NICTA

Logic and Complexity

Andreas Bauer

NICTA & ANU

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Members
NICTA Partners

Outline

Why and how to study complexity?

Some definitions

What we mean by "logic" (for now):

- Let T be a logical system (i.e., a formal language + axioms + inference rules), and f be an expression in the formal language of T. E.g., propositional logic, first-order logic, etc.
- We can interpret f in T wrt. an enforcement relation, $\models$, as follows. Given a logical structure M (whose properties depend on T), either $M \models f$ or $M \not\models f$ holds. E.g., M can be a truth-value assignment for f's variables.
- If $M \models f$ holds, then M is also called a model for f.

Note that

- f gives rise to a language, i.e., all logical structures M_i such that $M_i \models f$. (Sometimes infinitely many.)
- We can say that if $M \models f$, then $M \in \mathcal{L}(f)$. $\mathcal{L}(f)$ is a set, called the language of f.

Some definitions

What we mean by "logic" (for now):

- Let T be a logical system (i.e., a formal language + axioms + inference rules), and f be an expression in the formal language of T. E.g., propositional logic, first-order logic, etc.
- We can interpret f in T wrt. an enforcement relation, $\models$, as follows. Given a logical structure M (whose properties depend on T), either $M \models f$ or $M \not\models f$ holds. E.g., M can be a truth-value assignment for f's variables.
- If $M \models f$ holds, then M is also called a model for f.

Note that

- f gives rise to a language, i.e., all logical structures M_i such that $M_i \models f$. (Sometimes infinitely many.)
- We can say that if $M \models f$, then $M \in \mathcal{L}(f)$. $\mathcal{L}(f)$ is a set, called the language of f.

Connections between logic and complexity

Urging questions regarding any T from a "user's point of view" (besides others such as "is T suitable to express my problems?"):

The validity problem

Does there exist an effective method that decides whether any given f is a valid statement in T (i.e., whether or not all logical structures are models for f, then written as $\models f$).

Ex.: Propositional logic

Let T be propositional logic, and f be any propositional logic formula. To see whether $\models f$ holds, we can use a simple truth-table. Hence, propositional logic is decidable (as the truth-table method always works; it is effective but perhaps not efficient).

Connections between logic and complexity

Urging questions regarding any T from a "user's point of view" (besides others such as "is T suitable to express my problems?"):

The validity problem

Does there exist an effective method that decides whether any given f is a valid statement in T (i.e., whether or not all logical structures are models for f, then written as $\models f$).

Ex.: Propositional logic

Let T be propositional logic, and f be any propositional logic formula. To see whether $\models f$ holds, we can use a simple truth-table. Hence, propositional logic is decidable (as the truth-table method always works; it is effective but perhaps not efficient).

NICTA

The satisfiability problem

- Given a logic T and some expression f, how inherently difficult is it, to determine whether or not there exists a model for f?
- If, for any f from T, it is always decidable by an effective method whether or not there exists a model, then we call T a decidable logic.

Ex.: Propositional logic

We already know that propositional logic is decidable (truth-table test). But how efficient is the truth-table method for checking satisfiability of a given formula?

Connections between logic and complexity

The satisfiability problem

- Given a logic T and some expression f, how inherently difficult is it, to determine whether or not there exists a model for f?
- If, for any f from T, it is always decidable by an effective method whether or not there exists a model, then we call T a decidable logic.

Ex.: Propositional logic

We already know that propositional logic is decidable (truth-table test). But how efficient is the truth-table method for checking satisfiability of a given formula?

NICTA

The model checking problem

How inherently difficult is it for a given T, f and M, to determine whether $M \models f$ holds?

Ex.: Propositional logic

Let $f \equiv (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$, and $M := \{x_1 \mapsto \mathit{true}, x_2 \mapsto \mathit{true}, x_3 \mapsto \mathit{true}\}$. Does $M \models f$ hold? (Is f a valid formula?)

NICTA

The model checking problem

How inherently difficult is it for a given T, f and M, to determine whether $M \models f$ holds?

Ex.: Propositional logic

Let $f \equiv (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$, and $M := \{x_1 \mapsto \mathit{true}, x_2 \mapsto \mathit{true}, x_3 \mapsto \mathit{true}\}$. Does $M \models f$ hold? (Is f a valid formula?)

When is a procedure efficient?

NICTA

To answer this, we first need to agree what exactly we mean by some of the used terms, and then define a suitable metric wrt. them.

Definition (A problem consists of a...)

- General description of all its parameters;
- Statement of what the answer/solution is required to satisfy.

A particular problem instance is then obtained by instantiating all parameters.

Ex.: Satisfiability problem of PL

- Parameter: a propositional logic formula f
- Does there exist an assignment M, s.t. $M \models f$?
- Instance: $f \equiv (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$

When is a procedure efficient?

NICTA

To answer this, we first need to agree what exactly we mean by some of the used terms, and then define a suitable metric wrt. them.

Definition (A problem consists of a...)

- General description of all its parameters;
- Statement of what the answer/solution is required to satisfy.

A particular problem instance is then obtained by instantiating all parameters.

Ex.: Satisfiability problem of PL

- Parameter: a propositional logic formula f
- Does there exist an assignment M, s.t. $M \models f$?
- Instance: $f \equiv (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$

Algorithms

NICTA

Definition (Algorithm)

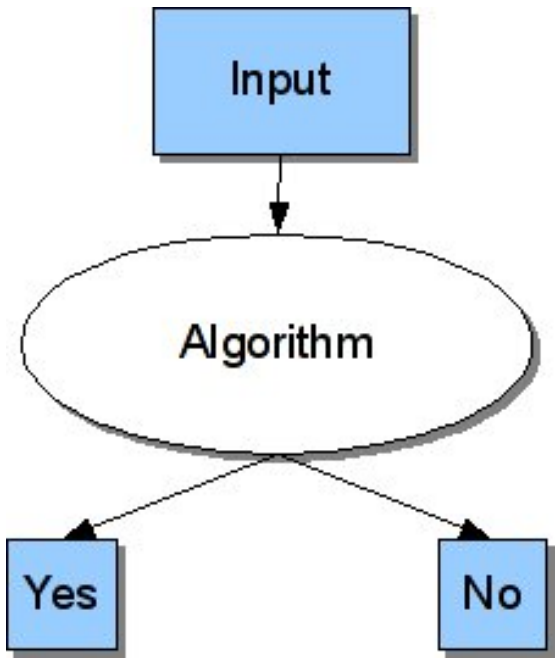
- An algorithm is a step-by-step procedure for solving problems.
- An algorithm solves problem Π if applied to any instance I of Π it always produces a solution.

Note that an algorithm for solving the satisfiability problem of PL (PL-SAT) doesn't necessarily have to compute a model, merely answer YES or NO.

Solving decision problems

NICTA

In complexity theory, we are mostly concerned with solving decision problems (although there are complexity classes for other problems, such as counting problems):

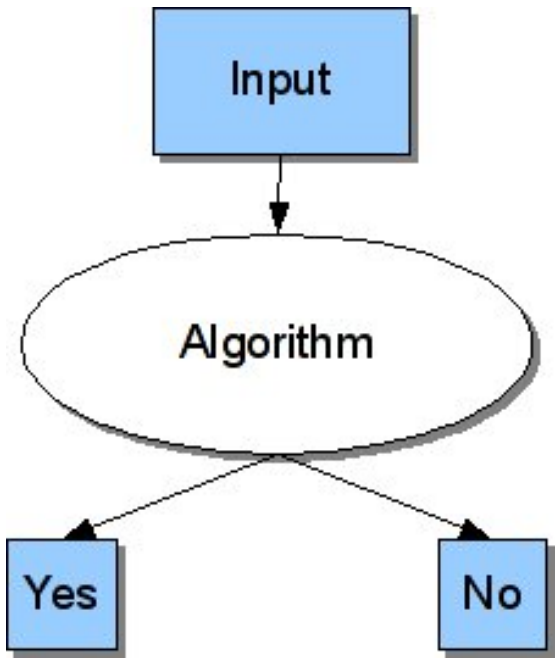


What other decision problems are there?

Solving decision problems

NICTA

In complexity theory, we are mostly concerned with solving decision problems (although there are complexity classes for other problems, such as counting problems):



What other decision problems are there?

Towards a precise metric for efficiency

NICTA

What is a metric to express efficiency of an algorithm?

- By efficiency of an algorithm, we normally mean the time it spends on solving a problem.
- Time can be measured w. r. t. the size of a problem instance (i.e., the amount of input data needed to describe an instance)

But how to express the size of the input data?

Towards a precise metric for efficiency

What is a metric to express efficiency of an algorithm?

- By efficiency of an algorithm, we normally mean the time it spends on solving a problem.
- Time can be measured w. r. t. the size of a problem instance (i.e., the amount of input data needed to describe an instance)

But how to express the size of the input data?

Encoding a problem instance

NICTA

Encoding scheme

We think of every problem providing a particular encoding scheme, which maps problem instances to strings describing them.

Input length

Input length of I for Π is the number of symbols in the description of I, obtained from the encoding scheme for Π.

Encoding a problem instance

NICTA

Encoding scheme

We think of every problem providing a particular encoding scheme, which maps problem instances to strings describing them.

Input length

Input length of I for Π is the number of symbols in the description of I, obtained from the encoding scheme for Π.

Encoding PL-SAT

NICTA

Ex.: PL-SAT

We could define an alphabet $A := \{x, 0, \ldots, 9, \vee, \wedge, \neg, (,), [,]\}$, and encode the instance $(\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$ as

$$(\neg x[1] \vee x[2] \vee x[3]) \wedge (\neg x[1] \vee x[2] \vee \neg x[3]),$$

a string of 36 symbols from our alphabet.

Time complexity

Definition (Time complexity)

Expresses the time requirements for an algorithm by giving for each possible input length, the largest amount of time needed by an algorithm to solve an instance of that size.

Naturally, not well-defined until details regarding

- encoding scheme, and
- computational model

are known. However, as we shall see, the particular choices for both do not reflect on the distinctions made between different complexity classes.

NICTA

Assumption

From now on, we assume for every problem a "reasonable encoding scheme".

Ex.: Different encodings for PL-SAT

Encoding scheme	String	Length
Via alphabet A	$(\neg x[1] \vee x[2] \vee x[3]) \wedge (\neg x[1] \vee x[2] \vee \neg x[3])$	36
SAT tools	$-1\ 2\ 3 \backslash n\ -1\ 2\ -3$	14

Not more than a polynomial difference!

