Algorithm Theory

Georg Moser Mircea Dan Hernest

Institute of Computer Science @ UIBK

Summer 2007

Speed-up Theorems

Theorem Linear Speed-Up Theorem (Time)

- Assume $\exists$ TM M deciding L within time-bound $f(n)$, and $f(n) = \omega(n)$
- Then $\forall\ d > 0$, $\exists\ n_0 \in \mathbb{N}$, $\exists$ TM M' deciding L in time-bound $d \cdot f(n)$ for all $n \geqslant n_0$

Theorem Linear Speed-Up Theorem (Space)

- Assume $\exists$ TM M deciding L within space-bound $f(n)$, and $f(n) = \omega(1)$
- Then $\forall\ d > 0$, $\exists\ n_0 \in \mathbb{N}$, $\exists$ TM M' deciding L in space-bound $d \cdot f(n)$ for all $n \geqslant n_0$

Quantitative Church-Turing Thesis

Thesis ①

The class of effectively computable functions $f : \mathbb{N} \to \mathbb{N}$ coincides with the class of functions computable by a TM.
Church-Turing *1936*

Thesis ② quantum computers may change that

All reasonable sequential models of computation have the same time complexity as (deterministic) TM upto a polynomial factor.
common knowledge ☺ *1976*

Thesis ③ quantum computers may change that

Tractable problems are those that are in the class **P**.
Cook-Karp *197?*

Random Access Machines

- A random access machine (RAM) consists of an array of registers each holding an arbitrarily large integer; register 0 is the accumulator
- A RAM program $\Pi = (\pi_1, \pi_2, \dots, \pi_m)$ is a sequence of instructions, κ denotes the program counter

Instruction	Op	Semantics	
READ	j $(\uparrow j)$	$r_0 := i_j$	$(r_0 := i_{r_j})$
STORE	j $(\uparrow j)$	$r_j := r_0$	$(r_{r_j} := r_0)$
LOAD	x	$r_0 := x$	$x \in \{j, \uparrow j, = j\}$
ADD	x	$r_0 := r_0 + x$	$x \in \{j, \uparrow j, = j\}$
SUB	x	$r_0 := r_0 - x$	$x \in \{j, \uparrow j, = j\}$
HALF		$r_0 := \lfloor \frac{r_0}{2} \rfloor$	

JUMP	j	$\kappa := j$
JPOS, JZERO, JNEG	j	**if** $r_0 > 0, r_0 = 0, r_0 < 0$ **then** $\kappa := j$
HALT		$\kappa := 0$

Definition

- A configuration of Π is (κ, R), where

$$R = \{(j_1, r_{j_1}), \dots, (j_k, r_{j_k})\}$$

denotes a set of register-value pairs, changed so far

- $\forall$ sets of finite sequences of integers D
$\forall$ functions ϕ: D $\to$ int
Π computes ϕ if for any $I \in$ D

$$(1, \emptyset) \xrightarrow{(\Pi, I)^*} (0, R)$$

where $(0, \phi(I)) \in R$

- We write $\ell(i) = |(i)_2|$ for the binary length of i
- We set $\ell(I) = \sum_{j=1}^{n} \ell(i_j) \qquad I \in$ D
- Suppose

$$(1, \emptyset) \xrightarrow{(\Pi, I)^t} (0, R) \quad \text{and} \quad t \leqslant f(\ell(I))$$

Then Π computes ϕ in time $f(n)$

Definition

ϕ_{L}

- $\Sigma = \{\sigma_1, \dots, \sigma_k\}$ alphabet of a TM
- We set $\mathrm{D}_\Sigma = \{(i_1, \dots, i_n, 0) \mid n \geqslant 0, i_j \in [1, k]\}$
- $\forall \, \mathrm{L} \subseteq (\Sigma - \{\sqcup, \triangleright\})^*$
Define ϕ_{L}: $\mathrm{D}_\Sigma \to \{0, 1\}$:

$$\phi_{\mathrm{L}}((i_1, \dots, i_n, 0)) = 1 \quad \text{iff} \quad \sigma_{i_1} \dots \sigma_{i_n} \in \mathrm{L}$$

