2012.12.10

Exploring the
Limits of
Computation

ELC Complexity Theory
Intro. Seminar Series

Basics: Complexity Classes

Osamu Watanabe
Dept. of Math and Comp. Sci.
Tokyo Institute of Technology

ELC is supported by Grant-in-Aid for Scientific Research on Innovative Areas, MEXT, Japan

0. Contents

1. Problem, Algorithm
2. Complexity Measure
3. Basic Complexity Classes
4. Class NP
5. Polynomial-time Reducibility

計算複雑さ博物学

0. Complexity Theory --- Zoology

Too many !!

Complexity Zoo

A A0PP - AC - AC0 - AC0[m] - AC1 - ACC0 - AH - AL - ALL - ALOGTIME - AlgP/poly - Almost-NP - Almost-P - Almost-PSPACE - AM - AMEXP - AM ∩ coAM - AM[polylog] - AmpMP - AmpP-BQP - AP - APP - APX - ATIME - AUC-SPACE(f(n)) - AuxPDA - AVBPP - AvgE - AvgP - AW[P] - AWPP - AW[SAT] - AW[*] - AW[t] - AxP - AxPP

B βP - BH - BPd(P) - BPE - BPEE - BPHSPACE(f(n)) - BPL - BP?NP - BPP - BPPcc - BPPkcc - BPPKT - BPP/log - BPP/mlog - BPP/
BPP/rlog - BPP-OBDD - BPPpath - BPQP - BPSPACE(f(n)) - BPTIME(f(n)) - BQNC - BQNP - BQP - BQP/log - BQP/poly - BQP/mlo
BQP/mpoly - BQP/qlog - BQP/qpoly - BQP-OBDD - BQPSPACE - BQPCTC - BQPtt/poly - BQTIME(f(n)) - k-BWBP

C C=AC0 -
compNP -
CSIZE(f(n

D D#P -
Dyn-ThC0

E E - EE
∃NISZK -

F FBQP -
FPNP[log]

G GA - G

H HalfP -

I IC[log,p

L L - LC0

M MA - M
MIPEXP -

N NAuxP
NISZKh -
NPMV - N
sel - NQP (f(n)) - NT - NT* - NTIME(f(n))

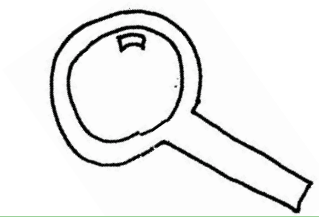
O OCQ -

Master's guide for your study:

1. Do not get trapped by notations and mathematical def.s
2. Consider always story!
3. There must be some motivations behind def.s and thm.s

Don't be panic !!

0. Complexity Theory --- Zoology



Let us find good targets !!!

Complexity Zoo

A A0PP - AC - AC0 - AC0[m] - AC1 - ACC0 - AH - AL - ALL - ALOGTIME - AlgP/poly - Almost-NP - Almost-P - Almost-PSPACE - AM - AMEXP - AM ∩ coAM - AM[polylog] - AmpMP - AmpP-BQP - AP - APP - APX - ATIME - AUC-SPACE(f(n)) - AuxPDA - AVBPP - AvgE - AvgP - AW[P] - AWPP - AW[SAT] - AW[*] - AW[t] - AxP - AxPP

B βP - BH - BPd(P) - BPE - BPEE - BPHSPACE(f(n)) - BPL - BP?NP - BPP - BPPcc - BPPkcc - BPPKT - BPP/log - BPP/mlog - BPP//log - BPP/rlog - BPP-OBDD - BPPpath - BPQP - BPSPACE(f(n)) - BPTIME(f(n)) - BQNC - BQNP - BQP - BQP/log - BQP/poly - BQP/mlog - BQP/mpoly - BQP/qlog - BQP/qpoly - BQP-OBDD - BQPSPACE - BQPCTC - BQPtt/poly - BQTIME(f(n)) - k-BWBP

C C=AC0 - C=L - C=P - CC - CC0 - CFL - CLOG - CH - Check - CL#P - CkP - CNP - coAM - coC=P - cofrIP - Coh - coMA - coModkP - compIP - compNP - coNE - coNEXP - coNL - coNP - coNPcc - coNP/poly - coNQP - coRE - coRNC - coRP - coSL - coSPARSE - coUCC - coUP - CP - CSIZE(f(n)) - CSL - CSP - CZK

D D#P - DCFL - Δ2P - δ-BPP - δ-RP - DET - DiffAC0 - DisNP - DistNP - DP - DQP - DSPACE(f(n)) - DTIME(f(n)) - DTISP(t(n),s(n)) - Dyn-FO - Dyn-ThC0

E E - EE - EEE - EESPACE - EEXP - EH - ELEMENTARY - ELkP - EP - EPTAS - k-EQBP - EQP - EQPK - EQTIME(f(n)) - ESPACE - ∃BPP - ∃NISZK - EXP - EXP/poly - EXPSPACE

F FBQP - Few - FewEXP - FewP - FH - FIXP - FNL - FNL/poly - FNP - FO - FO(DTC) - FO(LFP) - FO(PFP) - FO(TC) - FO(t(n)) - FOLL - FF FPNP[log] - FPR - FPRAS - FPT - FPTnu - FPTsu - FPTAS - FQMA - frIP - F-TAPE(f(n)) - F-TIME(f(n))

