseteq S$ such that $\sum_{a_i \in T} a_i = t$?

SUBSET SUM $\leq_p$ KNAPSACK

SUBSET SUM:

Given:	$S := \{a_1, \ldots, a_n\}$	collection of positive integers
	t	target integer
Problem:	Is there a subset $T \subseteq S$ such that $\sum_{a_i \in T} a_i = t$?	

Reduction: From this input to SUBSET SUM construct

- $I := \{1, \ldots, n\}$: set of items
- $v_i = w_i = a_i$ for all $1 \leq i \leq n$
- target value $t' := t$ weight limit $\ell := t$

SUBSET SUM:

Given:	$S := \{a_1, \ldots, a_n\}$	collection of positive integers
	t	target integer

Problem: Is there a subset $T \subseteq S$ such that $\sum_{a_i \in T} a_i = t$?

Reduction: From this input to SUBSET SUM construct

- $I := \{1, \ldots, n\}$: set of items
- $v_i = w_i = a_i$ for all $1 \leq i \leq n$
- target value $t' := t$ weight limit $\ell := t$

Clearly: For every $T \subseteq S$

$$\sum_{a_i \in T} a_i = t \quad \Longleftrightarrow \quad \begin{array}{ll} \sum_{a_i \in T} v_i \geq t' & = t \\ \sum_{a_i \in T} w_i \leq \ell & = t \end{array}$$

Hence: The reduction is correct and in polynomial time.

A Polynomial Time Algorithm for KNAPSACK?

KNAPSACK can be solved in time $\mathcal{O}(n\ell)$ using dynamic programming

Initialisation:

Create a $(\ell + 1) \times (n + 1)$ matrix M

Set $M(w, 0) = M(0, i) = 0$ for all $1 \leq w \leq \ell \quad 1 \leq i \leq n$

Example

Input: $I := \{1, 2, 3, 4\}$ with

Values:	$v_1 := 1$	$v_2 := 3$	$v_3 := 4$	$v_4 := 2$
Weight:	$w_1 := 1$	$w_2 := 1$	$w_3 := 3$	$w_4 := 2$

Weight limit: $\ell := 5$ ***Target value:*** $t := 7$

weight limit w	max. total value from first i items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0					
1					
2					
3					
4					
5					

Set $M(w, 0) = M(0, i) = 0$ for all $1 \leq w \leq \ell$ $1 \leq i \leq n$

Input: $I := \{1, 2, 3, 4\}$ with

Values: $v_1 := 1$ $v_2 := 3$ $v_3 := 4$ $v_4 := 2$

Weight: $w_1 := 1$ $w_2 := 1$ $w_3 := 3$ $w_4 := 2$

Weight limit: $\ell := 5$ ***Target value:*** $t := 7$

weight	max. total value from first i items				
limit w	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	0	0	0	0	0
1	0				
2	0				
3	0				
4	0				
5	0				

Set $M(w, 0) = M(0, i) = 0$ for all $1 \leq w \leq \ell$ $1 \leq i \leq n$

A Polynomial Time Algorithm for KNAPSACK?

KNAPSACK can be solved in time $\mathcal{O}(n\ell)$ using dynamic programming

Initialisation:

Create a $(\ell + 1) \times (n + 1)$ matrix M

Set $M(w, 0) = M(0, i) = 0$ for all $1 \leq w \leq \ell \quad 1 \leq i \leq n$

A Polynomial Time Algorithm for KNAPSACK?

KNAPSACK can be solved in time $\mathcal{O}(n\ell)$ using dynamic programming

Initialisation:

Create a $(\ell+1) \times (n+1)$ matrix M

Set $M(w,0) = M(0,i) = 0$ for all $1 \leq w \leq \ell \quad 1 \leq i \leq n$

Computation: For $i = 0, 1, \ldots, n-1$ set $M(w, i+1)$ as

$$M(w, i+1) := \max\{M(w,i), \quad M(w - w_{i+1}, i) + v_{i+1}\}$$

Here, if $w - w_{i+1} < 0$ we always take $M(w,i)$.

$M(w,i)$: Largest total value obtainable by selecting from the first i items with weight limit w

Acceptance: If M contains an entry $\geq t$, answer yes

Otherwise reject

Example

Input: $I := \{1, 2, 3, 4\}$ with

Values: $v_1 := 1$ $v_2 := 3$ $v_3 := 4$ $v_4 := 2$

Weight: $w_1 := 1$ $w_2 := 1$ $w_3 := 3$ $w_4 := 2$

Weight limit: $\ell := 5$ ***Target value:*** $t := 7$

weight limit w	max. total value from first i items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	0	0	0	0	0
1	0				
2	0				
3	0				
4	0				
5	0				