i.e., computing ϕ_{L} by a RAM is equivalent of deciding L

Example: Multiple two binary numbers

1.	READ 1		12.	ADD 5	
2.	STORE 1	$(R_1 = i_1)$	13.	STORE 4	$(R_4 = i_1 \cdot (i_2 \bmod 2^k))$
3.	STORE 5	$(R_5 = i_1 2^k)$	14.	LOAD 5	
4.	READ 2		15.	ADD 5	
5.	STORE 2 $(\star)$	(loop starts)	16.	STORE 5	
6.	HALF		17.	LOAD 3	
7.	STORE 3	$(R_3 = \lfloor \frac{i_2}{2^k} \rfloor)$	18.	JZERO 20	(**if** $R_3 = 0$ **done**)
8.	ADD 3		19.	JUMP 5	(**else, repeat**)
9.	SUB 2		20.	LOAD 4	
10.	JZERO 14		21.	HALT	
11.	LOAD 4				

$$(\star) \quad R_2 = i_2 \quad \text{if } k = 0\,, \qquad R_2 = \lfloor \frac{i_2}{2^{k-1}} \rfloor \quad \text{if } k > 0$$

TM and RAM

Theorem

Suppose $\mathrm{L} \in$ **TIME**$(f(n))$, then there is a RAM program which computes ϕ_{L} in time $\mathcal{O}(f(n))$

Definition

- Let I be a sequence $\{i_1, \dots, i_n\}$ of integers
we write $b(I)$ to denote the string $(i_1)_2; \dots; (i_n)_2$
- We say a TM M computes ϕ: D $\to$ int
if for any sequence $I \in$ D: $M(b(I)) = b(\phi(I))$

Theorem

If a RAM program Π computes a function ϕ in time $f(n)$, then there is a 7-string TM M which computes ϕ in time $\mathcal{O}(f(n)^3)$

Nondeterministic Time

➡ A nondeterministic Turing machine N is a quadruple (K, Σ, Δ, s) with

$$\Delta \colon K \times \Sigma \to \mathcal{P}((K \cup \{h, yes, no\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\})$$

➡ N decides L if for any $x \in \Sigma^*$:

$$x \in \mathrm{L} \quad \text{iff} \quad (s, \triangleright, x) \xrightarrow{N^*} (yes, w, u) \quad \text{for some } w \text{ and } u \,.$$

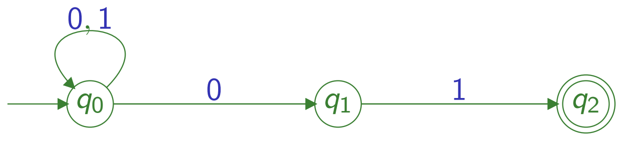
➡ N decides L in time $f(n)$ if

1. N decides L and
2. $\forall\, x \in \Sigma^*$:

$$(s, \triangleright, x) \xrightarrow{N^t} (q, w, u) \quad \text{implies} \quad t \leqslant f(|x|)$$

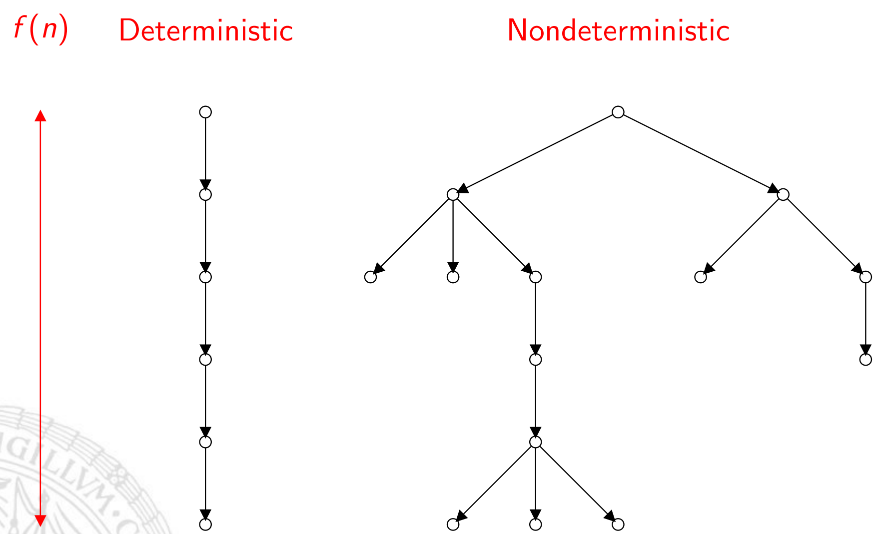
We write $\mathrm{L} \in \mathbf{NTIME}(f(n))$