G GA - GAN-SPACE(f(n)) - GapAC0 - GapL - GapP - GC(s(n),C) - GCSL - GI - GLO - GPCD(r(n),q(n)) - G[t]

H HalfP - HeurBPP - HeurBPTIME(f(n)) - HeurDTIMEδ(f(n)) - HeurP - HeurPP - HeurNTIMEδ(f(n)) - HkP - HVSZK

I IC[log,poly] - IP - IPP - IP[polylog]

L L - LC0 - LH - LIN - LkP - LOGCFL - LogFew - LogFewNL - LOGLOG - LOGNP - LOGSNP - L/poly - LWPP

M MA - MA' - MAC0 - MAE - MAEXP - mAL - MAPOLYLOG - MaxNP - MaxPB - MaxSNP - MaxSNP0 - mcoNL - MinPB - MIP - MIP*[2,1] - MIPEXP - (Mk)P - mL - MM - MMSNP - mNC1 - mNL - mNP - ModkL - ModL - ModkP - ModP - ModZkL - mP - MP - MPC - mP/poly - mTC^

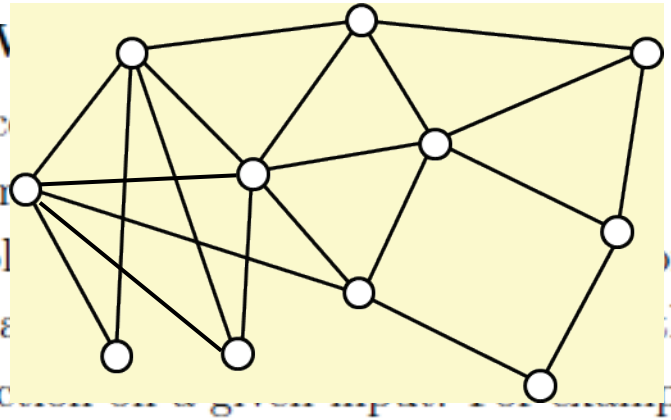
N NAuxPDAp - NC - NC0 - NC1 - NC2 - NE - NE/poly - Nearly-P - NEE - NEEE - NEEXP - NEXP - NEXP/poly - NIPZK - NIQSZK - NISZ NISZKh - NL - NL/poly - NLIN - NLOG - NONE - NNC(f(n)) - NP - NPC - NPC - NPcc - NPkcc - NPI - NP ∩ coNP - (NP ∩ coNP)/poly - NP NPMV - NPMV-sel - NPMVt - NPMVt-sel - NPO - NPOPB - NP/poly - (NP,P-samplable) - NPR - NPSPACE - NPSV - NPSV-sel - NPSVt - NPSVt-sel - NQP - NSPACE(f(n)) - NT - NT* - NTIME(f(n))

O OCQ - OptP

L. Fortnow
by wizard

by collaboration

1. Problem and Algorithm



	0	1	2	3	4	5	6	7	8	9
0	0	1	0	1	1	1	0	0	0	1
1	1	0	0	1	0	0	1	0	1	0
2	0	1	0	1	0	0	1	0	1	0
3	.....									
4	.....									
5										
6										
7										
8										
9										

adjacency matrix

01011000110010010100101001010 ....

- W

In c we consi at are completely and form mathem blems, a problem like "Sol f our inte sider a problem asking for a he probl computing some given func ple, a pr he following way:

問題→ Max. Clique Problem (abbrev. CLIQUE)

入力例→ **Instance**: An undirected graph G.

出力→ **Question**: What is the size of the largest clique (i.e., complete subgraph) of G?

Remarks:

1. Input instances and outputs are encoded appropriately as binary strings, i.e., finite strings over the alphabet $\{0, 1\}$.
2. In our *general* discussion, the *size* of an input is simply its length. Throughout this course, we use ℓ to denote the input size.

1. Problem and Algorithm

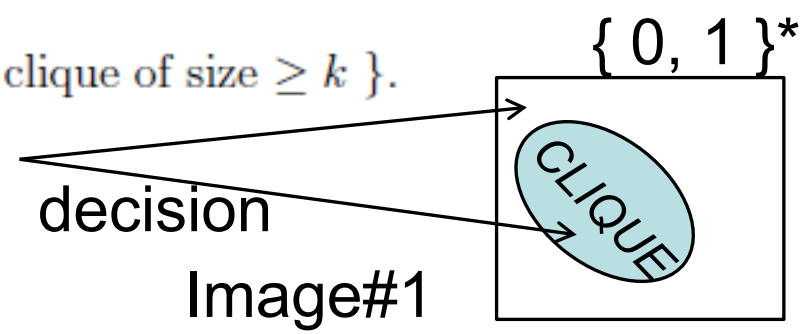
- **Decision problem**

Among computational problems like CLIQUE, we consider only "decision problems", where a *decision problem* is to decide whether a given instance is 'yes' (or simply 1) or 'no' (simply 0). For example, instead of the CLIQUE problem mentioned above we can consider the following decision version.

> Max. Clique Problem (CLIQUE)
> **Instance**: An undirected graph G and an integer k.
> **Question**: Does G have a clique of size $\geq k$?