NICTA

Assumption

From now on, we assume for every problem a "reasonable encoding scheme".

Ex.: Different encodings for PL-SAT

Encoding scheme	String	Length
Via alphabet A	$(\neg x[1] \vee x[2] \vee x[3]) \wedge (\neg x[1] \vee x[2] \vee \neg x[3])$	36
SAT tools	$-1\ 2\ 3 \backslash n\ -1\ 2\ -3$	14

Not more than a polynomial difference!

What is a reasonable problem encoding?

Formal definition difficult, but...

Definition (Reasonable encoding)

We call an encoding reasonable if

- the encoding of instance I is concise and not padded with unnecessary information or symbols, and
- numbers occurring in I are represented in binary (or decimal, octal, hexadecimal, etc.)

Intuitively: a problem's difficulty should not depend on its encoding!

Polynomial time algorithms

Let f and g be functions from $\mathbb{N}$ to $\mathbb{N}$.

Definition (Polynomial function)

We write $f(n) = O(g(n))$ if there are numbers c and n_0 s.t. for all $n \geq n_0$

$$f(n) \leq c \cdot g(n).$$

A polynomial time algorithm is one whose complexity is in $O(p(n))$, where p is a polynomial function and n the input length (e.g., $n^2, n^5, n^{16}, n^{4711}$).

An algorithm whose time complexity cannot be described as such is an exponential time algorithm.

Some example complexity functions

NICTA

Execution time of some algorithm in microseconds:

Time complexity function	Size n					
	10	20	30	40	50	60
n	.00001 second	.00002 second	.00003 second	.00004 second	.00005 second	.00006 second
n^2	.0001 second	.0004 second	.0009 second	.0016 second	.0025 second	.0036 second
n^3	.001 second	.008 second	.027 second	.064 second	.125 second	.216 second
n^5	.1 second	3.2 seconds	24.3 seconds	1.7 minutes	5.2 minutes	13.0 minutes
2^n	.001 second	1.0 second	17.9 minutes	12.7 days	35.7 years	366 centuries
3^n	.059 second	58 minutes	6.5 years	3855 centuries	2×10^8 centuries	1.3×10^{13} centuries

Garey & Johnson (1979), §1.3

Make the computers go faster!

NICTA

Size $(N_1, N_2, \ldots)$ of largest problem instance solvable in $1h$:

Time complexity function	With present computer	With computer 100 times faster	With computer 1000 times faster
n	N_1	$100\ N_1$	$1000\ N_1$
n^2	N_2	$10\ N_2$	$31.6\ N_2$
n^3	N_3	$4.64\ N_3$	$10\ N_3$
n^5	N_4	$2.5\ N_4$	$3.98\ N_4$
2^n	N_5	$N_5 + 6.64$	$N_5 + 9.97$
3^n	N_6	$N_6 + 4.19$	$N_6 + 6.29$

Garey & Johnson (1979), §1.3

Tractable problems

- Problems which can be solved by a polynomial time algorithm are called tractable problems.
- If no polynomial time algorithm exists, a problem is called intractable.
- Many interesting problems are (believed to be) intractable (e.g., model checking, decidability of formal logics, etc.).

Are we doomed?

- Complexity is a worst-case measure. 2^n means that there exist instances for which a running time of 2^n is unavoidable. Often algorithms perform much better in practice.
- Heuristics. For many intractable problems, there exist algorithms employing very efficient heuristics (e.g., for PL-SAT or just SAT).
- Tractability is not all. If you find an algorithm which has a time complexity function of $10^{99}n^2$, would you call it efficient?
- Size matters. E.g., a 2^n algorithm is faster than n^5 for $n \leq 20$.

Computational models

Recall: Time complexity of a problem defined wrt. encoding scheme and computational model.

All realistic models of computers studied so far, Turing machines (TM), multi-tape TMs, random access machines (RAM), are equivalent w. r. t. polynomial time complexity.

That is,

- there is a polynomial bound on the amount of work that can be done in a single time unit;
- if a problem is intractable on any of those computational models, it is intractable on the other ones as well.

Computational models

NICTA

Time required by machine A to simulate the execution of an algorithm of time complexity $T(n)$ on machine B:

Simulated machine B	Simulating machine A		
	1TM	kTM	RAM
1-Tape Turing Machine (1TM)	—	$O(T(n))$	$O(T(n)\log T(n))$
k-Tape Turing Machine (kTM)	$O(T^2(n))$	—	$O(T(n)\log T(n))$
Random Access Machine (RAM)	$O(T^3(n))$	$O(T^2(n))$	—

Garey & Johnson (1979), §1.4

Computational models

NICTA

Definition (Reasonable model of a computer)

A model of a computer is called reasonable if it is equivalent to a TM wrt. polynomial time complexity.

Example for a non-reasonable computational model: quantum computer.

For the quantum model, separate definitions and classes of complexity exist that cannot be naturally expressed in terms of the classical ones.

NICTA

Definition (Reasonable model of a computer)

A model of a computer is called reasonable if it is equivalent to a TM wrt. polynomial time complexity.

Example for a non-reasonable computational model: quantum computer.

For the quantum model, separate definitions and classes of complexity exist that cannot be naturally expressed in terms of the classical ones.

NICTA

Definition (Reasonable model of a computer)

A model of a computer is called reasonable if it is equivalent to a TM wrt. polynomial time complexity.

Example for a non-reasonable computational model: quantum computer.

For the quantum model, separate definitions and classes of complexity exist that cannot be naturally expressed in terms of the classical ones.

Proving intractability

NICTA

We now have two notions of defeating tractability:

- A problem can be shown to be so difficult that in the worst case an exponential amount of time is needed to solve it.
- The solution is so extensive that it cannot be described as expression of polynomial size.

Although accurate and correct, are these two notions exclusive, i.e., are there other (perhaps worse) forms of intractability?

Undecidability vs. Intractability

The famous Halting Problem

Given an arbitrary TM program, an arbitrary input, does the TM eventually halt when applied to that input?

- Turing showed in 1936 that this problem is undecidable, i.e., there exists no algorithm for solving it.
- Other undecidable problems include tiling problems, solvability of polynomial equations in integers, PCP, etc.
- Certainly, every undecidable problem is intractable, but not the other way round

And finally for today...

NICTA

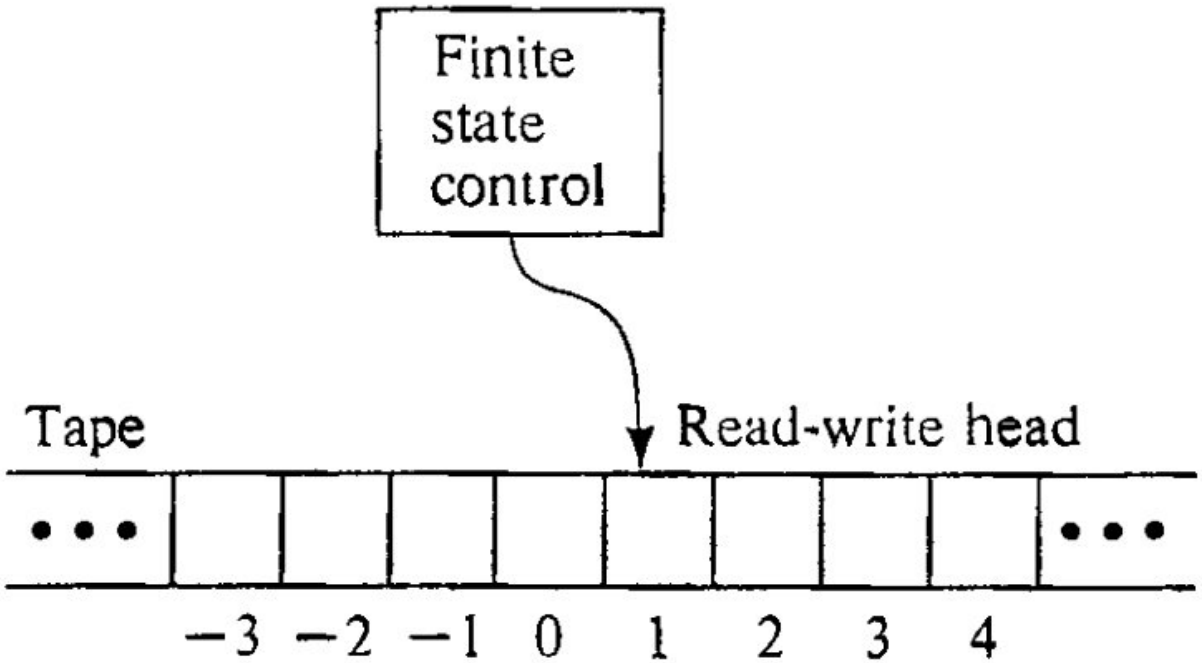
Turing machine $\neq$ Turing test:

"On the Internet, nobody knows you're a dog."

NP-completeness

Deterministic TMs (DTMs)

NICTA



Garey & Johnson (1979), §2.2

What is Turing-completeness?
Is <your PL of choice> Turing-complete?

DTM programs

NICTA

Definition (DTM program)

A program for a DTM specifies the following information:

- A finite set Γ of tape symbols, including a subset $\Sigma \subset \Gamma$ of input symbols and a distinguished blank $b \in \Gamma - \Sigma$.
- A finite set Q of states, including $q_0 \in Q$ and two halt states $q_Y, q_N \in Q$.
- A transition function

$$\delta : Q - \{q_Y, q_N\} \times \Gamma \to Q \times \Gamma \times \{-1, +1\}.$$

How a DTM operates

NICTA

- A DTM's input is $x \in \Sigma^*$, placed on squares $1, \ldots, |x|$.
- All other squares contain b.
- Let q_c denote the current state. Initially, $q_c = q_0$.
- If $q_c \in \{q_Y, q_N\}$ TM halts,
- Otherwise, a symbol $s \in \Sigma$ is read from the current tape square the head is on and a transition is made: $\delta(q_c, s) = (q', s', \Delta)$:
  - The DTM moves to state q' (i.e., we update $q_c = q'$),
  - replaces s with s',
  - and moves left if $\Delta = -1$,
  - or right if $\Delta = +1$.

This completes one step of the DTM, if $q_c \notin \{q_Y, q_N\}$, the operation continues accordingly.

