NP-completeness

Victor Eijkhout

Notes for CS 594 – Fall 2004

Complexity theory

- What is the running time of an algorithm as function of its input?
- What is the memory need of an algorithm as function of its input?
- Best, worst, average, expected.
- Notation: $f(n) = O(g(n))$, $f(n) = \Theta(g(n))$

Tractable and intractable

- Tractable: polymomial complexity

$$f(n) = O(n^k) \quad \text{some } k$$

- Intractable: exponential complexity

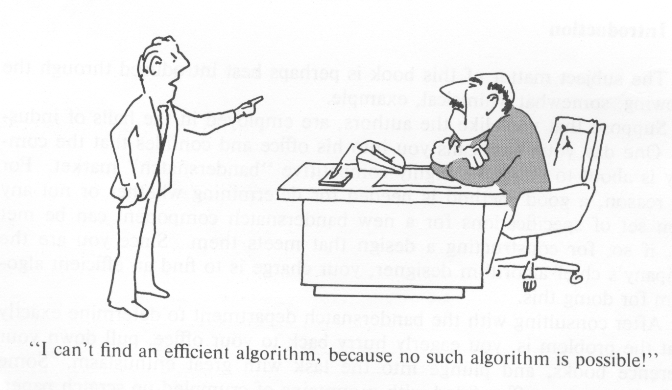
$$f(n) = O(2^n) \quad \text{or something like that}$$

Intractable problems

- They exist

Intractable problems

- They exist
- What do you tell your boss?



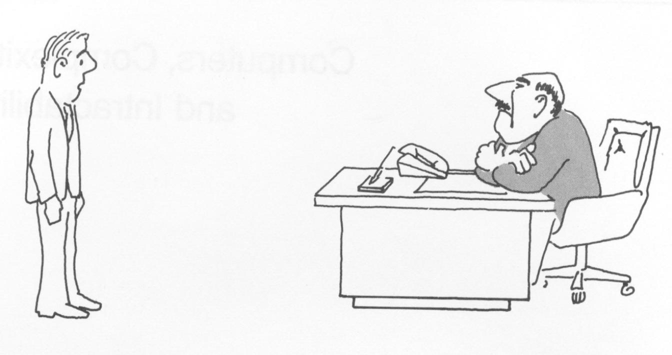
"I can't find an efficient algorithm, because no such algorithm is possible!"

Bounds versus reality

- Ideal: algorithm and bound are similar
- NP: all known algorithms are intractable, but not provably

Bounds versus reality

- Ideal: algorithm and bound are similar
- NP: all known algorithms are intractable, but not provably
- Wrong way to approach your boss:



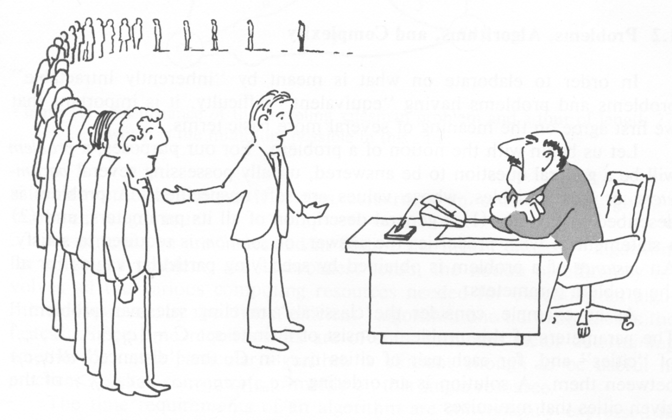
"I can't find an efficient algorithm, I guess I'm just too dumb."

Some consolation

- Problems in NP are equivalent: solve one, solve them all
- You're not alone: no one knows an efficient algorithm

Some consolation

- Problems in NP are equivalent: solve one, solve them all
- You're not alone: no one knows an efficient algorithm
- What to tell your boss:



"I can't find an efficient algorithm, but neither can all these famous people."

