Advanced Topics in Theoretical Computer Science

Part 5: Complexity (Part 1)

24.01.2013

Viorica Sofronie-Stokkermans

Universität Koblenz-Landau

e-mail: sofronie@uni-koblenz.de

Contents

- Recall: Turing machines and Turing computability
- Register machines (LOOP, WHILE, GOTO)
- Recursive functions
- The Church-Turing Thesis
- Computability and (Un-)decidability
- Complexity
- Brief outlook: other computation models, e.g. Büchi Automata

Motivation

Assume you are employed as software designer.

One day, your boss calls you into his office and and tells you that the company is about to enter a very competitive market which solved problem X. Your charge is to find an efficient algorithm for solving this problem.

Motivation

Assume you are employed as software designer.

One day, your boss calls you into his office and and tells you that the company is about to enter a very competitive market which needs problem X. Your charge is to find an efficient algorithm for solving this problem.

You certainly do not want to return to the office of your boss, after working hard on the problem, and report:

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

Motivation

It would be much better if you could prove that problem X is inherently intractable, i.e. that no algorithm could possibly solve it quickly.
You could then tell your boss:

> "I can't find an efficient algorithm, because no such algorithm exists."

Motivation

It would be much better if you could prove that problem X is inherently intractable, i.e. that no algorithm could possibly solve it quickly. You could then tell your boss:

"I can't find an efficient algorithm, because no such algorithm exists."

Unfortunately, proving inherent intractability can be just as hard as finding efficient algorithms. However, we will see that you can often answer:

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

NP complete problems, for instance, are widely recognized to be difficult, but no expert could prove until now that they can/cannot be solved quickly.

Motivation

Goals:

- Define formally time and space complexity
- Define a family of "complexity classes": P, NP, PSPACE, ...
- Study the links between complexity classes
- Learn how to show that a problem is in a certain complexity class

  Reductions to problems known to be in the complexity class
- Closure of complexity classes

We will give examples of problems from various areas and study their complexity.

Complexity

- Recall:
  - Big O notation
  - The structure of PSPACE
  - Complete problems; hard problems
  - Examples

Big O notation

Definition. Let $h, f : \mathbb{N} \to \mathbb{R}$ functions.
The function h is in the class $O(f)$ iff there exists $c \in \mathbb{R}$, $c > 0$ and there exists $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$ $|h(n)| \leq c|f(n)|$.

Notation: $f \in O(h)$, sometimes also $f(n) \in O(h(n))$;
by abuse of notation denoted also by $f = O(h)$)

Examples:

$5n + 4 \in O(n)$

$5n + n^2 \notin O(n)$

$\begin{pmatrix} n \\ 2 \end{pmatrix} = \frac{n(n-1)}{2} \in O(n)$

Let p be a polynomial of degree m. Then $p(n) \in O(n^m)$

Big O notation

Computation rules for O

- $f \in O(f)$
- $c \cdot O(f) = O(f)$
- $O(O(f)) = O(f)$
- $O(f) \cdot O(g) = O(f \cdot g)$
- $O(f \cdot g) = |f| O(g)$
- If $|f| \leq |g|$ then $O(f) \subseteq O(g)$

Lemma. The following hold:

- $\forall d > 0,\ n^{d+1} \notin O(n^d)$
- $\forall r > 1 \forall d (r^n \notin O(n^d)$ and $n^d \in O(r^n))$

Complexity

Types of complexity

- Time complexity
- Space complexity

DTIME and NTIME

Basic model: k-DTM or k-NTM M (one tape for the input)

If M makes for every input word of length n at most $T(n)$ steps, then M is $T(n)$-time bounded.

In this case, the language accepted by M has time complexity $T(n)$; (more precisely $\max(n+1, T(n))$.

Definition ($NTIME(T(n)), DTIME(T(n))$)

- $DTIME(T(n))$ class of all languages accepted by $T(n)$-time bounded DTMs.
- $NTIME(T(n))$ class of all languages accepted by $T(n)$-time bounded NTMs.

DSPACE and NSPACE

Basic model: k-DTM or k-NTM M with special tape for the input (is read-only) + k storage tapes (offline DTM) $\mapsto$ needed if $S(n)$ sublinear

If M needs, for every input word of length n, at most $S(n)$ cells on the storage tapes then M is $S(n)$-space bounded.