Example DTM program

NICTA

$$\Gamma = \{0, 1, b\},\ \Sigma = \{0, 1\}$$

$$Q = \{q_0, q_1, q_2, q_3, q_Y, q_N\}$$

q	0	1	b
q_0	$(q_0, 0, +1)$	$(q_0, 1, +1)$	$(q_1, b, -1)$
q_1	$(q_2, b, -1)$	$(q_3, b, -1)$	$(q_N, b, -1)$
q_2	$(q_Y, b, -1)$	$(q_N, b, -1)$	$(q_N, b, -1)$
q_3	$(q_N, b, -1)$	$(q_N, b, -1)$	$(q_N, b, -1)$

$$\delta(q, s)$$

An example DTM program $M = (\Gamma, Q, \delta)$.

Garey & Johnson (1979), §2.2

What does M compute? Try it on $x = 10100$!

The language of a DTM

Recall: We already defined for a logical formula φ in some logic, its language, i.e., $\mathcal{L}(\varphi)$ is a set containing all of φ's models.

Language of a TM

The (accepted) language of a TM consists of all input strings for which the TM's answer is "yes", i.e., it halts on q_Y, or:

$$L_M = \{x \in \Sigma^* \mid M \text{ accepts } x\}$$

The language of the previous example DTM can thus be given as

$$\{x \in \{0,1\} \mid \text{ the rightmost two symbols are } 0\}.$$

How does the DTM behave for any $x \notin L_M$?

DTMs and time complexity

NICTA

A formal definition of time complexity is now possible:

For a DTM program M that halts for all inputs $x \in \Sigma^*$, its time complexity function $T_M : \mathbb{N} \to \mathbb{N}$ is given by

$$T_M(n) = max \left\{ m \mid \begin{array}{l} \text{there is an } x \in \Sigma^* \text{ with } |x| = n, \text{ such that} \\ M\text{'s computation on } x \text{ takes time } m. \end{array} \right\}$$

A program said to run in polynomial time if there exists a polynomial p s.t. for all $n \in \mathbb{N}$, $T_M(n) \leq p(n)$.

The complexity class P

NICTA

Finally, we can define our first complexity class P:

Definition (The complexity class P)

$$P = \{L \mid \text{ there is a polytime DTM } M \text{ for which } L = L_M\}$$

(After all this work, this doesn't look spectacular, does it? And considering that NP vs. P is a Millennium Problem...)

Polynomial time verifiability

NICTA

Consider the the problem $\Pi = \text{SAT}$ again:

- No known algorithm for solving SAT in polynomial time.
- Suppose, someone had for an instance of SAT, I, a solution S.
- We can then verify in time polynomial in $\mathit{Length}(I)$ that the answer for S is, indeed, "yes".

Polynomial time verifiability

NICTA

Consider a variant of the Travelling Salesman Problem (TSP):

Definition (TSP)

INSTANCE: A finite set $C = \{c_1, \ldots, c_m\}$ of "cities" (i.e., identifiers for cities), a distance $d(c_i, c_j) \in \mathbb{N}$ for each pair of cities $c_i, c_j \in C$, and some bound $B \in \mathbb{N}$.
QUESTION: Is there a tour of all cities in C having a total length no greater than B, that is, an ordering $\langle c_{\pi(1)}, \ldots, c_{\pi(m)} \rangle$ of C such that

$$\left(\sum_{i=1}^{m-1} d(c_{\pi(i)}, c_{\pi(i+1)})\right) + d(c_{\pi(m)}, c_{\pi(1)}) \leq B \ ?$$

Polynomial time verifiability

NICTA

Some more notation: For some problem Π, we say...

- $I \in Y_\Pi$ iff (= if and only if) there exists some structure S that, when guessed (= magically provided) for an input I, allows us to determine that S is, indeed, a solution for I.
- $I \notin Y_\Pi$, otherwise.

Some properties of TSP:

- As with SAT, a concrete instance of TSP, I, is not (known to be)[1] solvable in polynomial time in $Length(I)$.
- However, given some (probably guessed) S, it is easily verifiable in time polynomial in $Length(I)$, whether the answer for S is "yes" (and consequently $I \in Y_\Pi$).

[1] It may happen that occasionally I omit (= forget) the "known to be part", and just say "is not solvable", which is NOT ACCURATE!

Polynomial time verifiability

Note that

- TSP originally asks for a minimal tour, without parameter B (i.e., an optimisation problem)
- We will use the TSP definition scheme as template to define further decision problems

An informal definition of the class NP

NICTA

Informally, a problem Π is in the class NP iff the TM for solving Π can be devised using a

- a guessing stage which guesses a solution S for given I, and
- a verification stage which verifies that the answer for S is, indeed, "yes" (and then, consequently $I \in Y_\Pi$).

I.e., the TM is nondeterministic.
Note that the verification stage has to be deterministic!

An informal definition of the class NP

We say...

A nondeterministic algorithm/TM solves a problem in polynomial time if there exists a polynomial p, such that for every instance $I \in Y_{\Pi}$, there is some guess S that leads the deterministic checking stage to respond "yes" for I and S within time $p(Length(I))$.

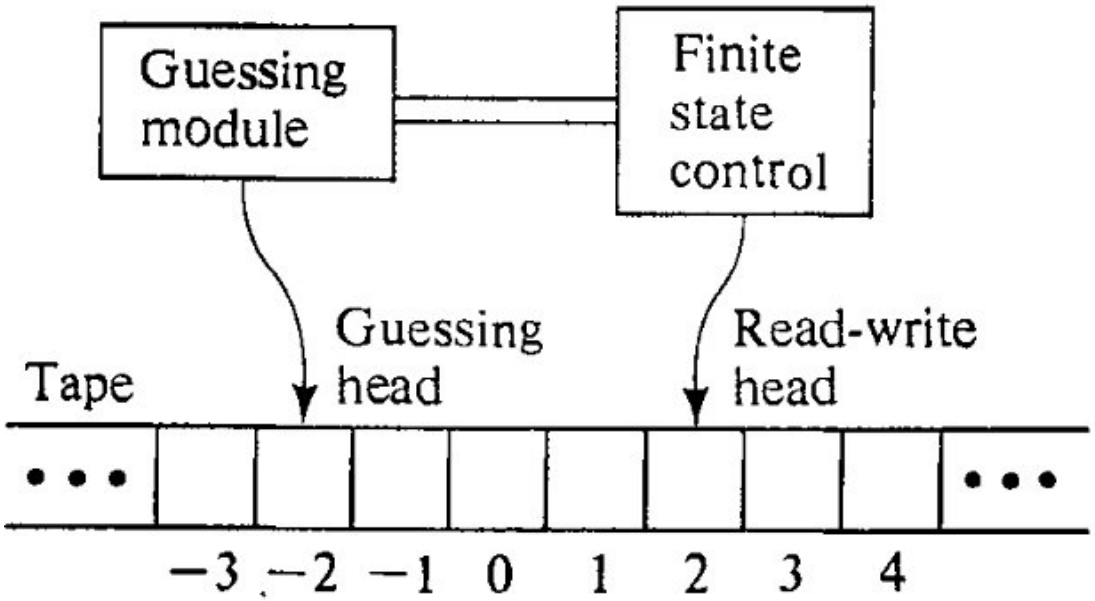
How is nondeterminism useful?

- That is not fully understood, i.e., we do not know whether $P = NP$?
- The deterministic model for PCs has succeeded. :-) (or do you know of a vendor that sells nondeterministic PCs?)

Since there isn't a standard for a nondeterministic computation device, we will define our own which is somewhat simpler than the one usually found in the literature (cf. Aho, Hopcroft & Ullman (1974)) (yet equivalent wrt. polynomial time transferrability!)

Nondeterministic Turing Machine (NDTM)

NICTA



Garey & Johnson (1979), §2.4

The guessing head is also called a write-only head.

NDTM programs

NICTA

Definition (NDTM program)

A program for an NDTM specifies the following information:

- A finite set Γ of tape symbols, including a subset $\Sigma \subset \Gamma$ of input symbols and a distinguished blank $b \in \Gamma - \Sigma$.
- A finite set Q of states, including $q_0 \in Q$ and two halt states $q_Y, q_N \in Q$.
- A transition function

$$\delta : Q - \{q_Y, q_N\} \times \Gamma \to Q \times \Gamma \times \{-1, +1\}.$$

I.e., exactly as a DTM program. However, computation strictly occurs in two stages.

NDTM programs

NICTA

Stage 1: Guessing stage

- Input is written on squares $1, \ldots, |x|$ (others blank)
- Read-write head is positioned at 1
- Write-only head is positioned at -1
- WO head writes symbols from Γ to current square and moves left, or stops. If stop, then guessing stage is finished.

Stage 2: Checking/verification stage

- Finite state control is active in state q_0
- WO head inactive
- Accepting computation: Finite state control reaches q_Y.
- Non-accepting computation: Reaching of q_N or no halt at all.

NICTA

Language of an NDTM

An NDTM M has an infinite number of computations for a given x (i.e., input string encoding I), one for each guessed string from Γ^*. M accepts x if at least one of them leads to an accepting computation. The (accepted) language of M is:

$$L_M = \{x \in \Sigma^* \mid M \text{ accepts } x\}$$

NDTMs and time complexity

The time complexity function of a NDTM, M, $T_M : \mathbb{N} \to \mathbb{N}$ is given by

$$T_M(n) = max \left\{ m \mid \{1\} \cup \begin{array}{l} \text{there is an } x \in \Sigma^* \text{ with } |x| = n, \text{ such that} \\ M\text{'s computation on } x \text{ takes time } m. \end{array} \right\}$$

Note that

- the time required by M to accept a string $x \in L_M$ is defined to be the minimum over all accepting computations of M for x
- the time complexity depends only on the number of steps occurring in accepting computations and that it is defined to be 1 whenever no inputs of length n are accepted by M.

The NDTM program M is a polynomial time program if there exists a polynomial p s.t. for all $n \in \mathbb{N}$, $T_M(n) \leq p(n)$.

Pretty much all as before.

The complexity class NP

NICTA

Definition (The complexity class NP)

$$NP = \{L \mid \text{ there is a polytime NDTM } M \text{ for which } L = L_M\}$$

Also as before, except NDTM instead of DTM.

The relationship between P and NP

- $P \subseteq NP$, i.e., every decision problem solvable in polytime by a DTM, is solvable by a polytime NDTM.
- I.e., if $\Pi \in P$ and A is a deterministic algorithm for Π, we get a nondet. algorithm for Π by using A as the checking stage, and ignoring the guess.
- Thus, $\Pi \in P$ implies $\Pi \in NP$.
- It is believed that this inclusion is proper, i.e., there is no det. algorithm which simulates nondeterminism.
- In fact, all we know is

Theorem

If $\Pi \in NP$, then there exists a polynomial p such that Π can be solved by a deterministic algorithm having time complexity $O(2^{p(n)})$.