Decision problems

Optimization vs decision

- Many problems are optimization type:
- traveling salesman: what is the shortest route
- For complexity analysis more convenient decision problem
- translation needed: given a bound B, is there a route shorter than B

Language of a problem

- For a problem, there are 'instances'
- example: in traveling salesman problem, set of numbered cities $\{(i, c_i)\}_i$

Language of a problem

- For a problem, there are 'instances'
- example: in traveling salesman problem, set of numbered cities $\{(i, c_i)\}_i$
- 'Yes set' of a problem: Y_Π contains the instances that give a 'yes' decision
- Encoding e gives 'language of a problem':

$$L[\Pi, e] = \{\text{the instances in } Y_\Pi \text{ encoded under } e\}$$

Language of a problem

- For a problem, there are 'instances'
- example: in traveling salesman problem, set of numbered cities $\{(i, c_i)\}_i$
- 'Yes set' of a problem: Y_Π contains the instances that give a 'yes' decision
- Encoding e gives 'language of a problem':

$$L[\Pi, e] = \{\text{the instances in } Y_\Pi \text{ encoded under } e\}$$

- for traveling salesman: ordered list of cities $\langle c_1, c_2, \ldots\rangle$

Turing machines

Accepting strings

- A Turing machine can halt in 'yes' state q_y, 'no' state q_n, or not halt.
- The TM *accepts* a string if it halts in q_y
- The *language* is the set of all accepted strings

'Solving' a problem

A Deterministic Turing Machine (DTM) M *solves* a problem Π (equivalently: recognizes $L[\Pi, e]$) iff

- The DTM for all strings over the input alphabet
- $L_M = L[\Pi, e]$

'Solving' a problem

A Deterministic Turing Machine (DTM) M *solves* a problem Π (equivalently: recognizes $L[\Pi, e]$) iff

- The DTM for all strings over the input alphabet
- $L_M = L[\Pi, e]$
- Example: traveling salesman problem for network of cities and bound B,
- M always returns yes or no
- input is list of cities
- M returns yes if list satisfies bound

Solution checking vs generating

- In the traveling salesman problem, ‘solving’ means checking a solution

Solution checking vs generating

- In the traveling salesman problem, 'solving' means checking a solution
- Other example: primality testing
- Input is number, output is yes/no (this number is prime / not prime)
- $\Rightarrow$ more intuitive notion of solving

Complexity classes

Class P

- Definition in terms of languages:

$$P = \{L : \text{there is DTM that recognizes } L \text{ in polynomial time}\}$$

Class P

- Definition in terms of languages:

$$P = \{L : \text{there is DTM that recognizes } L \text{ in polynomial time}\}$$

- in terms of problems:

$$\begin{aligned}\Pi \in P \quad &\equiv \quad L[\Pi, e] \in P \quad \text{for some encoding } e \\ &\equiv \quad \text{there is a polynomial time DTM that recognizes } L[\Pi, e]\end{aligned}$$

Class P

- Definition in terms of languages:

$$P = \{L : \text{there is DTM that recognizes } L \text{ in polynomial time}\}$$

- in terms of problems:

$$\begin{aligned} \Pi \in P &\equiv L[\Pi, e] \in P \quad \text{for some encoding } e \\ &\equiv \text{there is a polynomial time DTM that recognizes } L[\Pi, e] \end{aligned}$$

- Interpretation:
  $\Pi \in P$ means is a polynomial time Turing machine M
  that recognizes Y_Π
  for strings not in Y_Π: it halts in state q_N

Class *NP*

- *Certificate*: guessed solution
  (e.g. list of cities for traveling salesman)
- Non-Deterministic Turing Machine:
  first generate certificate by ϵ-moves
  then check correctness

Class *NP*

- *Certificate*: guessed solution
  (e.g. list of cities for traveling salesman)
- Non-Deterministic Turing Machine:
  first generate certificate by ϵ-moves
  then check correctness
- Let NDTM M, then L_M is the set of decision problems for which M can generate a certificate
- Certificate exists $\Leftrightarrow$ decision problem has 'true' answer

Class *NP*

- *Certificate*: guessed solution
  (e.g. list of cities for traveling salesman)
- Non-Deterministic Turing Machine:
  first generate certificate by ϵ-moves
  then check correctness
- Let NDTM M, then L_M is the set of decision problems for which M can generate a certificate
- Certificate exists $\Leftrightarrow$ decision problem has 'true' answer
- $\Pi \in NP$ iff there is NDTM M that recognizes $L[\Pi]$ in polynomial time

Class *NP*

- *Certificate*: guessed solution (e.g. list of cities for traveling salesman)
- Non-Deterministic Turing Machine: first generate certificate by ϵ-moves then check correctness
- Let NDTM M, then L_M is the set of decision problems for which M can generate a certificate
- Certificate exists $\Leftrightarrow$ decision problem has 'true' answer
- $\Pi \in NP$ iff there is NDTM M that recognizes $L[\Pi]$ in polynomial time
- Π is in NP iff there is a polynomial time function $A(\cdot,\cdot)$ such that

$$w \in Y_\Pi \Leftrightarrow \exists_C : A(w, C) = \text{true}.$$

Examples

- Trivially in P: given a, b, n, check whether $a \times b = n$

Examples

- Trivially in P: given a, b, n, check whether $a \times b = n$
- Non-trivially in P: given n, check whether a, b exist