3. Any decision problem is completely specified by the set of 'yes' instances of the problem, or more precisely, the set of binary strings encoding those 'yes' instances. Thus, we usually identify a problem with its 'yes' instances. For example,

$$\text{CLIQUE} = \{ \underbrace{(G, k)}_{\text{Input instance}} : G \text{ has a clique of size} \geq k \}.$$



Image#1

1. Problem and Algorithm

- **Function and algorithm**

A problem can be regarded as a function mapping $\{0,1\}^*$ to $\{0,1\}^*$, and an *algorithm* (for the problem) is a method for computing this function. We assume that each algorithm is implemented by a *program* for some machinery, and we will identify them throughout this course.

Notations:

function:	f, g_{clique}, ...	usage:	$f(x) \Leftarrow$ the value of f on x.
program:	$\mathtt{A}$, $\mathtt{clique}$, ...	usage:	$\mathtt{A}(x) \Leftarrow$ the output of $\mathtt{A}$ on x
			or the execution of $\mathtt{A}$ on x.

We say that an algorithm $\mathtt{A}$ (or, a program $\mathtt{A}$) *solves* a decision problem L if the following holds.

$$\forall x \in \{0,1\}^* \quad \left[\begin{array}{l} x \in L \implies \mathtt{A}(x) = 1 \\ x \notin L \implies \mathtt{A}(x) = 0 \end{array} \right]$$

1. Problem and Algorithm

For our basic computation model, we consider some standard computation device and assume some simple programming language controlling the device. For example, we consider the following program.

```
program A
input register:   $1
output register:  $1
work registers:   $2, $3, $4
1:  $2 ← ···      ⇐ Only very primitive instructions are allowed.
2:  if $2 ≠ 0 then goto 6
3:     ⋮
4:     ⋮
5:  goto 2
6:  halt
```

var. $1

010110

bin string in { 0, 1 }*

integer 22

Note

1. program (algorithm) itself is encoded in {0, 1}*.
2. # of produced bits ≤ # of comp. steps

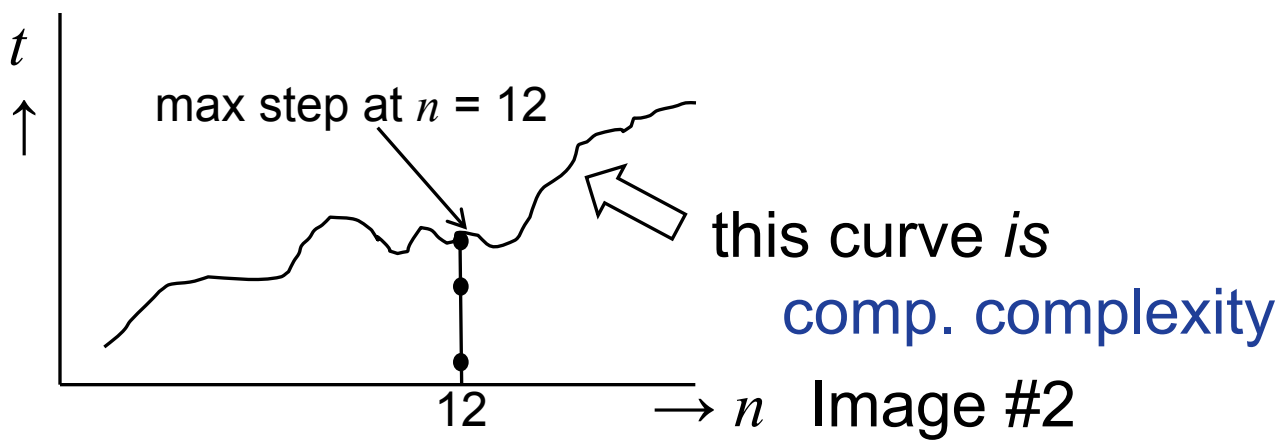
2. Comp. Complexity Measure

Consider any program A. For a given input x, we estimate its *computation time* and *consumed space* as follows.

$$\begin{aligned} \mathrm{time}_{\mathtt{A}}(x) &= \text{the number of executed instructions until } \mathtt{A}(x) \text{ halts.} \\ \mathrm{space}_{\mathtt{A}}(x) &= \text{the total number of bits in all registers.} \end{aligned}$$

Since we are interested in how the complexity grows when input size gets increased, our complexity analysis is based on the following functions, which are respectively called a *time complexity* and a *space complexity*.

$$\begin{aligned} \mathrm{time}_{\mathtt{A}}(\ell) &= \max\{\ \mathrm{time}_{\mathtt{A}}(x) \,|\, x \in \{0,1\}^{\ell}\ \}. \\ \mathrm{space}_{\mathtt{A}}(\ell) &= \max\{\ \mathrm{space}_{\mathtt{A}}(x) \,|\, x \in \{0,1\}^{\ell}\ \}. \end{aligned}$$



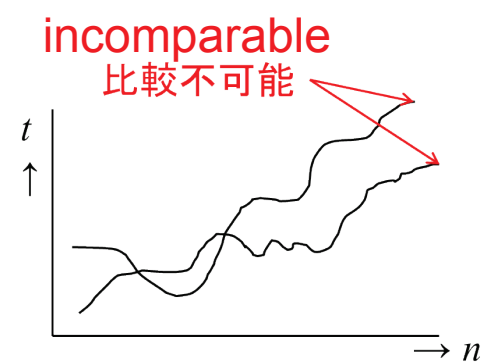


Image #2

2. Comp. Complexity Measure

Definition 1. For any functions f and g from $\mathbf{Z}^+$ to $\mathbf{Z}^+$, we write $f = O(g)$ if the following holds for some constants c_0 and n_0.