The language accepted by M has space complexity $S(n)$;
(more precisely $\max(1, S(n))$).

Definition ($NSPACE(T(n))$, $DSPACE(T(n))$)

- $DSPACE(S(n))$ class of all languages accepted by $S(n)$-space bounded DTMs.
- $NSPACE(S(n))$ class of all languages accepted by $S(n)$-space bounded NTMs.

Example

To which time/space complexity does the following language belong:

$$L_{\text{mirror}} = \{wcw^R \mid w \in \{0,1\}^*\}$$

Example

To which time/space complexity does the following language belong:

$$L_{\text{mirror}} = \{wcw^R \mid w \in \{0,1\}^*\}$$

Time: $DTIME(n+1)$: copy input to the right of c in reverse order. When c is found, the rest is compared with the copy of w on the tape.

Space: $DSPACE(n)$: previous DTM

Example

To which time/space complexity does the following language belong:

$$L_{\text{mirror}} = \{wcw^R \mid w \in \{0,1\}^*\}$$

Time: $DTIME(n+1)$: copy input to the right of c in reverse order. When c is found, the rest is compared with the copy of w on the tape.

Space: $DSPACE(n)$: previous DTM

Even better $DSPACE(log(n))$: use two tapes as binary counters (length of word before/after c). Remember: the definition of DSPACE does not count the space used on the input tape.

Questions

Time: Is any language in $DTIME(f(n))$ decided by some DTM?

Space: Is any language in $DSPACE(f(n))$ decided by some DTM?

The functions f are usually very simple functions; in particular they are all computable.

We will consider e.g. powers $f(n) = n^k$.

Questions

Time: Is any language in $DTIME(f(n))$ decided by some DTM?

Space: Is any language in $DSPACE(f(n))$ decided by some DTM?

The functions f are usually are very simple functions; in particular they are all computable.

We will consider e.g. powers $f(n) = n^k$.

Time/Space: What about $NTIME(f(n))$, $NSPACE(f(n))$

Time vs. Space: What are the links between $DTIME(f(n))$, $DSPACE(f(n))$, $NTIME(f(n))$, $NSPACE(f(n))$

Questions

Time bounded What does it mean that a DTM makes at most n steps?

Strictly speaking, after n steps it should halt or hang.

Halt? Input is accepted

Hang? DTM on band which is infinite on both sides cannot hang!

Questions

Time bounded What does it mean that a DTM makes at most n steps?

Strictly speaking, after n steps it should halt or hang.

Halt? Input is accepted

Hang? DTM on band which is infinite on both sides cannot hang!

Stop after n steps

Stop: We understand the following under M makes at most n steps:

- It halts (and accepts the input) within n steps
- It hangs (and does not accept the input) within n steps
- It halts after n steps, but not in halting mode, so it does not accept the input.

Answers

Answers (Informally)

Time: Every language from $DTIME(f(n))$ is decidable:

for an input of length n we wait as long as the value $f(n)$.

If until then no answer "YES" then the answer is "NO".

Space: Every language from $DSPACE(f(n))$ is decidable:

There are only finitely many configurations. We write all configurations.

If the TM does not halt then there is a loop. This can be detected.

Answers

Answers (Informally)

NTM vs. DTM: Clearly, $DTIME(f(n)) \subseteq NTIME(f(n))$ and

$$DSPACE(f(n)) \subseteq NSPACE(f(n))$$

If we try to simulate an NTM with a DTM we may need exponentially more time. Therefore:

$$NTIME(f(n)) \subseteq DTIME(2^{h(n)}) \text{ where } h \in O(f).$$

For the space complexity we can show that:

$$NSPACE(f(n)) \subseteq DSPACE(f^2(n))$$

Answers

Answers (Informally)

Time vs. Space: Clearly, $DTIME(f(n)) \subseteq DSPACE(f(n))$ and

$$NTIME(f(n)) \subseteq NSPACE(f(n))$$

$DSPACE(f(n))$, $NSPACE(f(n))$ are much larger.

Question

What about constant factors?

Question

What about constant factors?

Constant factors are ignored. Only the rate of growth of a function in complexity classes is important.