Proof

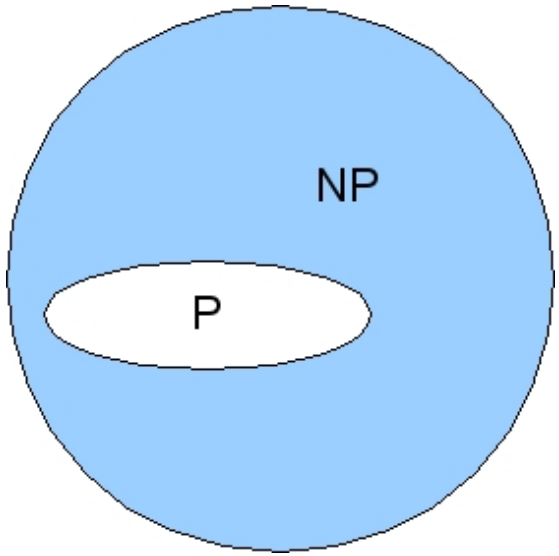
Outline:

- Suppose A is nondet. polytime alg. for Π, and $q(n)$ the time complexity bound of A.
- Then, for every input of length n, there is a guess over Γ of length at most $q(n)$ that leads A to respond "yes".
- Thus, the number of possible guesses to be considered is at most $|\Gamma|^{q(n)}$.
- We can deterministically discover whether A has an accepting computation by trying all those guesses out, one by one.
- This is clearly a deterministic procedure, but may take time $|\Gamma|^{q(n)}$, which is clearly exponential. □

Our tentative view of the world

NICTA

Hence, we have reason to believe the world looks somewhat like this:



We'll refine this view slightly as we move along.

Polynomial time reductions

NICTA

Definition (Polynomial time reduction)

A PTR from one language $L_1 \subseteq \Sigma_1^*$ to $L_2 \subseteq \Sigma_2^*$ is a function $f : \Sigma_1^* \to \Sigma_2^*$ s.t.

- there is a polynomial time DTM program that computes f
- for all $x \in \Sigma_1^*, x \in L_1$ iff $f(x) \in L_2$

If there is a PTR from L_1 to L_2 we write $L_1 \propto L_2$ ("L_1 transforms to L_2")

Lemma

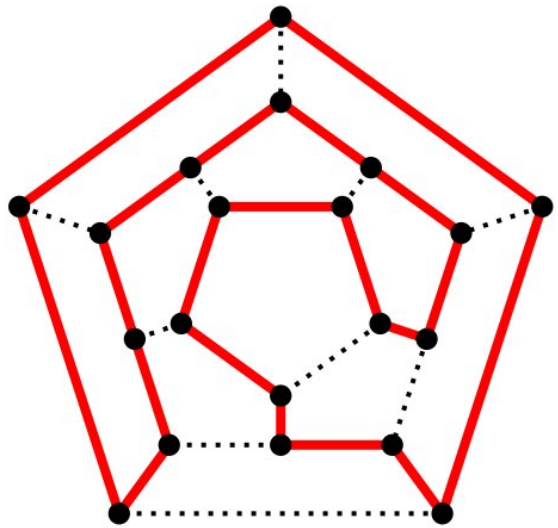
If $L_1 \propto L_2$, then $L_2 \in P$ implies $L_1 \in P$ (and, equivalently, $L_1 \notin P$ implies $L_2 \notin P$).

We will use this notion also, more generally, wrt. decision problems, and then use $\Pi_1 \propto \Pi_2$, accordingly.

The Hamiltonian cycle problem

NICTA

A Hamiltonian cycle is a cycle in an undirected graph which visits each vertex exactly once and also returns to the starting vertex.



More formally: consider a graph $G = (V, E)$. A simple cycle is a sequence $\langle v_1, \ldots, v_k \rangle$ of distinct vertices such that $(v_i, v_{i+1}) \in E$ for $1 \leq i < k$ and $(v_k, v_1) \in E$. A Hamilton cycle includes ALL vertices of V.

The Hamiltonian cycle problem

NICTA

Definition (HC)

INSTANCE: A graph $G = (V, E)$.
QUESTION: Does G contain a Hamiltonian cycle?

HC $\propto$ TSP

NICTA

Recall: find an f which is (i) polynomial time computable, and (ii) for all $x \in \Sigma_1$, $x \in L_1$ iff $f(x) \in L_2$.

Let $G = (V, E)$ with $|V| = m$ be HC instance.
We define the according TSP as follows:

- For any two cities $v_i, v_j \in C$ we have $d(v_i, v_j) = 1$ iff $(v_i, v_j) \in E$, and 2 otherwise.
- Max. tour length $B := m$.

(Informally) easy to see that this transformation is computable in polynomial time wrt. $|V| + |E|$: to determine all $m(m-1)/2$ distances $d(v_i, v_j)$ one only has to examine G whether or not $(v_i, v_j) \in E$.

Thus property (i) is satisfied.

HC $\propto$ TSP

NICTA

For (ii) we must show that G contains a Hamiltonian cycle iff the constructed instance of TSP has a tour of length at most B:
$(\Rightarrow)$: Suppose $\langle v_1, \ldots, v_m \rangle$ is a HC in G. Then, by definition of f/construction, $\langle v_1, \ldots, v_m \rangle$ is also a tour in $f(G)$, and this tour has a length of exactly B.
$(\Leftarrow)$: Suppose $\langle v_1, \ldots, v_m \rangle$ is a tour in $f(G)$ with length $\leq B$. By definition, two cities v_i, v_j can be either 1 or 2 apart, and we have a m-city tour. It follows that the individual distances must be 1. Moreover, by def. of $f(G)$, it follows that $\langle v_1, \ldots, v_m \rangle$ are all edges in G, and since $f(G)$ is a cycle we also have $(v_m, v_1) \in G$.

$\square$

Polynomial transformability

NICTA

Polynomial time reductions are transitive, i.e., we can "transform" one problem into another.

Lemma

If $L_1 \propto L_2$ and $L_2 \propto L_3$, then $L_1 \propto L_3$.

NP-completeness

NICTA

Definition (NP-completeness)

A language L is defined to be NP-complete if $L \in NP$ and for all other languages $L' \in NP$, $L' \propto L$.
Or: A decision problem Π is NP-complete if $\Pi \in NP$, and for all other decision problem $\Pi' \in NP$, $\Pi' \propto \Pi$.

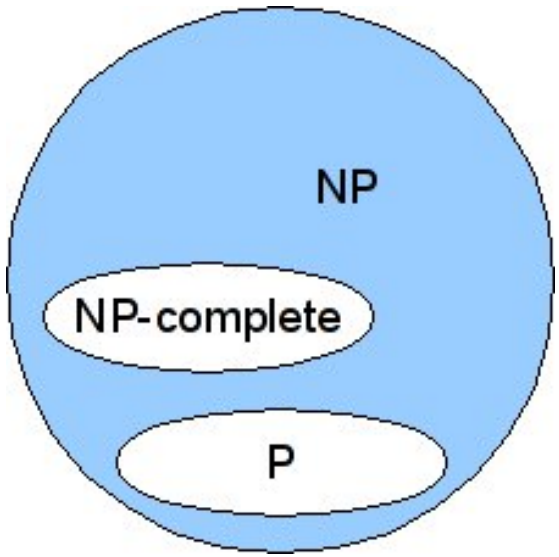
Corrolary

If any single problem in NP can be solved in polynomial time, then all problems in NP can be solved in polynomial time (and we would have, $P = NP$).

Our tentative view of the world

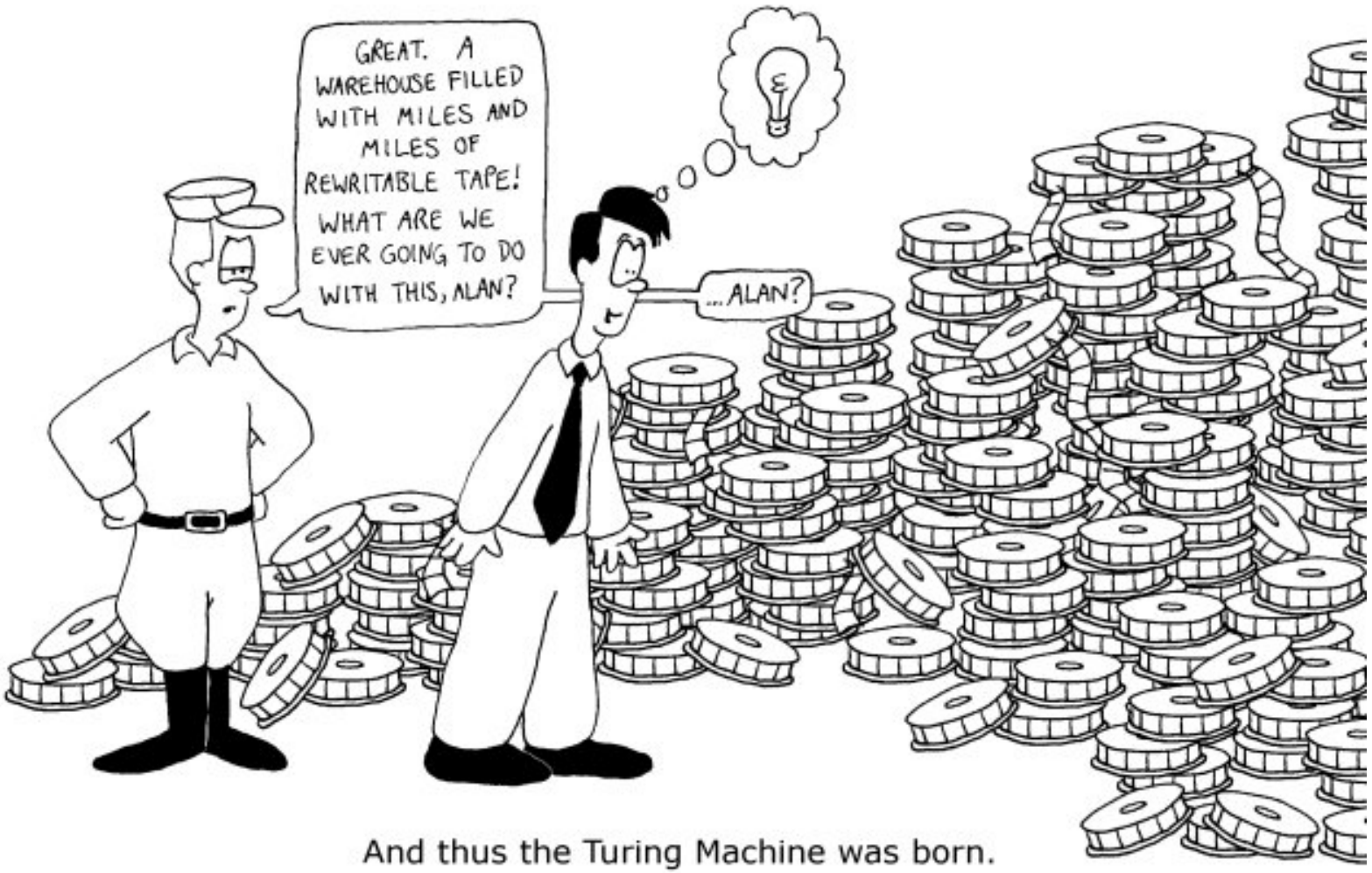
NICTA

If we assume that $P \neq NP$, we obtain the following picture:



What text books usually forget to tell you...