$$\forall n \geq n_0 \; [\; f(n) \;\leq\; c_0 \cdot g(n) \;].$$

Remarks:

1. The big-O notation does not imply the asymptotic equivalence between two functions; it just means that one function is asymptotically bounded *from above* by another function. For example, we can write $f = O(g)$ for $f(n) = n$ and $g(n) = 2^n$.
2. Important examples: (below p denotes any polynomial)

$$p(\log n) \;=\; O((\log n)^k) \;=\; O(n) \quad (\text{for some } k > 0)$$
$$p(n) \;=\; O(n^k) \;=\; O(2^n) \quad (\text{for some } k$$
$$f(n) = O(g(n)) \;\&\; g(n) = O(h(n)) \implies$$

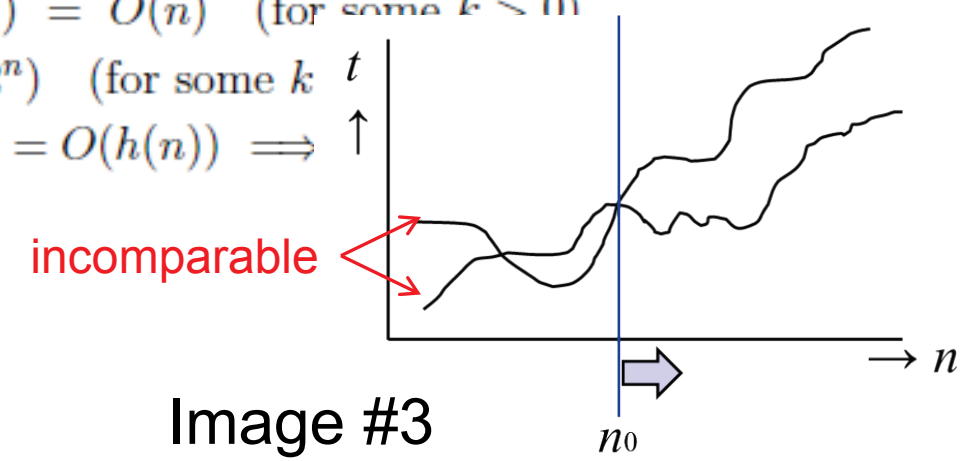
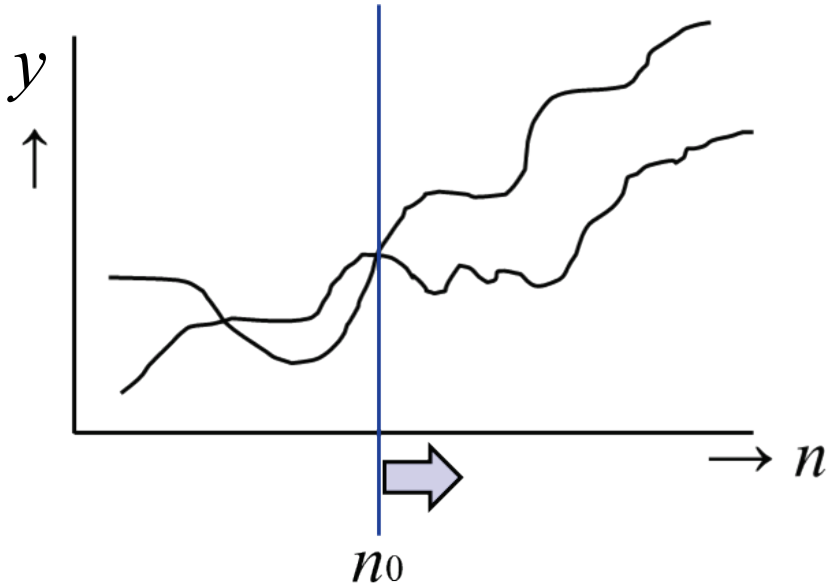
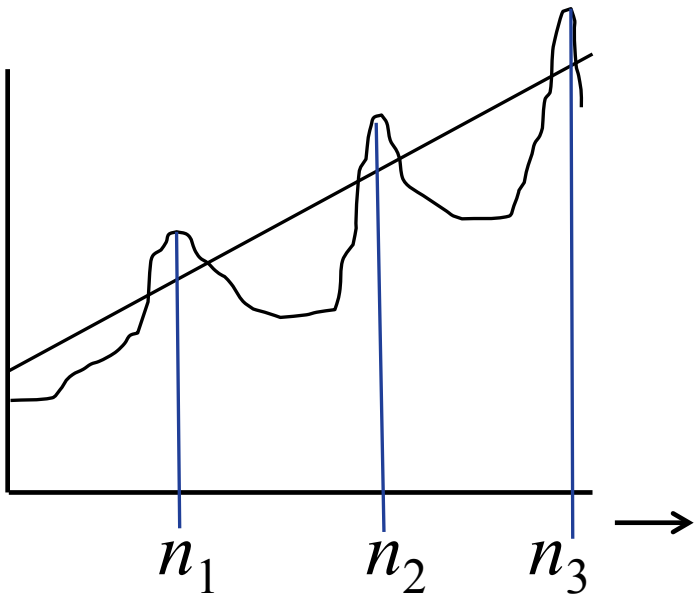