Theorem.
For every $c \in \mathbb{R}^+$ and every storage function $S(n)$ the following hold:

- $DSPACE(S(n)) = DSPACE(cS(n))$
- $NSPACE(S(n)) = NSPACE(cS(n))$

Question

What about constant factors?

Constant factors are ignored. Only the rate of growth of a function in complexity classes is important.

Theorem.
For every $c \in \mathbb{R}^+$ and every storage function $S(n)$ the following hold:

- $DSPACE(S(n)) = DSPACE(cS(n))$
- $NSPACE(S(n)) = NSPACE(cS(n))$

Proof. One direction is trivial. The other direction can be proved by representing a fixed amount $r > \frac{2}{c}$ of neighboring cells on the tape as a new symbol.

The states of the new machine simulate the movements of the read/write head as transitions. For r-cells of the old machine we use only two: in the most unfavourable case when we go from one block to another.

Time acceleration

Theorem For every $c \in \mathbb{R}^+$ and every time function $T(n)$ with $\lim_{n\to\infty} \frac{T(n)}{n} = \infty$ the following hold:

- $DTIME(T(n)) = DTIME(cT(n))$
- $NTIME(T(n)) = NTIME(cT(n))$

Time acceleration

Theorem For every $c \in \mathbb{R}^+$ and every time function $T(n)$ with $\lim_{n\to\infty} \frac{T(n)}{n} = \infty$ the following hold:

- $DTIME(T(n)) = DTIME(cT(n))$
- $NTIME(T(n)) = NTIME(cT(n))$

Proof. One direction is trivial. The other direction can be proved by representing a fixed amount $r > \frac{4}{c}$ of neighboring cells on the tape as a new symbol.

The states of the new machine simulate also now which symbol and which position the read/write head of the initial machine has. When the machine is simulated the new machine needs to make 4 steps instead of r: 2 in order to write on the new fields and 2 in order to move the head on the new field and then back on the old (in the worst case).

Big O notation

Theorem: Let T be a time function with $\lim_{n\to\infty} \frac{T(n)}{n} = \infty$ and S a storage function.

(a) If $f(n) \in O(T(n))$ then $DTIME(f(n)) \subseteq DTIME(T(n))$.

(b) If $g(n) \in O(S(n))$ then $DSPACE(g(n)) \subseteq DSPACE(S(n))$.

P, NP, PSPACE

Definition

$$\begin{array}{rcl} P & = & \bigcup_{i \geq 1} DTIME(n^i) \\ NP & = & \bigcup_{i \geq 1} NTIME(n^i) \\ PSPACE & = & \bigcup_{i \geq 1} DSPACE(n^i) \end{array}$$

P, NP, PSPACE

Definition

$$\begin{aligned} P &= \bigcup_{i\geq 1} DTIME(n^i) \\ NP &= \bigcup_{i\geq 1} NTIME(n^i) \\ PSPACE &= \bigcup_{i\geq 1} DSPACE(n^i) \end{aligned}$$

Lemma $NP \subseteq \bigcup_{i\geq 1} DTIME(2^{O(n^d)})$

Proof: Follows from the fact that if L is accepted by a $f(n)$-time bounded NTM then L is accepted by an $2^{O(f(n))}$-time bounded DTM, hence for every $d \geq 1$ we have:

$$NTIME(n^d) \subseteq DTIME(2^{O(n^d)})$$

P, NP, PSPACE

$$
\begin{array}{lcl}
P & = & \bigcup_{i \geq 1} DTIME(n^i) \\
NP & = & \bigcup_{i \geq 1} NTIME(n^i) \\
PSPACE & = & \bigcup_{i \geq 1} DSPACE(n^i) \\
NP & \subseteq & \bigcup_{i \geq 1} DTIME(2^{O(n^d)})
\end{array}
$$

Intuition

- Problems in P can be solved efficiently; those in NP can be solved in exponential time
- PSPACE is a very large class, much larger that P and NP.

Complexity classes for functions

Definition

A function $f : \mathbb{N} \to \mathbb{N}$ is in P if there exists a DTM M and a polynomial $p(n)$ such that for every n the value $f(n)$ can be computed by M in at most $p(\text{length}(n))$ steps.

Here $\text{length}(n) = \log(n)$: we need $\log(n)$ symbols to represent (binary) the number n.