Define a nondeterministic Turing machine (NTM) M that decides the language L of binary strings ending in the string 01:



$p \in K$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	$\triangleright$	$\{(s, \triangleright, \rightarrow)\}$
s	$\sqcup$	$\emptyset$
s	0	$\{(s, 0, \rightarrow), (q_1, 0, \rightarrow)\}$
s	1	$\{(s, 1, \rightarrow)\}$
q_1	$\sqcup$	$\emptyset$
q_1	0	$\emptyset$
q_1	1	$\{(q_2, 1, \rightarrow)\}$
q_2	$\sqcup$	$\{yes\}$
q_2	_	$\emptyset$

Measuring Time



Nondeterministic Space

Definition
N decides L within space $f(n)$ if

1. N decides L and
2. $\forall\, x \in (\Sigma - \{\triangleright, \sqcup\})^*$:

$$(s, \triangleright, x, \dots, \triangleright, \epsilon) \xrightarrow{N^*} (q, w_1, u_1, \dots, w_k, u_k)$$
$$\text{implies } \textstyle\sum_{j=2}^{k-1} |w_j u_j| \leqslant f(|x|)$$

We write $\mathrm{L} \in \mathbf{NSPACE}(f(n))$

Example REACHABILITY $\in \mathbf{NSPACE}(\mathcal{O}(\log n))$

1. use 2 strings beside the input
2. on the 2nd string write the currently checked node i
3. on the 3rd string we write a guess j
4. check whether (i, j) is in the graph; repeat

Complexity Classes (continued)

Definition

TIME$(f(n))$ **SPACE**$(f(n))$ **NTIME**$(f(n))$ **NSPACE**$(f(n))$

We may replace f by a family of functions, parameterised by k

$$\mathbf{TIME}(n^k) = \bigcup_{i>0} \mathbf{TIME}(n^i) = \mathbf{P}$$

$$\mathbf{NTIME}(n^k) = \bigcup_{i>0} \mathbf{NTIME}(n^i) = \mathbf{NP}$$

Other classes:

$$\mathbf{PSPACE} = \mathbf{SPACE}(n^k) \qquad \mathbf{NPSPACE} = \mathbf{NSPACE}(n^k)$$

$$\mathbf{EXP} = \mathbf{TIME}(2^{n^k})$$

$$\mathbf{L} = \mathbf{SPACE}(\log n) \qquad \mathbf{NL} = \mathbf{NSPACE}(\log n)$$

Determinism vs Nondeterminism

Example TSP(D)

TSP(D) $\in$ **NP** as TSP(D) $\in$ **NTIME**(n^2):

1. Use 2 strings
2. Guess a tour on the first string
3. Check the tour on the second string.

Theorem

Suppose $\mathrm{L} \in \mathbf{NTIME}(f(n))$
Then L is also decided by a 3-string deterministic TM M in time $\mathcal{O}(d^{f(n)})$
$d > 1$ depends on the NTM N deciding L, i.e.,

$$\mathbf{NTIME}(f(n)) \subseteq \bigcup_{d>1} \mathbf{TIME}(d^{f(n)})$$

Alternative Definition of **NP**

Definition

- A verifier of a language L is an algorithm P such that:

  $$\mathrm{L} = \{w \mid \text{there exists a string } c \text{ so that P accepts } \langle w, c\rangle\}$$

- A polynomial verifier is one that runs in time polynomial in $|w|$

Theorem

NP is the class of all languages that have polynomial verifiers

Proof

A polynomial verifier P runs in polynomial time:
Hence it can have only polynomial bounded certificates c

Then

- Transform a NTM into a verifier, by reading the certificate as a choice sequence
- Transform a verifier into a NTM by guessing the certificate □