NICTA



And thus the Turing Machine was born.

Showing NP-completeness

NICTA

Lemma

If L_1 and L_2 belong to NP, L_1 is NP-complete, and $L_1 \propto L_2$, then L_2 is NP-complete.

Proof.

Follos from definition of NP-completeness and transitivity of polynomial time reductions. □

This lemma suggests a way to proof NP-completeness of a "new problem" Π:

- Show that $\Pi \in NP$, and
- transform a known NP-complete problem Π' to Π.

The Chicken & Egg Problem in Complexity

If we want to show any problem to be NP-complete, we need an NP-complete problem to begin with!

...this is where Cook's Theorem comes in.

Notations and definitions

Let us make some previously used notions precise:

- $U = \{u_1, \ldots, u_n\}$ is a set of Boolean variables
- A truth assignment is a function $t : U \to \mathbb{B}$ (if $t(u) = \top$, we say "u is true under t")
- If $u \in U$, then u and $\overline{u}$ are literals:
  - u is true under t iff the variable u is true under t
  - $\overline{u}$ is true under t iff the variable u is false under t
- A clause (over U) is a set of literals, e.g., $\{u_1, \overline{u}_3, u_8\}$ (disjunction of literals)

The above clause is satisfied by t unless $t(u_1) = \bot, t(u_3) = \top, t(u_8) = \bot$.

- A collection C of clauses over U is satisfiable iff there exists a truth assignment that satisfies all clauses in C.

The world's first NP-complete problem: SAT

NICTA

Definition (SAT)

INSTANCE: A set of U of variables and a collection C of clauses over U.
QUESTION: Is there a satisfying truth assignment for C?

Example: $U = \{u_1, u_2\}$ and $C = \{\{u_1, \overline{u}_2\}, \{\overline{u}_1, u_2\}\}$

$t(u_1) = t(u_2) = \top$.
Note that $C = \{\{u_1, \overline{u}_2\}, \{u_1, \overline{u}_2\}, \{\overline{u}_1\}\}$ is not sat.

Cook showed in a paper presented at STOC'71 that SAT is NP-complete.

Proof outline

NICTA

Recall, a problem is NP-complete if (i) it is in NP and (ii) all problems in NP are polynomial time reducible to it.

- Membership in NP straightforward.
- Idea of NP-hardness proof:
  - On the language level, SAT is represented as $L_{SAT} = L[SAT, e]$
  - We must show that for all $L \in NP$, it holds that $L \propto L_{SAT}$.
  - There are infinitely many such L!
  - But every such L can be represented as a polynomial time NDTM program.
  - Devise a generic transformation from a polynomial time NDTM program to L_{SAT}!

Proof: NP-hardness of SAT

- Let M be a polynomial time NDTM program given by $\Gamma, \Sigma, b, Q, q_0, q_Y, q_N$, and δ, recognising the language $L = L_M$.
- Let $p(n)$ be a polynomial over integers that bounds the time complexity function $T_M(n)$. That is,
  - M can only access squares between $-p(n)$ and up to $p(n) + 1$,
  - M will not take more than $p(n)$ steps.
- The generic transformation f_L will be derived in terms of $\Gamma, \Sigma, b, Q, q_0, q_Y, q_N, \delta$, and p:

f_L has the property that ...

for all $x \in \Sigma^*$ with $x \in L$, we have that $f_L(x)$ has a satisfying truth assignment.

Proof: NP-hardness of SAT

- f_L labels Q as follows $Q = q_0, q_1 = Y, q_2 = N, q_3, q_4, \ldots, q_r$, where $r = |Q| - 1$,
- and $\Gamma = s_0 = b, s_1, \ldots, s_v$ with $v = |\Gamma| - 1$.
- The constructed set U of variables will contain three different "types" of variables with the following intended meaning:

Variable	Range	Intended meaning
$Q[i,k]$	$0 \leqslant i \leqslant p(n)$ $0 \leqslant k \leqslant r$	At time i, M is in state q_k.
$H[i,j]$	$0 \leqslant i \leqslant p(n)$ $-p(n) \leqslant j \leqslant p(n)+1$	At time i, the read-write head is scanning tape square j.
$S[i,j,k]$	$0 \leqslant i \leqslant p(n)$ $-p(n) \leqslant j \leqslant p(n)+1$ $0 \leqslant k \leqslant v$	At time i, the contents of tape square j is symbol s_k.

Note that every computation of M automatically induces a truth assignment of those vars (but not vice versa).

Proof: NP-hardness of SAT

Our task (or rather f_L's task): for some input to M, call it x with length n, to construct (in polynomial time) a set of clauses C over U, such that C is satisfiable iff $x \in L$.

Proof: NP-hardness of SAT

The clauses in $f_L(x)$ can be categorised into six groups, each imposing a different restriction on any satisfying truth assignment:

Clause group	Restriction imposed
G_1	At each time i, M is in exactly one state.
G_2	At each time i, the read-write head is scanning exactly one tape square.
G_3	At each time i, each tape square contains exactly one symbol from Γ.
G_4	At time 0, the computation is in the initial configuration of its checking stage for input x.
G_5	By time $p(n)$, M has entered state q_Y and hence has accepted x.
G_6	For each time i, $0 \leqslant i < p(n)$, the configuration of M at time $i+1$ follows by a single application of the transition function δ from the configuration at time i.

Proof: NP-hardness of SAT

NICTA

G_1: At each time i, M is exactly in one state.

This means that

- for each $i = 0, \ldots, p(n)$, we have a clause

$$\{Q[i,0], Q[i,1], \ldots, Q[i,r]\}$$

  yielding $r \cdot p(n)$ literals in total, i.e., M is in at least one state in each i;
- for each $i = 0, \ldots, p(n)$ and for each pair j, j' of distinct states, we have a clause

$$\{\overline{Q[i,j]}, \overline{Q[i,j']}\},$$

  i.e, M cannot be in more than one state at a time. Yielding a total of $r \cdot (r-1) \cdot p(n)$ literals.

Proof: NP-hardness of SAT

NICTA

G_2: At each time i, M is scanning exactly one tape square

- for each $i = 0, \ldots, p(n)$ we have a clause

$$\{H[i, -p(n)], H[i, -p(n)+1], \ldots, H[i, p(n)+1]\}$$

  i.e., at least one symbol is scanned.
- for each $i = 0, \ldots, p(n)$ and distinct pair j, j', where $-p(n) \leq j \leq j' \leq p(n)+1$, we have a clause

$$\{\overline{H[i,j]}, \overline{H[i,j']}\}$$

  i.e., at most one symbol is scanned.

Proof: NP-hardness of SAT

NICTA

G_3: At each time i, each tape quare contains exactly one symbol.

- for each $i = 0, \ldots, p(n)$ and $-p(n) \leq j \leq p(n) + 1$, we have a clause
$$\{S[i,j,0], S[i,j,1], \ldots, S[i,j,v]\},$$
where $v = |\Gamma| - 1$, i.e., at least one symbol is contained
- for each $i = 0, \ldots, p(n), -p(n) \leq j \leq p(n) + 1$, and $0 \leq k \leq k' \leq v$ we have a clause
$$\{\overline{S[i,j,k]}, \overline{S[i,j,k']}\},$$
i.e., at most one symbol is contained.

Proof: NP-hardness of SAT

NICTA

G_4: At time 0, the computation is in the initial configuration of its checking state for input x.

This one's comparatively easy! We add the following clauses:

- $\{Q[0,0]\}, \{H[0,1]\}$ (set state and head)
- $\{S[0,0,0], S[0,1,k_1], \ldots, S[0,n,k_n]\}$, where $x = s_{k_1} \ldots s_{k_n}$ (input word)
- $\{S[0,n+1,0], \ldots, S[0,p(n)+1,0]\}$ (padding)

Proof: NP-hardness of SAT

NICTA

G_5: By time $p(n)$, M has entered state q_Y (and thus accepted x)

Simply add $\{Q[p(n), 1]\}$.

Proof: NP-hardness of SAT

NICTA

G_6: For each time i, $o \leq i \leq p(n)$, the configuration of M at time $i+1$ follows by a single application of δ from the configuration at time i.

For each quadruple (i, j, k, l), $0 \leq i < p(n)$, $-p(n) \leq j \leq p(n)+1$, $0 \leq k \leq r$, and $0 \leq l \leq v$, we have the following clauses:

- $\{\overline{H[i,j]}, \overline{Q[i,k]}, \overline{S[i,j,l]}, H[i+1, j+\Delta]\}$
- $\{\overline{H[i,j]}, \overline{Q[i,k]}, \overline{S[i,j,l]}, Q[i+1, k']\}$
- $\{\overline{H[i,j]}, \overline{Q[i,k]}, \overline{S[i,j,l]}, S[i+1, j, l']\}$,

where if $q_k \in Q - \{q_Y, q_N\}$, then the values of Δ, k', and l' are such that $\delta(q_k, s_l) = (q_{k'}, s_{l'}, \Delta)$, and if $q_k \in \{q_Y, q_N\}$, then $\Delta = 0$, $k' = k$, and $l' = l$.

This step adds $6(p(n))(p(n)+1)(r+1)(v+1)$ which may seem a lot but is still polynomial!

Proof: NP-hardness of SAT

NICTA

If $x \in L$, then there is an accepting computation of M on x of length $p(n)$ or less, and this computation, given the interpretation of the variables, imposes a truth assignment that satisfies all the clauses in

$$C = G_1 \cup G_2 \cup G_3 \cup G_4 \cup G_5 \cup G_6.$$

Conversely, it is easy to see that the construction of C does correspond to an accepting computation of M on x.