Examples

- Trivially in P: given a, b, n, check whether $a \times b = n$
- Non-trivially in P: given n, check whether a, b exist
- Not in P: actually find a, b

$$\Theta(\exp((n \cdot 64/9)^{1/3})(\log n)^{2/3})$$

Examples

- Trivially in P: given a, b, n, check whether $a \times b = n$
- Non-trivially in P: given n, check whether a, b exist
- Not in P: actually find a, b

$$\Theta(\exp((n \cdot 64/9)^{1/3})(\log n)^{2/3})$$

- Provably exponential time: best move in chess

Examples

- Trivially in P: given a, b, n, check whether $a \times b = n$
- Non-trivially in P: given n, check whether a, b exist
- Not in P: actually find a, b

$$\Theta(\exp((n \cdot 64/9)^{1/3})(\log n)^{2/3})$$

- Provably exponential time: best move in chess
- Unclear: traveling salesman

NP-completeness

Definition of Polynomial transformation

- L_1 and L_2 languages over alphabets $\sum_1^*$ and $\sum_2^*$
- $f : \sum_1^* \mapsto \sum_2^*$ is called polynomial transformation of problem 1 into problem 2 if
  - There is DTM that computes $f(x)$ in time $T_f(x) \leq p_f(|x|)$ for polynomial p_f, and
  - For all $x \in \sum_1^*$, $x \in L_1$ iff $f(x_1) \in L_2$.
- 'many-to-one polynomial transformation'

Relations

- Let f polynomial transformation from L_1 to L_2,
- then

$$L_2 \in P \Rightarrow L_1 \in P$$

Relations

- Let f polynomial transformation from L_1 to L_2,
- then
$$L_2 \in P \Rightarrow L_1 \in P$$
- Proof: Let $M_2 : L_2 \to \{0,1\}$ DTM that recognizes L_2, then $M_2 \circ f$ is DTM that recognizes L_1,

Relations

- Let f polynomial transformation from L_1 to L_2,
- then
$$L_2 \in P \Rightarrow L_1 \in P$$
- Proof: Let $M_2 : L_2 \to \{0,1\}$ DTM that recognizes L_2, then $M_2 \circ f$ is DTM that recognizes L_1,
- composite recognizer runs in time
$$T_{M_2 \circ f}(x) \leq p_{T_2}(|p_f(|x|)|)$$
which is polynomial

Relations'

- Notation: L_1 transforms to L_2 (in polynomial time):

$$L_1 \leq L_2$$

- (suggested reading: L_1 is easier than L_2.)

Relations'

- Notation: L_1 transforms to L_2 (in polynomial time):

$$L_1 \leq L_2$$

- (suggested reading: L_1 is easier than L_2.)
- Transitivity:

$$L_1 \leq L_2 \wedge L_2 \leq L_3 \Rightarrow L_1 \leq L_3,$$

NP-complete

Definition

- L is NP-complete if
  - $L \in NP$, and
  - for all $L' \in NP$: $L' \leq L$

Definition

- L is NP-complete if
  - $L \in NP$, and
  - for all $L' \in NP$: $L' \leq L$
- First condition: NDTM has to halt.
- Without that: NP-hard
- Example: halting problem

Relations”

- If $L_1, L_2 \in NP$, L_1 is NP-complete, and $L_1 \leq L_2$, then L_2 is NP-complete.

Relations”

- If $L_1, L_2 \in NP$, L_1 is NP-complete, and $L_1 \leq L_2$, then L_2 is NP-complete.
- Proof: need to check $\forall_{L_2 \in NP} : L' \leq L_2$
- L_1 is NP-complete, so $L' \leq L_1$.
  Now use transitivity

Proof of NP-completeness

Bootstrap problem

- Have one NP-complete problem, construct translation, have another
- Where to begin? How to check that every other problem is easier?

Bootstrap problem

- Have one NP-complete problem, construct translation, have another
- Where to begin? How to check that every other problem is easier?
- Strategy: for NP-complete problem, NDTM exists
- encode that NDTM in some specific problem

Bootstrap problem

- Have one NP-complete problem, construct translation, have another
- Where to begin? How to check that every other problem is easier?
- Strategy: for NP-complete problem, NDTM exists
- encode that NDTM in some specific problem
- First done for SAT, boolean satisfiability
  Cook, 1971

Satisfiability

- Problem statement: given boolean variables $x_1 \ldots x_n$ and a logical formula $F(x_1, \ldots, x_n)$
  is there is an assignment $x_i \mapsto \{0, 1\}$ such that $F(x_1, \ldots) = 1$.