Image #3

2. Comp. Complexity Measure

This asymptotic analysis is often used.





$$\exists n_0 \forall n > n_0 = \overset{\infty}{\forall} n$$

$$\forall n' \exists n > n' = \overset{\infty}{\exists} n$$

Image #3

3. Basic Complexity Classes

Time classes:

$$\mathrm{P} = \bigcup_{p:\mathrm{poly}} \mathrm{DTime}(p), \quad \mathrm{E} = \bigcup_{c>0} \mathrm{DTime}(2^{c\ell}), \quad \mathrm{DEXP} = \bigcup_{p:\mathrm{poly}} \mathrm{DTime}(2^{p(\ell)}).$$

EXP

Space classes:

$$\mathrm{DLOG} = \mathrm{DSpace}(\log \ell), \quad \mathrm{PSPACE} = \bigcup_{p:\mathrm{poly}} \mathrm{DSpace}(p).$$

Polynomial ?

$$12n^8 - 819n^5 + 52n^3 + 128$$

$$= O(n^8)$$

$$= n^{O(1)}$$ = polynomial on n

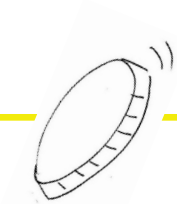
functions larger than polynomials:

$$n^{\log(\log n)} \quad n^{\log n} \quad 2^n \quad 2^{n^3}$$

3. Basic Comp. Classes: 乱択計算

Randomized Comp. Classes

Idea



Randomized computation is allowed

to use random coin tosses for some decision during the computation + to make some errors

input string

x

random 0,1 -seq. w

0110 ⋯

A

```
program A
input register:  $1
output register: $1
work registers:  $2, $3, $4
1: $2 ← ···      ⇐ Only very primitive instructions are allowed.
2: if $2 ≠ 0 then goto 6
3:   ⋮
4:   ⋮
```

deterministic algorithm

1 (= yes)

0 (= no)

$\Pr_w [\, A(x\,;w) = [\, x \in L \,?\,] \,]$ = correct prob.

$\Pr_w [\, A(x\,;w) \neq [\, x \in L \,?\,] \,]$ = error prob.

3. Basic Comp. Classes:

Randomized Comp. Classes

Def.

For a given decision problem, an algo. A is called **bounded error** if

$$\Pr_w [\, A(x\,;w) \neq [\, x \in L \,?\,]\,] < 1/3$$

in other words

$$x \in L \;\Rightarrow\; \Pr_w[\, A(x\,;w) = 1\,] > 2/3$$

$$x \notin L \;\Rightarrow\; \Pr_w[\, A(x\,;w) = 0\,] > 2/3$$

promise algorithm

1/3 ~ 2/3 does not occur

input string

x

random 0,1 -seq. w

0110 ⋯

A

```
program A
input register:  $1
output register: $1
work registers:  $2, $3, $4
1: $2 ← ⋯      ⇐ Only very primitive instructions are allowed.
2: if $2 ≠ 0 then goto 6
3:   ⋮
4:   ⋮
```

deterministic algorithm

→ 1 (= yes)

→ 0 (= no)

$\Pr_w [\, A(x\,;w) = [\, x \in L \,?\,]\,]$ = correct prob.

$\Pr_w [\, A(x\,;w) \neq [\, x \in L \,?\,]\,]$ = error prob.

3. Basic Complexity Classes

Def.

P = class of problems solvable in poly. time

BPP = class of problems solvable in poly. time by some bounded error algorithm

EXP = class of problems solvable in 2^{poly} time

Thm.

$$P \subseteq BPP \subseteq EXP$$

Conjecture

$$P = BPP$$

4. Class NP

Def.

NP

= class of problems for deciding x in L for a set that can be characterized as follows

$$\forall x \in \{0,1\}^* \quad \left[\begin{array}{l} x \in L \implies \exists w : |w| \le q_L(|x|) \ [\ R_L(x,w) = 1\ ] \\ x \notin L \implies \forall w : |w| \le q_L(|x|) \ [\ R_L(x,w) = 0\ ] \end{array} \right]$$

with some polynomial $q_L(n)$ and some poly. time computable predicate R_L.

証拠 witness

witness checking predicate

4. Class NP

The NP-condition is equivalent to the following one.

$$\forall x \in \{0,1\}^* \quad \left[\begin{array}{l} x \in L \implies \exists w : |w| \leq q_L(|x|) \; [\; R_L(x,w) = 1 \;] \\ x \notin L \implies \forall w : |w| \leq q_L(|x|) \; [\; R_L(x,w) = 0 \;] \end{array} \right]$$

$$\begin{array}{rcl}
\text{HAM} & = & \{\; G \;:\; G \text{ is a Hamiltonian graph} \;\}. \\
\text{COLOR} & = & \{\; (G,k) \;:\; G \text{ is } k\text{-colorable} \;\}. \\
\text{VC} & = & \{\; (G,k) \;:\; G \text{ has a vertex cover of size } k \;\}. \\
\text{SAT} & = & \{\; F \;:\; F \text{ is a satisfiable (extended) Boolean formula} \;\}. \\
\text{3SAT} & = & \{\; F \;:\; F \text{ is a satisfiable 3CNF formula} \;\}. \\
\text{KNAP} & = & \{\; ((s_1,\ldots,s_n),(c_1,\ldots,c_n),B,K) \;: \\
& & \quad \exists U \subseteq \{1,\ldots,n\} \; [\; \sum_{u \in U} s_u \leq B \;\wedge\; \sum_{u \in U} c_u \geq K \;] \;\}. \\
\text{SSUM} & = & \{\; ((a_1,\ldots,a_n),B) \;:\; \exists U \subseteq \{1,\ldots,n\} \; [\; \sum_{u \in U} a_u = B \;] \;\}.
\end{array}$$