How to prove NP-completeness

Given what we know already, what is the canonical procedure of showing a problem Π to be NP-complete?

How to prove NP-completeness

- Show that Π is in NP.
- Select a known NP-complete problem Π'.
- Construct a transformation f from Π' to Π.
- Prove that f is a polynomial transformation.

Alternatively, use NDTM for the reduction.

NICTA

Definition (3SAT)

INSTANCE: A set of U of variables and a collection C of clauses over U, such that for all $c_i \in C$, we have $|c_i| = 3$.
QUESTION: Is there a satisfying truth assignment for C?

Theorem

3SAT is in NP.

Proof.

Easy. □

3SAT is NP-complete

NICTA

Proof via reduction from SAT:

- Let a SAT problem be defined as before via sets $C = \{c_1, \ldots, c_m\}$ and $U = \{u_1, \ldots, u_n\}$.
- We construct an instance of 3SAT using sets C' and U' s.t. C' is SAT over U' iff C is SAT over U.
- The idea is as follows: for each clause $c_j \in C$, we construct a set C'_j of 3-literal clauses, and a set of variables U'_j only used in C'_j, i.e.

$$U' := U \cup (\bigcup_{j=1}^{m} U'_j)$$

and

$$C' := \bigcup_{j=1}^{m} C'_j$$

3SAT is NP-complete

NICTA

How to construct C'_j and U'_j from $c_j = \{z_1, \ldots, z_k\} \in C$:

- $k = 1$: $U'_j := \{y_j^1, y_j^2\}$ and
  $C'_j := \{\{z_1, y_j^1, y_j^2\}, \{z_1, \overline{y}_j^1, y_j^2\}, \{z_1, y_j^1, \overline{y}_j^2\}, \{z_1, \overline{y}_j^1, \overline{y}_j^2\}\}$,
  i.e., z_1 has to be satisfied in order to satisfy C'_j.
- $k = 2$: Excercise! $U'_j := \{y_j^1\}$ and
  $C'_j := \{\{z_1, z_2, y_j^1\}, \{z_1, z_2, \overline{y}_j^1\}\}$,
  i.e., one of the two z_i has to be true in order to sat. C'_j.
- $k = 3$: Excercise! $U'_j := \emptyset$ and $C'_j := \{\{c_j\}\}$.
- $k > 3$: $U'_j := \{y_j^i \mid 1 \leq i \leq k-3\}$ and
  $C'_j := \{\{z_1, z_2, y_j^1\}\} \cup \{\{\overline{y}_j^i, z_{i+2}, y_j^{i+1} \mid 1 \leq i \leq k-4\}\} \cup \{\{\overline{y}_j^{k-3}, z_{k-1}, z_k\}\}$, i.e., one of the z_i has to be true in order to sat. C'_j.

By the discussion, we have informally shown both directions: C sat iff C' sat!

3SAT is NP-complete

NICTA

Recall: Considering our work so far, 3SAT is NP-complete iff the transformation is polynomial!
Observe that the number of 3-literal clauses is bounded by a polynomial in nm.

□

Note that usually restricted variants of NP-complete problems are not NP-complete problems, cf. 2SAT.

Want to buy a TM?

NICTA

Showing NP-completeness & Space complexity

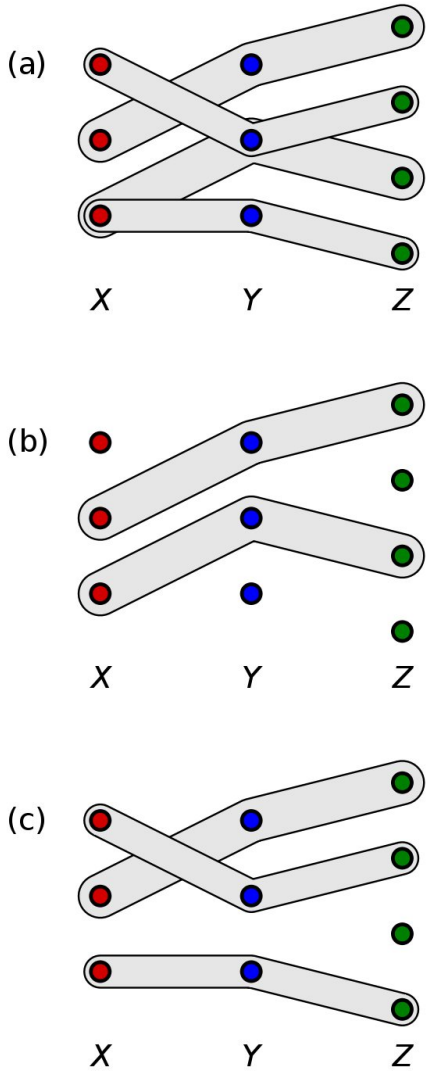
More NP-complete problems

NICTA

Definition (3D matching (3DM))

INSTANCE: 3 sets X, Y, Z, and a set $M \subseteq X \times Y \times Z$.
QUESTION: Does M contain a matching, that is, a set $M' \subseteq M$ such that s.t. for each triple (x_i, y_i, z_i) and (x_i, y_i, z_i) from M' the following holds: $x_i \neq x_j$, $y_i \neq y_j$, and $z_i \neq z_j$?

Note, we will consider the restricted case where $|X| = |Y| = |Z|$.



3DM is NP-complete

NICTA

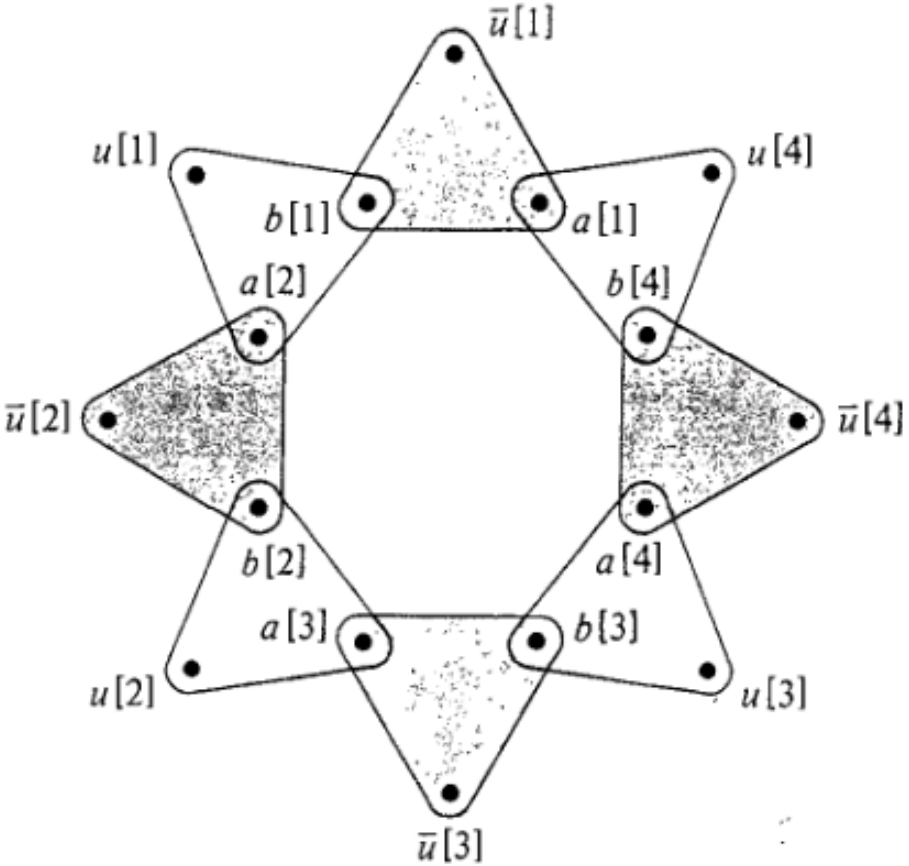
- How do we show membership in NP? Excercise
- For NP-hardness, let $C = \{c_1, \ldots, c_m\}$ and $U = \{u_1, \ldots, u_n\}$ be an instance of SAT.
- We construct the sets $|W| = |X| = |Y|$ and a set $M \subseteq W \times X \times Y$ s.t., M contains a matching iff C is sat.
- M will be partitioned into classes of triples, each serving a different purpose:
  - truth-setting and fan-out
  - satisfaction testing
  - garbage collection

NICTA

Truth-setting and fan-out component:

- Each component corresponds to one variable $u \in U$.
- Structure depends on number of clauses m.
- As in the previous proof, we have elements only used inside a component, not elsewhere:
  For some var u_i, we have $a_i[j] \in X$ and $b_i[j] \in Y$, where $1 \leq j \leq m$ as "internal" elements.
- Elements $u_i[j], \overline{u}_i[j]$ $(1 \leq j \leq m)$ are "external" and will occur in other triples, too.
- The triples of this component can, again, be divided into two sets:
  - $T_i^{\top} := \{(\overline{u}_i[j], a_i[j], b_i[j]) : 1 \leq j \leq m)\}$
  - $T_i^{\perp} := \{(u_i[j], a_i[j+1], b_i[j]) : 1 \leq j \leq m)\} \cup \{(u_i[m], a_i[1], b_i[m])\}$

3DM is NP-complete



Truth setting component T_i when $m = 4$ (subscripts have been deleted for simplicity). Either all the sets of T_i^t (the shaded sets) or all the sets of T_i^f (the unshaded sets) must be chosen, leaving uncovered all the $u_i[j]$ or all the $\bar{u}_i[j]$, respectively.

That is, any matching $M' \subseteq M$ includes m triples.

3DM is NP-complete

NICTA

Satisfaction testing component:

- Each component corresponds to one clause $c_j \in C$.
- Only two "internal" elements $s_1[j] \in X$ and $s_2[j] \in Y$.
- External elements are from $\{u_i[j], \overline{u}_i[j] \mid 1 \leq i \leq n\}$, depending on which occur in c_j.
- The set of triples making up this component is as follows:

$$C_j := \{(u_i[j], s_1[j], s_2[j]) \mid u_i \in c_j\} \cup \{(\overline{u}_i[j], s_1[j], s_2[j]) \mid \overline{u}_i \in c_j\}$$

- That is, any matching $M' \subseteq M$ has to contain exactly one triple from C_j.
- This is only possible if some $u_i[j]$ (or $\overline{u}_i[j]$) for $u_i \in c_j$ ($\overline{u}_i \in c_j$) does not occur in the triples in $T_i \cap M'$ (otherwise it wouldn't be a matching), which will be the case iff the truth setting determined by M' satisfies clause c_j.