Satisfiability

- Problem statement: given boolean variables $x_1 \ldots x_n$ and a logical formula $F(x_1, \ldots, x_n)$
  is there is an assignment $x_i \mapsto \{0, 1\}$ such that $F(x_1, \ldots) = 1$.
- Examples: $x_1 \vee \neq x_1$ is always true;
  $x_1 \wedge \neq x_1$ is always false
  $x_1 \wedge \neq x_2$ is only true for $(x_1 = T, x_2 = F)$.
- second example not satisfiable

Satisfiability

- Problem statement: given boolean variables $x_1 \ldots x_n$ and a logical formula $F(x_1, \ldots, x_n)$
  is there is an assignment $x_i \mapsto \{0, 1\}$ such that $F(x_1, \ldots) = 1$.
- Examples: $x_1 \vee \neq x_1$ is always true;
  $x_1 \wedge \neq x_1$ is always false
  $x_1 \wedge \neq x_2$ is only true for $(x_1 = T, x_2 = F)$.
- second example not satisfiable
- $\textsc{Sat}$ is in NP

Transformations into SAT

- Let Π be NP-complete; M an NDTM that solves it
- construct logical formula such that

$$\text{formula true} \Leftrightarrow \text{successful TM run}$$

The Turing machine

$$M = \langle Q, s, \Sigma, F, \delta \rangle$$

where

Q is the set of states, and $s \in Q$ the initial state,

Σ the alphabet of tape symbols,

$F \subset Q$ the set of accepting states, and

$\delta \subset Q \times \Sigma \times Q \times \Sigma \times \{-1, +1\}$ the set of transitions,

Assume M accepts or rejects instances in time $p(n)$ where n is the size of the instance and $p(\cdot)$ is a polynomial.

Variables

For $q \in Q$, $-p(n) \leq i \leq p(n)$, $j \in \Sigma$, $0 \leq k \leq p(n)$:

Variables	Intended interpretation	How many
T_{ijk}	True iff tape cell i contains symbol j at step k of the computation	$O(p(n)^2)$
H_{ik}	True iff the M's read/write head is at tape cell i at step k of the computation.	$O(p(n)^2)$
Q_{qk}	True iff M is in state q at step k of the computation.	$O(p(n))$

Expression

Conjunction of whole bunch of clauses
$(-p(n) \leq i \leq p(n),\ j \in \Sigma$, and $0 \leq k \leq p(n))$
Initial conditions

For all:	Add the clauses	Interpretation	How many clauses?
Tape cell i of the input I contains symbol j.	T_{ij0}	Initial contents of the tape.	$O(p(n))$
	Q_{s0}	Initial state of M	$O(1)$
	H_{00}	Initial position of read/write head.	$O(1)$

Constraint clauses

For all:	Add the clauses	Interpretation	How many clauses?
symbols $j \neq j'$	$T_{ijk} \rightarrow \neg T_{ij'k}$	One symbol per tape cell.	$O(p(n)^2)$
states $q \neq q'$	$Q_{qk} \rightarrow \neg Q_{q'k}$	Only one state at a time.	$O(p(n))$
cells $i \neq i'$	$H_{ik} \rightarrow \neg H_{i'k}$	Only one head position at a time.	$O(p(n))$

Turing machine clauses

For all:	Add the clauses	Interpretation	How many clauses?
i, j, k	$T_{ijk} = T_{ij(k+1)} \vee H_{ik}$	Tape remains unchanged unless written.	$O(p(n)^2)$
$f \in F$	The disjunction of the clauses $Q_{f,p(n)}$	Must finish in an accepting state.	$O(1)$

Transition table clauses

For all:	Add the clauses	Interpretation	How many clauses?
$(q, \sigma, q', \sigma', d) \in \delta$	The disjunction of the clauses $(H_{ik} \wedge Q_{qk} \wedge T_{i\sigma k}) \rightarrow (H_{(i+d)(k+1)} \wedge Q_{q'(k+1)} \wedge T_{i\sigma'(k+1)})$	Possible transitions at computation step k when head is at position i.	$O(p(n)^2)$

Equivalence

If there is an accepting computation for M on input I,
then B is satisfiable, by assigning T_{ijk}, H_{ik} and Q_{ik}

If B is satisfiable,
then there is an accepting computation for M on input I:
follows the steps indicated by the assignments to the variables.

Polynomial time

- $O(p(n)^2)$ Boolean variables, each encodable in space $O(\log p(n))$
- Number of clauses $O(p(n)^2)$
- Total size of B is $O((\log p(n))p(n)^2)$