The other complexity classes for functions are defined in an analogous way.

Relationships between complexity classes

Question:

Which are the links between the complexity classes P, NP and PSPACE?

Relationships between complexity classes

Question:

Which are the links between the complexity classes P, NP and PSPACE?

$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE}$

Complexity classes

How do we show that a certain problem is in a certain complexity class?

Complexity classes

How do we show that a certain problem is in a certain complexity class?

Reduction to a known problem

We need one problem we can start with! SAT

Complexity classes

Can we find in NP problems which are the most difficult ones in NP?

Complexity classes

Can we find in NP problems which are the most difficult ones in NP?

Answer

There are various ways of defining "the most difficult problem".

They depend on the notion of reducibility which we use.

For a given notion of reducibility the answer is YES.

Such problems are called complete in the complexity class with respect to the notion of reducibility used.

Reduction

Definition (Polynomial time reducibility)
Let L_1, L_2 be languages.
L_2 is polynomial time reducible to L_1 (notation: $L_2 \preceq_{\text{pol}} L_1$)
if there exists a polynomial time bounded DTM, which for every input w computes an output $f(w)$ such that

$$w \in L_2 \text{ if and only if } f(w) \in L_1$$

Reduction

Lemma (Polynomial time reduction)

- Let L_2 be polynomial time reducible to L_1 ($L_2 \preceq_{\text{pol}} L_1$). Then:

  If $L_1 \in NP$ then $L_2 \in NP$.

  If $L_1 \in P$ then $L_2 \in P$.

- The composition of two polynomial time reductions is again a polynomial time reduction.

Reduction

Lemma (Polynomial time reduction)

- Let L_2 be polynomial time reducible to L_1 ($L_2 \preceq_{\text{pol}} L_1$). Then:

  If $L_1 \in NP$ then $L_2 \in NP$.

  If $L_1 \in P$ then $L_2 \in P$.

- The composition of two polynomial time reductions is again a polynomial time reduction.

Proof: Assume $L_1 \in P$. Then there exists $k \geq 1$ such that L_1 is accepted by n^k-time bounded DTM M_1.

Since $L_2 \preceq_{\text{pol}} L_1$ there exists a polynomial time bounded DTM M_2, which for every input w computes an output $f(w)$ such that $w \in L_2$ if and only if $f(w) \in L_1$.

Let $M_2 = M_f M_1$. Clearly, M_2 accepts L_2. We have to show that M_2 is polynomial time bounded. $w \mapsto M_f$ computes $f(w)$ (pol.size) $\mapsto M_1$ decides if $f(w) \in L_1$ (polynomially many steps)

NP

Theorem (Characterisation of NP)
A language L is in NP if and only if there exists a language L' in P and a $k \geq 0$ such that for all $w \in \Sigma^*$:

$$w \in L \text{ iff } \quad \text{there exists } c : \langle w, c \rangle \in L' \text{ and } |c| < |w|^k$$

c is also called witness or certificate for w in L.
A DTM which accepts the language L' is called verifier.

Important

A decision procedure is in NP iff every "Yes" instance has a short witness

(i.e. its length is polynomial in the length of the input)

which can be verified in polynomial time.

Complete and hard problems

Definition (NP-complete, NP-hard)
A language L is NP-hard (NP-difficult) if every language L' in NP is reducible in polynomial time to L.

A language L is NP-complete if:

– $L \in NP$

– L is NP-hard

Complete and hard problems

Definition (PSPACE-complete, PSPACE-hard)
A language L is PSPACE-hard (PSPACE-difficult) if every language L' in PSPACE is reducible in polynomial time to L.

A language L is PSPACE-complete if:

– $L \in PSPACE$

– L is PSPACE-hard

Complete and hard problems

Remarks:

- If we can prove that at least one NP-hard problem is in P then P $=$ NP
- If P $\neq$ NP then no NP complete problem can be solved in polynomial time

Open problem: Is P $=$ NP? (Millenium Problem)

Complete and hard problems

How to show that a language L is NP-complete?

Complete and hard problems

How to show that a language L is NP-complete?

1. Prove that $L \in NP$

2. Find a language L' known to be NP-complete and reduce it to L

Complete and hard problems

How to show that a language L is NP-complete?