NICTA

Garbage collection component:

- Basically deals with all the variables $u_i \in U$ whose value doesn't matter for the satisfiability of C (as we have already set them in the previous components).
- "Internal" elements:
  $g_1[k] \in X$, $g_2[k] \in Y$, where $1 \leq k \leq m(n-1)$.
- "External" elements: $u_i[j], \overline{u}_i[j] \in W$.
- Triples are of the form:

$$G := \{(u_i[j], g_1[k], g_2[k]), (\overline{u}_i[j], g_1[k], g_2[k])\},$$

  where we add all $1 \leq k \leq m(n-1), 1 \leq i \leq n, 1 \leq j \leq m$.
- That is, each pair $g_1[k], g_2[k]$ must be matched with a unique $u_i[j]$ (or $\overline{u}_i[j]$) that does not occur in any triples of $M' - G$. There are exactly $m(n-1)$ such uncovered elements.

3DM is NP-complete

NICTA

So, we constructed:

- $W := \{u_i[j], \overline{u}_i[j] \mid 1 \leq i \leq n, 1 \leq j \leq m\}$ (only the garbage collection added to W)
- $X := A \cup S_1 \cup G_1$, where
  - $A := \{a_i[j] \mid 1 \leq i \leq n, 1 \leq j \leq m\}$
  - $S_1 := \{s_1[j] \mid 1 \leq j \leq m\}$
  - $G_1 := \{g_1[j] \mid 1 \leq j \leq m(n-1)\}$
- $Y := B \cup S_2 \cup G_2$, where
  - $B := \{b_i[j] \mid 1 \leq i \leq n, 1 \leq j \leq m\}$
  - $S_2 := \{s_2[j] \mid 1 \leq j \leq m\}$
  - $G_2 := \{g_2[j] \mid 1 \leq j \leq m(n-1)\}$
- And by our careful choice of which elements to add to M, we now have: $M = (\bigcup_{i=1}^{n} T_i) \cup (\bigcup_{j=1}^{m} C_j) \cup G$

But does the construction serve its purpose, i.e., does M always contain a matching M'?

3DM is NP-complete

NICTA

(M contains a matching $\Rightarrow$ C is sat.):

- We have already (informally) convinced ourselves of that.

(C is sat. $\Rightarrow$ M contains a matching):

- Given $t : U \to \mathbb{B}$, and C is sat. by t, we extract/construct a suitable $M' \subseteq M$.
- For each clause $c_j \in C$, let $z_j \in \{u_i, \overline{u}_i \mid 1 \leq i \leq n\} \cap c_j$ be a literal that is set to $\top$ by t (one must exist to satisfy c_j).
- We then set:

$$M' := \left(\bigcup_{t(u_i)=\top} T_i^{\top}\right) \cup \left(\bigcup_{t(u_i)=\bot} T_i^{\bot}\right) \cup \left(\bigcup_{j=1}^{m} \{z_j[j], s_1[j], s_2[j])\}\right) \cup G'$$

where G' is a subset of G that contains all the $g_1[k], g_2[k]$ and the remaining $u_i[j]$ and $\overline{u}_i[j]$. We note that such a G' can always be chosen, and thus M' is a matching. $\square$

Some tricks to determine complexity

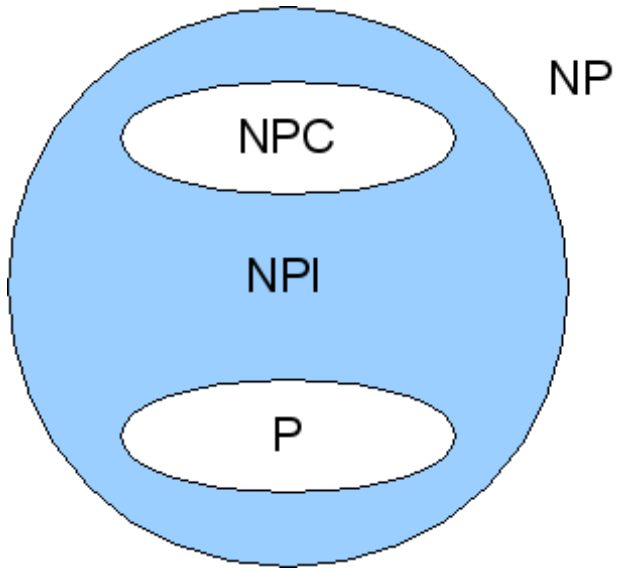
NICTA

- Try to constrain the problem (may not always work, e.g., 3DM).
- Fix a parameter (i.e., move it from the problem input to the problem description)

Beyond NP-completeness

NICTA

Recall our world-view:



- If we assume $P \neq NP$, then $NPC \cap P = \emptyset$.
- However, is it the case that, if $P \neq NP$, that $NPI = NPC \cup P$?

NICTA

Theorem (Ladner (1975))

Let B be a recursive language (i.e., recognisable by a DTM program that halts on all inputs) such that $B \notin P$. Then there exists a polynomial time recognisable language $D \in P$ such that the language $A = D \cap B$ does not belong to P, $A \propto B$, and yet $B \not\propto A$.

We apply this as follows:

- Let $B \in NPC$, so (if $P \neq NP$) $B \notin P$.
- Then A belongs to NP, because $D \in P$ and $B \in NP$.
- Ladner now says: $B \not\propto A$, so A cannot be NP-complete, and because $A \notin P$, A must be in NPI.
- That is, $NPI \neq \emptyset$ if $P \neq NP$.

Beyond NP-completeness

NICTA

A more concrete example:

- Suppose B is the Hamiltonian Cycle problem.
- Ladner now says: there exists a polynomial time recognisable set of graphs, such that Hamiltonian Cycle, when restricted to that set is neither an NP-complete nor a problem in P (assuming $P \neq NP$).
- WOW!

Beyond NP-completeness

NICTA

Natural question to ask:

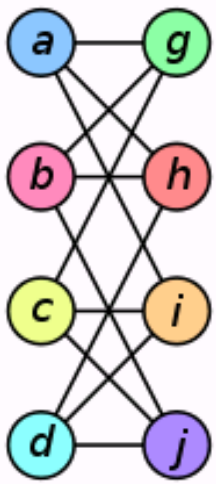
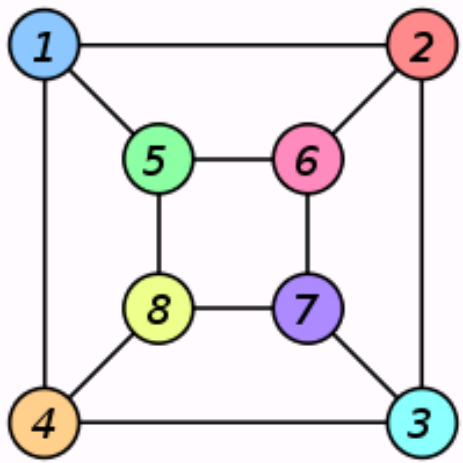
Do there exist real-world/natural problems that are in NPI?

- First, note that there is a heap of problems in NP, which people have failed to prove complete.
- All those are candidates.
- Some withstood the test of time better than others... (i.e., people failed to show NP-hardness for these, nor found a polynomial time algorithm)

NICTA

Definition (Graph Isomorphism)

INSTANCE: Graphs $G = (V, E)$ and $G' = (V', E')$.
QUESTION: Are G and G' isomorphic, that is, is there a one-to-one mapping $f : V(G) \to V(G')$ such that $(u, v) \in E$ iff $(f(u), f(v)) \in E'$?

Graph G	Graph H	An isomorphism between G and H
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$

Complementary classes

Take a problem Π for which you know the complexity class, and consider its complement, Π^c. Is Π^c in the same complexity class as Π?

Theorem

If there exists one NP-complete problem Π such that $\Pi^c \in NP$, then $NP = co\text{-}NP$.

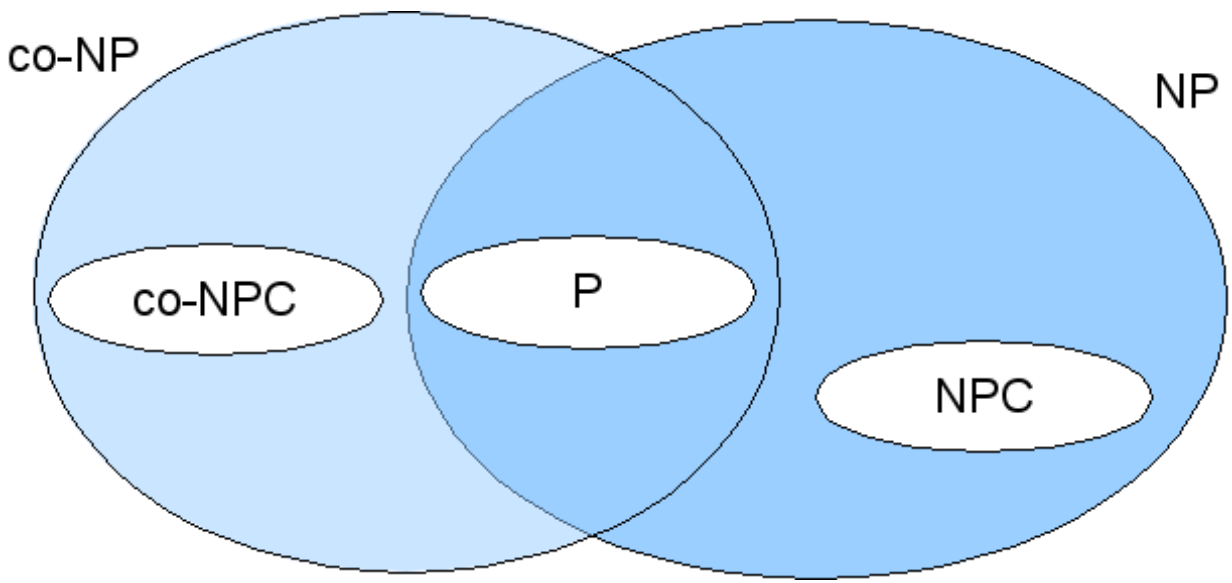
Proof.

Via the transitivity of polynomial problem reductions. $\square$

Complementary classes

NICTA

- Based on all evidence so far: co-NP $\neq$ NP.
- In fact if that is a theorem (co-NP $\neq$ NP), it would imply that $P \neq NP$.
- Note, we do not know whether co-NP $\neq$ NP nor whether $P \neq NP$. And more so, $P \neq NP$ does NOT imply, vice versa, that NP $\neq$ co-NP.
- Note further that, co-P $=$ P.