4. Class NP

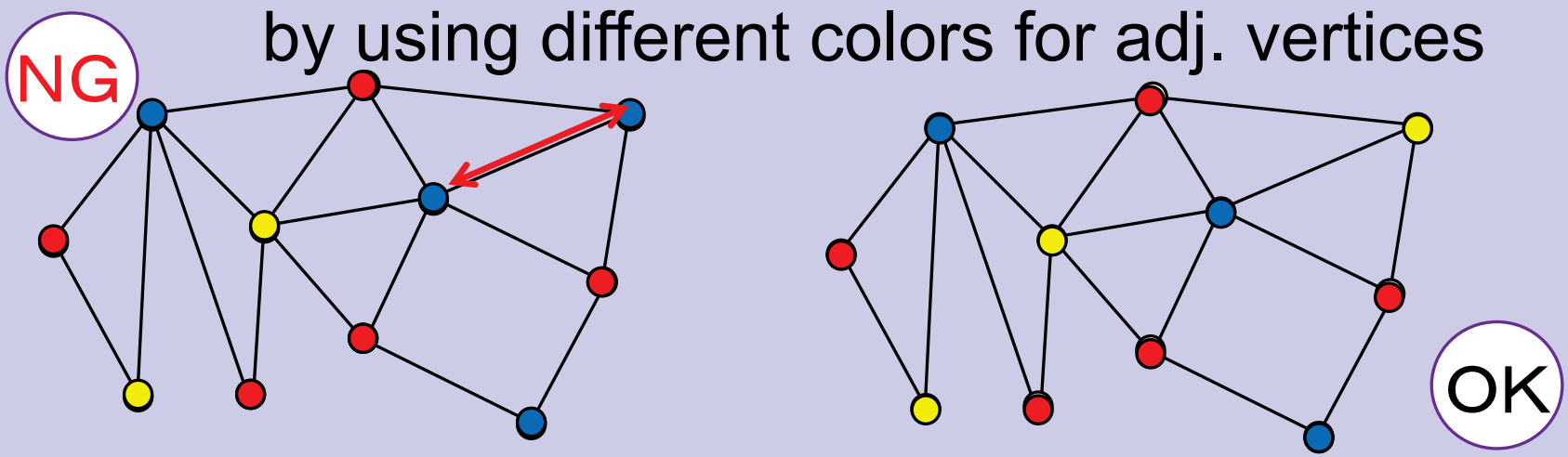
The NP-condition is equivalent to the following one.

$$\forall x \in \{0,1\}^* \quad \left[\begin{array}{l} x \in L \implies \exists w : |w| \le q_L(|x|) \ [\ R_L(x,w) = 1\ ] \\ x \notin L \implies \forall w : |w| \le q_L(|x|) \ [\ R_L(x,w) = 0\ ] \end{array} \right]$$

Example: 3Coloring Problem

Color vertices with three colors

by using different colors for adj. vertices



4. Class NP

NP set

$$\forall x \in \{0,1\}^* \quad \left[\begin{array}{l} x \in L \implies \exists w : |w| \le q_L(|x|) \; [\, R_L(x,w) = 1 \,] \\ x \notin L \implies \forall w : |w| \le q_L(|x|) \; [\, R_L(x,w) = 0 \,] \end{array} \right]$$

Thm.

$$\boxed{P \subseteq NP \subseteq EXP}$$

Conjecture

$$\boxed{P \neq NP}$$

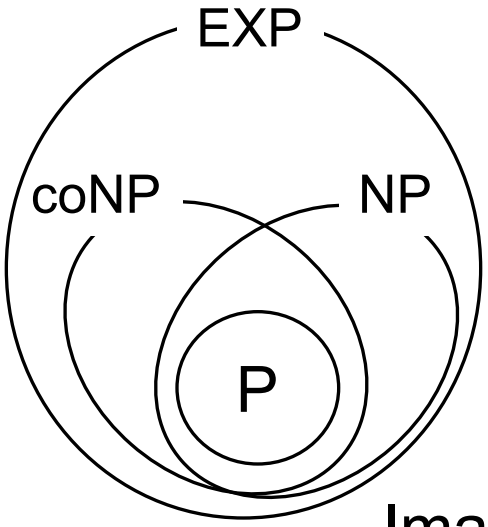


Image #4

SAT $\overset{?}{\in}$ P

3SAT $\overset{?}{\in}$ P

素因数分解 $\overset{?}{\in}$ P

VC $\overset{?}{\in}$ P

HAM $\overset{?}{\in}$ P

5. Poly. time reducibility

Idea

Solve A by using algo. for B

Definition 4. For any decision problems (or, sets of strings) A and B, we say that A is *many-one reducible* to B (abbrev. $A \leq_{\mathrm{m}} B$) if there exists a function h with the following properties. (h itself is called a *many-one reduction* (abbrev. $\leq_{\mathrm{m}}$-reduction).)