For $i = 0, 1, \ldots, n-1$ set $M(w, i+1)$ as

$$M(w, i+1) := \max\{M(w, i), \quad M(w - w_{i+1}, i) + v_{i+1}\}$$

Example

Input: $I := \{1, 2, 3, 4\}$ with

Values:	$v_1 := 1$	$v_2 := 3$	$v_3 := 4$	$v_4 := 2$
Weight:	$w_1 := 1$	$w_2 := 1$	$w_3 := 3$	$w_4 := 2$

Weight limit: $\ell := 5$ ***Target value:*** $t := 7$

weight limit w	max. total value from first i items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	0	0	0	0	0
1	0	1			
2	0	1			
3	0	1			
4	0	1			
5	0	1			

For $i = 0, 1, \ldots, n-1$ set $M(w, i+1)$ as

$$M(w, i+1) := \max\{M(w,i), \quad M(w - w_{i+1}, i) + v_{i+1}\}$$

Example

Input: $I := \{1, 2, 3, 4\}$ with

Values:	$v_1 := 1$	$v_2 := 3$	$v_3 := 4$	$v_4 := 2$
Weight:	$w_1 := 1$	$w_2 := 1$	$w_3 := 3$	$w_4 := 2$

Weight limit: $\ell := 5$ ***Target value:*** $t := 7$

weight limit w	max. total value from first i items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	0	0	0	0	0
1	0	1	3		
2	0	1			
3	0	1			
4	0	1			
5	0	1			

For $i = 0, 1, \ldots, n-1$ set $M(w, i+1)$ as

$$M(w, i+1) := \max\{M(w, i), \quad M(w - w_{i+1}, i) + v_{i+1}\}$$

Example

Input: $I := \{1, 2, 3, 4\}$ with

Values: $v_1 := 1$ $v_2 := 3$ $v_3 := 4$ $v_4 := 2$
Weight: $w_1 := 1$ $w_2 := 1$ $w_3 := 3$ $w_4 := 2$

Weight limit: $\ell := 5$ ***Target value:*** $t := 7$

weight limit w	max. total value from first i items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	0	0	0	0	0
1	0	1	3		
2	0	1	4		
3	0	1			
4	0	1			
5	0	1			

For $i = 0, 1, \ldots, n-1$ set $M(w, i+1)$ as

$$M(w, i+1) := \max\{M(w, i), \quad M(w - w_{i+1}, i) + v_{i+1}\}$$

Example

Input: $I := \{1, 2, 3, 4\}$ with

Values:	$v_1 := 1$	$v_2 := 3$	$v_3 := 4$	$v_4 := 2$
Weight:	$w_1 := 1$	$w_2 := 1$	$w_3 := 3$	$w_4 := 2$

Weight limit: $\ell := 5$ ***Target value:*** $t := 7$

weight limit w	max. total value from first i items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	0	0	0	0	0
1	0	1	3		
2	0	1	4		
3	0	1	4		
4	0	1			
5	0	1			

For $i = 0, 1, \ldots, n-1$ set $M(w, i+1)$ as

$$M(w, i+1) := \max\{M(w, i), \quad M(w - w_{i+1}, i) + v_{i+1}\}$$

Example

Input: $I := \{1, 2, 3, 4\}$ with

Values:	$v_1 := 1$	$v_2 := 3$	$v_3 := 4$	$v_4 := 2$
Weight:	$w_1 := 1$	$w_2 := 1$	$w_3 := 3$	$w_4 := 2$

Weight limit: $\ell := 5$ ***Target value:*** $t := 7$

weight limit w	max. total value from first i items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	0	0	0	0	0
1	0	1	3		
2	0	1	4		
3	0	1	4		
4	0	1	4		
5	0	1			

For $i = 0, 1, \ldots, n-1$ set $M(w, i+1)$ as

$$M(w, i+1) := \max\{M(w, i), \quad M(w - w_{i+1}, i) + v_{i+1}\}$$

Example

Input: $I := \{1, 2, 3, 4\}$ with

Values:	$v_1 := 1$	$v_2 := 3$	$v_3 := 4$	$v_4 := 2$
Weight:	$w_1 := 1$	$w_2 := 1$	$w_3 := 3$	$w_4 := 2$

Weight limit: $\ell := 5$ ***Target value:*** $t := 7$

weight limit w	max. total value from first i items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	0	0	0	0	0
1	0	1	3		
2	0	1	4		
3	0	1	4		
4	0	1	4		
5	0	1	4		

For $i = 0, 1, \ldots, n-1$ set $M(w, i+1)$ as

$$M(w, i+1) := \max\{M(w, i), \quad M(w - w_{i+1}, i) + v_{i+1}\}$$

Example

Input: $I := \{1, 2, 3, 4\}$ with

Values:	$v_1 := 1$	$v_2 := 3$	$v_3 := 4$	$v_4 := 2$
Weight:	$w_1 := 1$	$w_2 := 1$	$w_3 := 3$	$w_4 := 2$

Weight limit: $\ell := 5$ ***Target value:*** $t := 7$

weight limit w	max. total value from first i items				
	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	0	0	0	0	0
1	0	1	3	3	3
2	0	1	4	4	4
3	0	1	4	4	5
4	0	1	4	7	7
5	0	1	4	8	8

For $i = 0, 1, \ldots, n-1$ set $M(w, i+1)$ as

$$M(w, i+1) := \max\{M(w, i), \quad M(w - w_{i+1}, i) + v_{i+1}\}$$

So what's wrong? Did we prove $P = NP$?