1. Prove that $L \in NP$

2. Find a language L' known to be NP-complete and reduce it to L

Is this sufficient?

Complete and hard problems

How to show that a language L is NP-complete?

1. Prove that $L \in NP$

2. Find a language L' known to be NP-complete and reduce it to L

Is this sufficient?

Yes.

If L' is NP-complete then every language in NP is reducible to L', therefore also to L.

Complete and hard problems

How to show that a language L is NP-complete?

1. Prove that $L \in NP$
2. Find a language L' known to be NP-complete and reduce it to L

Is this sufficient?

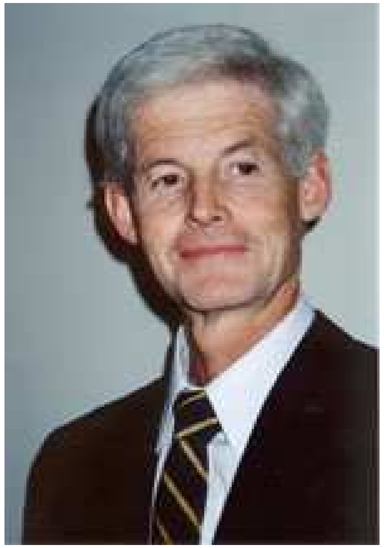
Yes.

If $L' \in NP$ then every language in NP is reducible to L' and therefore also to L.

Often used: the SAT problem (Proved to be NP-complete by S. Cook)

$$L' = L_{\text{sat}} = \{w \mid w \text{ is a satisfiable formula of propositional logic}\}$$

Stephen Cook



Stephen Arthur Cook (born 1939)

- Major contributions to complexity theory.
  Considered one of the forefathers of computational complexity theory.
- 1971 'The Complexity of Theorem Proving Procedures'
  Formalized the notions of polynomial-time reduction and NP-completeness, and proved the existence of an NP-complete problem by showing that the Boolean satisfiability problem (SAT) is NP-complete.
- Currently University Professor at the University of Toronto
- 1982: Turing award for his contributions to complexity theory.

Cook's theorem

Theorem $SAT = \{w \mid w \text{ is a satisfiable formula of propositional logic}\}$ is NP-complete.

Cook's theorem

Theorem $SAT = \{w \mid w \text{ is a satisfiable formula of propositional logic}\}$ is NP-complete.

Proof (Idea)

To show: (1) $SAT \in NP$

(2) for all $L \in NP$, $L \preceq_{\text{pol}} SAT$

Cook's theorem

Theorem $SAT = \{w \mid w \text{ is a satisfiable formula of propositional logic}\}$ is NP-complete.

Proof (Idea)

To show: (1) $SAT \in NP$

(2) for all $L \in NP$, $L \preceq_{\text{pol}} SAT$

(1) Construct a k-tape NTM M which can accept SAT in polynomial time:

$w \in \Sigma^*_{PL} \quad \mapsto \quad$ M does not halt if $w \not\in SAT$

M finds in polynomial time a satisfying assignment

(a) scan w and see if it a well-formed formula; collect atoms $\mapsto O(|w|^2)$
(b) if not well-formed: inf.loop; if well-formed M guesses a satisfying assignment $\mapsto O(|w|)$
(c) check whether w true under the assignment $\mapsto O(p(|w|))$
(d) if false: inf.loop; otherwise halt.

Cook's theorem

Theorem $SAT = \{w \mid w \text{ is a satisfiable formula of propositional logic}\}$ is NP-complete.

Proof (Idea) (2) We show that for all $L \in NP$, $L \preceq_{\text{pol}} SAT$

We show that we can simulate the way a NTM works using propositional logic.

Need formula $G(x_1, \dots, x_n)$ expressing:

from the atoms $x_1, \dots, x_n$ exactly one is true

Such a G exists (length k^2).

Let $L \in NP$. There exists a p-time bounded NTM which accepts L.

(Assume w.l.o.g. that M has only one tape and does not hang.)

For M and w we define a propositional logic language and a formula $T_{M,w}$ such that

$$M \text{ accepts } w \quad \text{iff} \quad T_{M,w} \text{ is satisfiable.}$$

We show that the map f with $f(w) = T_{M,w}$ has polynomial complexity.