(a) h is a total function from $\{0,1\}^*$ to $\{0,1\}^*$,

(b) $\forall x \in \{0,1\}^*\ [\ x \in A \iff h(x) \in B\ ]$, and

(c) h is computable.

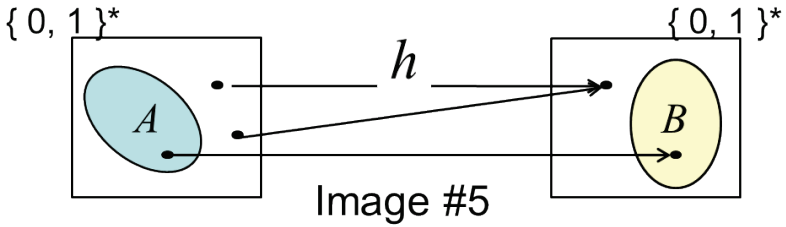


Image #5

If furthermore, h is polynomial-time computable, then we say that A is *polynomial-time many-one reducible* (abbrb. $A \leq_{\mathrm{m}}^{\mathrm{p}} B$). The *log-space many-one reducibility* (abbrb. $\leq_{\mathrm{m}}^{\log}$-

B∈P ⇒ A∈P

Then we have

A's difficulty ≦ B's difficulty

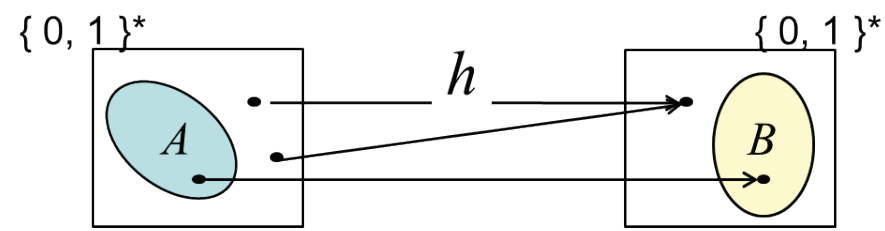
without answering A, B ∈ P ?

	P or not P ?	
B	○	×
A	○	○ or ×

5. Poly. time reducibility

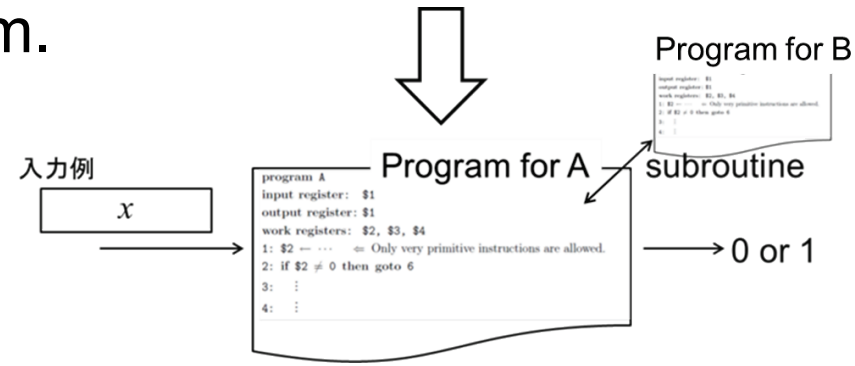
Thm

$$A \leqq_m^P B \ \& \ B \in P \Rightarrow A \in P$$



Proof. By the following algorithm.

```
program IsA(input x);
    y ← h(x);
    halt(IsB(y));
end-program.
```



Thm

- $A \leqq_m^P B \ \& \ B \in NP \Rightarrow A \in NP$
- $A \leqq_m^P B \ \& \ B \in BPP \Rightarrow A \in BPP$

. . . .

Thm

$$A \leqq_m^P B \ \& \ B \leqq_m^P C \Rightarrow A \leqq_m^P C$$

5. Poly. time reducibility: Completeness

Def.

We say that C is NP-complete if
(1) C in NP,
(2) every L in NP, we have $L \leqq_{\mathrm{m}}^{\mathrm{P}} C$.

Consider any complete set C.