Complexity of enumeration

- SAT asks "Does there exist a satisying truth assignment?"
- We could also ask "How many such assignments exist?"
- What's the complexity of this question? Guess! Excercise!
- We do not want to see them all, but just the number of assignments!
- These are captured by a class $\#P$ (Valiant, 1977).
- $\#SAT \in \#P$, in fact $\#P$-complete.
- How does $\#SAT$ relate to the NP-completeness of SAT?
  - The counting problem is at least as hard as the underlying decision problem.
  - Even if $P = NP$, it is not clear whether counting the number of solutions for an NP-complete problem could be done in polymomial time.

Space complexity

NICTA

Note that

So far, we measured the complexity of a problem merely in terms of the time that was required to find an answer (by a deterministic, respectively nondeterministic machine).

A natural question to ask:

- How much space does an algorithm use?
- If a problem is solvable in polynomial time, is it solvable in polynomial space?
- What is the relation between time complexity and space complexity?

All problems in P, NP (and even #P) are solvable in polynomial space, i.e., $P \subseteq NP \subseteq PSpace$.

Space complexity

Do there exist problems solvable in polynomial space that cannot be solved in polynomial time? That is, does $PSpace = P$ hold?

Slightly more formal:

PSpace

PSpace is the class of languages that are decidable in polynomial space by a DTM, that is,

$$PSpace = \bigcup_k Space(n^k).$$

PSpace-completeness

A language L is PSpace-complete, if $L \in PSpace$ and for all $L' \in PSpace$, $L' \propto L$.

The world's first PSpace-complete problem

NICTA

Definition (QBF)

INSTANCE: A well-formed Boolean formula

$$F = (Q_1 x_1) \dots (Q_n x_n) E,$$

where E is a Boolean expression involving variables $x_1, \dots, x_n$, and each Q_i is either $\forall$ or $\exists$.
QUESTION: Is F true, i.e., $F = \top$?

Example

$$\forall x.\ \exists y.\ \exists z.\ (x \vee z) \wedge y$$

QBF is in PSpace

Proof: The following algorithm will solve an instance $I(\psi)$ (for brevity, paremeterised by the formula ψ) of QBF using at most polynomial space in the size of I.

Algorithm A**:** Let ϕ be the input to A (initially, $\phi \equiv \psi$):

- if ϕ contains no quantifiers, simply evaluate ϕ and "accept", if true, "reject" otherwise.
- if $\phi \equiv \exists x.\ \phi'$, then call $A(\phi'(\bot/x))$, and then $A(\phi'(\top/x))$, and return "yes" if either of the calls is "accept", "no" otherwise.
- if $\phi \equiv \exists x.\ \phi'$, then call $A(\phi'(\bot/x))$ and then $A(\phi'(\top/x))$, and return "yes" if both of the calls is "accept", "no" otherwise.

QBF is in PSpace

Observe:

- Depth of tree at most number of variables.
- At each level, we store the value of only one var.
- So total space is $O(m)$ where m is the number of vars.
- Hence, A runs in liner space.

□

QBF is PSpace-hard

Proof idea: Similar as with 3SAT:

Show that all $L \in PSpace$ are polytime reducible to QBF, i.e., we construct for all words $w \in L$ (accepted by a polyspace DTM M) a QBF formula ϕ such that $\phi = \top$ iff $w \in L$ (M accepts w).

To show how to construct ϕ, we solve a more general problem:

- Let c_1, c_2 be variables representing two configs of M.
- Let $t \in \mathbb{N}$, then $\phi_{c_1,c_2,t} = \top$ iff M can get from c_1 to c_2 in at most t steps.
- How do we construct a polynomial size $\phi_{c_1,c_2,t}$ when $t = exp(w)$?
- That is, the proof constructs a formula $\phi_{c_{start},c_{accept},T}$, where c_{start} is the initial and c_{accept} the final config of M, and T the max. number of steps (possibly exponentially many) taken by M to accept w.

QBF is PSpace-hard

NICTA

- If $t = 1$, we can easily construct $\phi_{c_1,c_2,t}$: either one of the configurations changes to the other in one step or it does not. Since M is deterministic, this presents no problem: Use Cook's construction.
- $t > 1$: We construct $\phi_{c_1,c_2,t}$ recursively:

$$\phi_{c_1,c_2,t} = \exists m_1.\ \phi_{c_1,m_1,t/2} \wedge \phi_{m_1,c_2,t/2},$$

where m_1 is a configuration in between c_1 and c_2, for which we then proceed recursively as well, etc. pp.

QBF is PSpace-hard

NICTA

Problem

We end up with a new formula for $\phi_{c_1,c_2,t}$ which is roughly of size t. (And t may be exponential.)

Solution

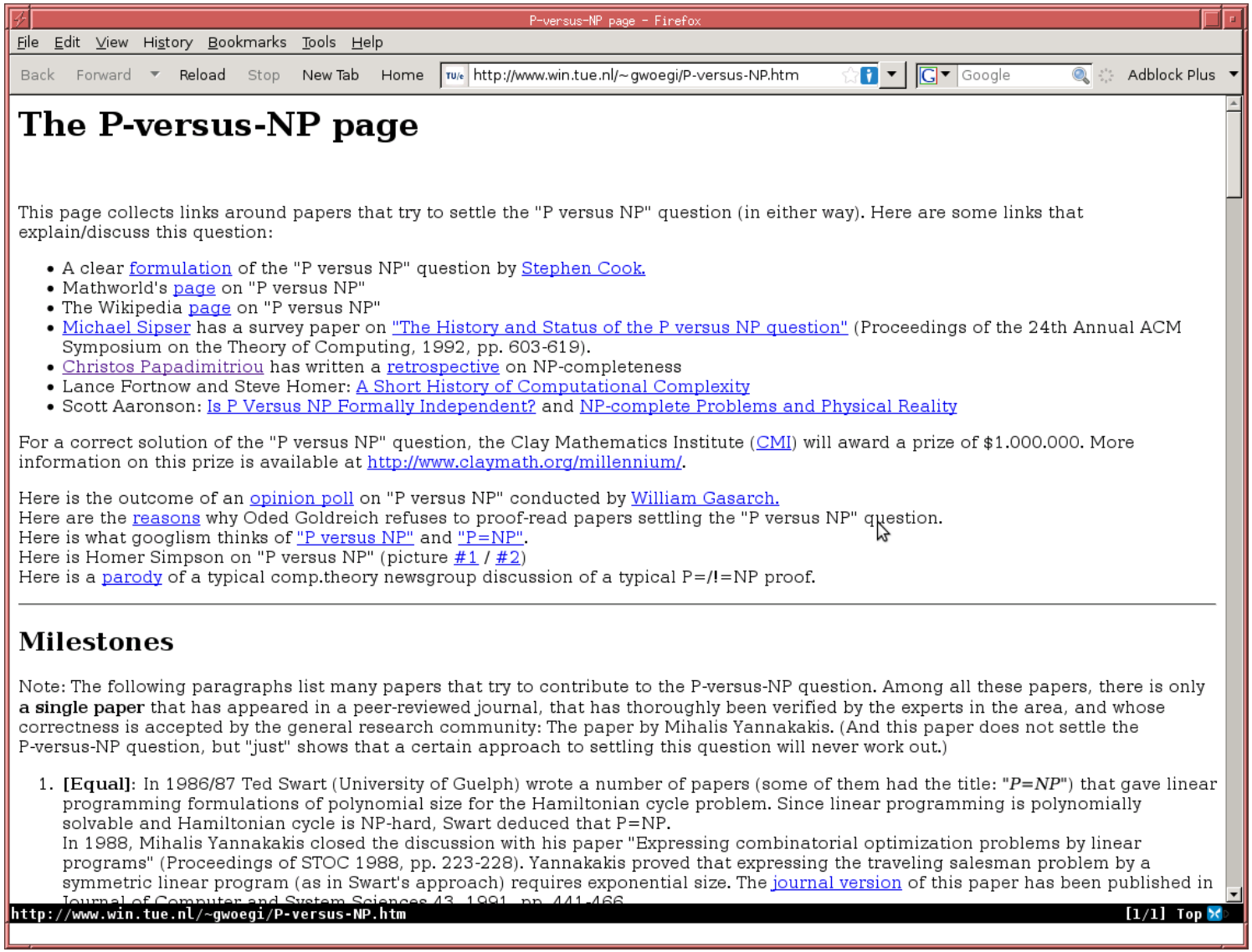
Use an $\forall$-quantified variant of the formula:

$$\exists m_1.\ \forall c_3, c_4.\ (c_3 = c_1 \wedge c_4 = m_1 \vee c_3 = m_1 \wedge c_4 = c_2) \rightarrow \phi_{c_3,c_4,t/2}$$

This yields a polynomial size formula, and thus transformation. $\square$

P vs NP is solved!

NICTA



PSpace-completeness and logic

Theorem of Savitch (1970)

Theorem

$NSpace(f(n)) \subseteq DSpace(f(n^2))$

That is, if a NDTM can solve a problem using space $f(n)$, then a DTM can solve the problem using space $f(n^2)$.

Proof.

Via a configuration-reachability, similar as in the QBF proof. □

Corollary

PSpace = NPSpace, ExpSpace = NExpSpace, etc.
(but assumedly, P $\neq$ NP)

. . . and also co-PSpace = PSpace (but assumedly, co-NP $\neq$ NP)

Linear-time temporal logic (LTL)

NICTA

Let

- AP be a non-empty, finite set of propositional symbols,
- $\Sigma := 2^{AP}$ be an alphabet,
-

LTL (Pnueli, 1977) formulae are given by the following syntax:

$$\varphi ::= \mathit{true} \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi U \varphi \mid X\varphi \quad (p \in AP),$$

Journal of the ACM:

"The JACM frequently receives submissions purporting to solve a long-standing open problem in complexity theory, such as the P/NP problem. Such submissions tax the voluntary editorial and peer-reviewing resources used by the JACM, by requiring the review process to identify the errors in them. The JACM remains open to the possibility of eventual resolution of P/NP and related questions, and continues to welcome submissions on the subject. However, to mitigate the burden of repeated resubmissions of incremental corrections of errors identified during editorial review, the JACM has adopted the following policy:
No author may submit more than one paper on the P/NP or related long-standing questions in complexity theory in any 24 month period, except by invitation of the Editor-in-Chief. This applies to resubmissions of previously rejected manuscripts."