Recall:

- Theorem: KNAPSACK is NP-complete
- KNAPSACK can be solved in time $\mathcal{O}(n\ell)$ using dynamic programming

KNAPSACK

Input: A set $I := \{1, \ldots, n\}$ of items
each of value v_i and weight w_i for $1 \leq i \leq n$
target value t weight limit ℓ

Problem: Is there $T \subseteq I$ such that

- $\sum_{i \in T} v_i \geq t$
- $\sum_{i \in T} w_i \leq \ell$

Pseudo-Polynomial Time

This algorithm does not show that KNAPSACK is in P!

The length of the input to KNAPSACK is $\mathcal{O}(n \log \ell)$

$n \cdot \ell$ is not bounded by a polynomial in the input length!

Pseudo-Polynomial Time: Algorithms polynomial in the maximum of the input length and the value of numbers occurring in the input.

If KNAPSACK is restricted to instances with $\ell \leq p(n)$ for some polynomial p, then we obtain a problem in P.

Equivalently: KNAPSACK is in polynomial time for unary encoding of numbers.

Pseudo-Polynomial Time

This algorithm does not show that KNAPSACK is in P!

The length of the input to KNAPSACK is $\mathcal{O}(n \log \ell)$

$n \cdot \ell$ is not bounded by a polynomial in the input length!

Pseudo-Polynomial Time: Algorithms polynomial in the maximum of the input length and the value of numbers occurring in the input.

If KNAPSACK is restricted to instances with $\ell \leq p(n)$ for some polynomial p, then we obtain a problem in P.

Equivalently: KNAPSACK is in polynomial time for unary encoding of numbers.

Strong NP-completeness: Problems which remain NP-complete even if all numbers are bounded by a polynomial in the input length (equivalently, for unary encoding of numbers).

Strong NP-completeness

Pseudo Polynomial time: Algorithms polynomial in the maximum of the input length and the value of numbers occurring in the input.

Examples.

- SUBSET SUM
- KNAPSACK

Strong NP-completeness: Problems which remain NP-complete even if all numbers are bounded by a polynomial in the input length.

Examples.

- CLIQUE
- SAT
- HAMILTON CYCLE

Note. The reduction SAT $\leq_p$ SUBSET SUM involved exponentially large numbers.

digression: dealing with NP-hardness

- Maybe a pseudo-polynomial time algorithm is OK
- Move from exact to approximate optimisation: it may be hard to find optimal solution, but finding one within fact 2 (say) of optimal of optimal, is in P.
- fixed-parameter tractability
- model data as noisy (e.g. in smoothed analysis)

NP and co-NP

Notation. For a language $\mathcal{L} \subseteq \Sigma^*$ let $\overline{\mathcal{L}} := \Sigma^* \setminus \mathcal{L}$ be its complement.

Definition.
If $\mathcal{C}$ is a complexity class, we define

$$\text{co-}\mathcal{C} := \{\mathcal{L} : \overline{\mathcal{L}} \in \mathcal{C}\}.$$

CO-NP***:*** In particular, $co - NP := \{\mathcal{L} : \overline{\mathcal{L}} \in \mathrm{NP}\}$

A problem belongs to co-NP, if no-instances have short certificates.

Examples of problems in co-NP:

NO HAMILTONIAN CYCLE

Given:	Graph G
Question:	Is it true that G contains no Hamiltonian cycle?

TAUTOLOGY

Given:	Formula φ
Question:	Is φ a tautology, i.e. satisfied by all assignments?

Examples of problems in co-NP:

NO HAMILTONIAN CYCLE
Given: Graph G
Question: Is it true that G contains no Hamiltonian cycle?

TAUTOLOGY
Given: Formula φ
Question: Is φ a tautology, i.e. satisfied by all assignments?

Definition. A language $\mathcal{C} \in$ co-NP is co-NP-complete, if $\mathcal{L} \leq_p \mathcal{C}$ for all $\mathcal{L} \in$ co-NP.

P, NP, and co-NP

Proposition.

1. P $=$ co-P
2. Hence, P $\subseteq \mathrm{NP} \cap \mathrm{co\text{-}NP}$

Question:

- NP $=$ co-NP?

  Again, most people do not think so.

- P $=$ NP $\cap$ co-NP?

  Again, most people do not think so.