Idea

For any L in NP
hardness of L ≤ harness of C

Thm

$C \in \mathrm{P} \Leftrightarrow \mathrm{P} = \mathrm{NP}$

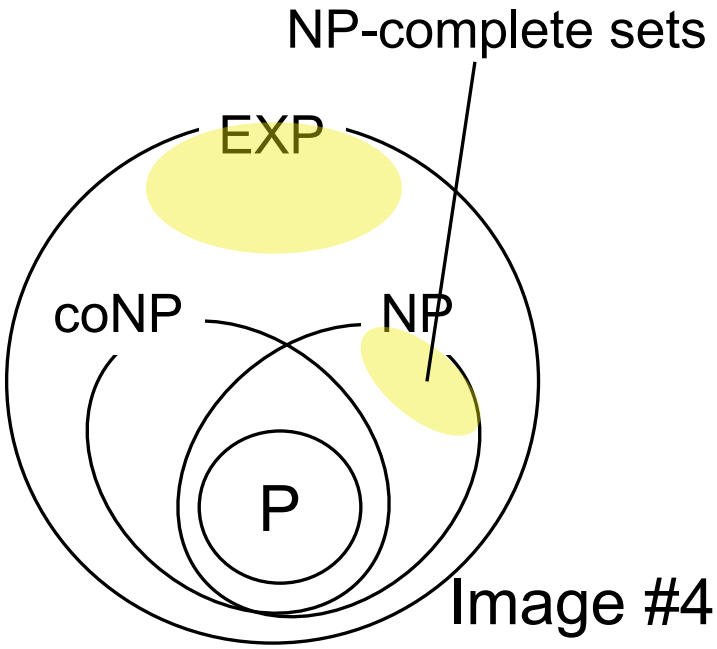


Image #4

5. Poly. time reducibility: Completeness

Example

encoded in {0, 1}*

seq. of t 0's, i.e., 00...0

$K = \{ \ (R, x, 0^t) : \ \exists w$ such that $R(x, w) = 1$ in t steps $\}$

Thm [Cook-Levin]

SAT is NP-complete

Thm **K is NP-complete**

our reduction

$h(x) = (R_L, x, 0^{p(n)})$

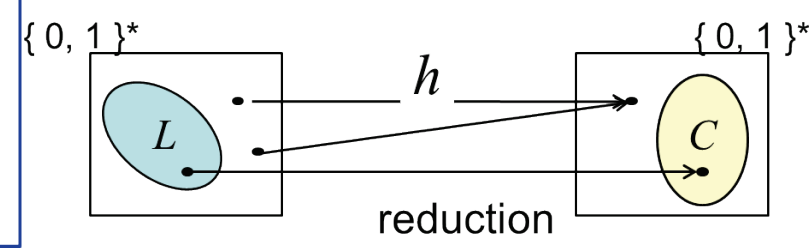
Proof. Consider any L in NP.
We may assume some R_L such that for any x.

$$x \text{ in } L \iff \exists w \text{ such that } R_L(x, w) \text{ in } p(n) \text{ steps}$$

$$\iff (R_L, x, 0^{p(n)}) \text{ in } K \iff h(x) \text{ in } K$$

Def.

We say that C is NP-complete if
(1) C in NP,
(2) every L in NP, we have $L \leq_m^P C$.



典型的 NP 問題 = NP 完全問題

効率解法が
見つかる場合も！

抽象化された問題

三彩色問題　充足割当て問題
整数計画問題　：

線形計画問題　素因数分解
マッチング問題　：

具体的な問題

ナッシュ均衡計算　最適戦略
最適ポートフォリオ設計
エネルギー最小自触媒作用
ニューラルネットワークの訓練
結晶格子解析（3次元結晶）
復元問題
最適進化系統樹
タンパク質の立体構造

プログラム・コード最適化
穴あけ工程の最適化　色空間の最適量子
VLSIのチッププレースメント
最適配置問題　最適配置
最小距離配線
ジスター・アロケーション
ロボットの最短移動　ロジスティック最適化
スケジューリング
乗務員スケジューリング
製造工程スケジューリング

キャッシュフローの管理
電力網設計　**公開鍵暗号の解読**
病院・医師　配置

<u>Cook-Levin の定理</u>

次のうちどちらかである：
○ これら全部が容易（P）
× これら全部が困難（非 P）

5. Poly. time reducibility: Completeness

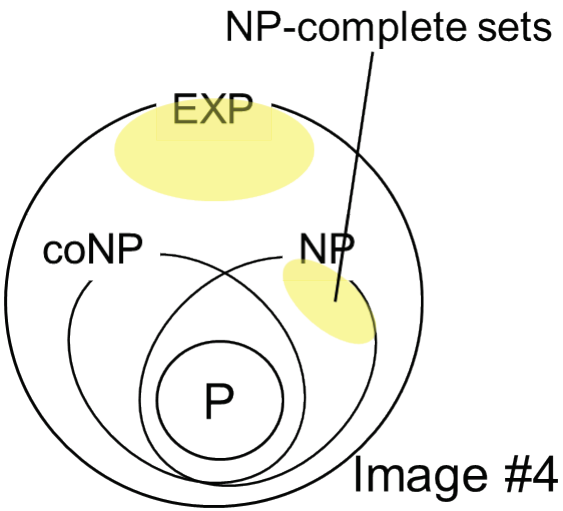
Thm

$$\boxed{C \in \mathrm{P} \Leftrightarrow \mathrm{P} = \mathrm{NP}}$$

NP-complete problems are

* typical NP problems, and
* most difficult problems in NP

but !! ....



Remark !! Everything is within poly. time factors

Example 4. There are some A and B such that $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ and yet $A \notin \mathrm{DTime}(\ell^{100})$ and $B \in \mathrm{DTime}(\ell^2)$.

In fact, every L in P is $\leq_{\mathrm{m}}^{\mathrm{p}}$-reducible to $\{0\}$ by the following reduction h_L.

$$h_L(x) = \begin{cases} 0, & \text{if } x \in L, \text{ and} \\ 00, & \text{otherwise.} \end{cases}$$

furthermore, within the worst-case scenario.

0. Contents

1. Problem, Algorithm
2. Complexity Measure
3. Basic Complexity Classes
4. Class NP
5. Polynomial-time Reducibility

Fin