2012.12.10

Exploring the
Limits of
Computation

ELC Complexity Theory
Intro. Seminar Series

Average-Case Complexity Theory for NP

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. Framework for average-case NP vs. P
2. Worst-case vs. average-case
3. Reducibility for the average-case
4. Search vs. decision
5. P-samplable dist. and one-way function

key reference

1. Jie Wang's Average-case complexity forum
2. Du and Ko, Theory of Computational Complexity John-Wisely Sons, Inc., 2000.

$\boxed{\star k}$ ·· See the end for more precise explanation.

1. Framework for average-case P

Idea

Complexity is analyzed on average for randomly given problem instances.

expected. comp. time

input distribution

Def

A **distributional problem** is a pair (L, D) of a problem L and a distribution D on input instances.

Def ??

Algorithm A's **expected time complexity**

$$= \Sigma_x \mathrm{time}_A(x) \times D(x)$$

Pr[x is given as an input]

1. Framework for average-case P

input distribution ?

Notation $D(x) = \Pr[\, x \text{ is given as an input} \,]$

uniform distribution D_U is defined by (let $n = |x|$)

$D_U(x) = c2^{-n} / n^2$ over $\{0, 1\}^*$

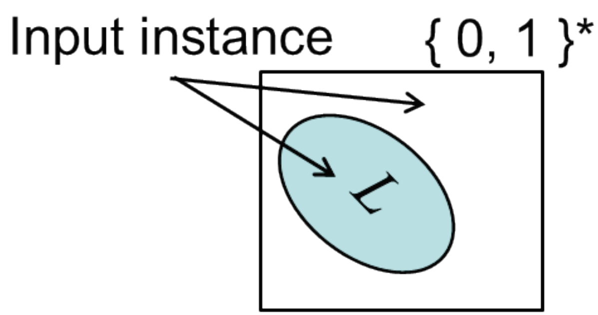
$D_U(x) = 2^{-n}$ over $\{0, 1\}^n$ for each n

Note !

How shall we define input distribution range?

$\{0, 1\}^*$ or $\{0, 1\}^n$?

$D(\{0, 1\}^*) = 1$ or $D(\{0, 1\}^n) = 1$



Image#1

unfriendly

less beautiful (mathematically)

1. Framework for average-case P

input distribution ?

Notation $D(x) = \Pr[\, x \text{ is given as an input} \,]$

uniform distribution D_U is defined by (let $n = |x|$)

$D_U(x) = c2^{-n} / n^2$ over $\{0, 1\}^*$

$D_U(x) = 2^{-n}$ over $\{0, 1\}^n$ for each n

Def poly. time computable distribution

A distribution D is **P-dist** (= poly. time comp. dist.)

$\Leftrightarrow$ $F(x) = \Sigma_{y < x} D(y)$ is poly. time computable

cumulative probability

1. Framework for average-case P

input distribution: over $\{ 0, 1 \}^*$

Ex $D_U(x) = c2^{-n} / n^2$

expected. comp. time ??

Def ??

A's **expected time?** $= \Sigma_x \text{time}_A(x) \times D(x)$

Algorithm A is polynomial time on average

$\Leftrightarrow \Sigma_{|x|=n} \text{time}_A(x) \times D(x) = n^{O(1)} = n^c$

Ex

$$\text{time}_A(x) = \begin{cases} 2^{0.5n} & \text{for } 2^{0.4n} \text{ instances, and} \\ 4n^3 & \text{for the rest, i.e., } 2^n - 2^{0.4n} \text{ instances} \end{cases}$$

$$\Rightarrow \Sigma_{|x|=n} \text{time}_A(x) \times D_U(x) \le 5n$$

$$\text{time}_B(x) = \text{time}_A(x)^2$$

$$\Rightarrow \Sigma_{|x|=n} \text{time}_B(x) \times D_U(x) = \text{exponential}$$

1. Framework for average-case P

input distribution: over $\{0, 1\}^*$

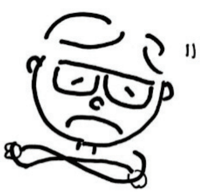
Ex $D_U(x) = c2^{-n} / n^2$

Def average-case poly. time computability [Levin86]

A is **poly. time on average**

$\Leftrightarrow$ there exists some const. $k > 0$ such that

$$\sum_x \frac{\mathrm{time}_A(x)^{1/k} \times D(x)}{n} < \text{const.}$$



(L, D) is **poly. time solvable on average**

$\Leftrightarrow$ there exists some A such that

(i) $A = L$ and (ii) A is poly. time on average.

1. Framework for average-case P

input distribution: over $\{0, 1\}^n$ for each n

Ex $D_U(x) = 2^{-n}$

heuristic poly. time

Def average-case poly. time computability [folkrore]

(L, D) is **poly. time solvable on average**

$\Leftrightarrow$ there exists some poly. time A such that

$$D(\{x : L(x) \neq A(x)\}) < 1/\nu(n)$$

for some superpoly ν.

randomized

$\forall$ poly. p $\overset{\infty}{\forall} n\, [\, \nu(n) > p(n)\,]$

1. Framework for average-case P

Def

distNP = a class of dist. problems (L, D) such that
(i) D is P-comp (= poly. time computable),
(ii) L is in NP.

aveP = a class of dist. problems (L, D) such that
(i) D is P-comp, and
(ii) (L, D) poly. time solvable on average in the sense of [Levin].

heurP = a class of dist. problems (L, D) such that
(i) D is P-comp, and
(ii) (L, D) poly. time solvable on average in the sense of [folkroe].

2. Worst-case vs. average-case

Conjecture

$\boxed{\text{distNP} \not\subseteq \text{heurP}}$

by def.

Conjecture

$\boxed{P \neq NP}$

ご本家

Thm almost equivalent in high comp. classes

1. dist#P = heurP ⇔ #P = BPP
2. distPSPACE = heurP ⇔ PSPACE = BPP
3. distEXP = heurEXP ⇔ EXP = BPP

2. Worst-case vs. average-case

Thm almost equivalent in high comp. classes

1. dist#P = heurP ⇔ #P = BPP
2. distPSPACE = heurP ⇔ PSPACE = BPP
3. distEXP = heurEXP ⇔ EXP = BPP

[IP = PSPACE, MIP = NEXP]

Ex (Perm, U) in heurP ⇒ Perm in BPP [Lipton89]

Notation $D_n(x) = 2^{-n} = U(x)$

Proof ☆1 Suppose that some algorithm A computes Perm on ave. under the uniform dist. U. Then we design our algorithm for computing perm(M) for n x n matrix M as follows:

degree n poly.

random matrix

generate a random matrix R, and
define $g(x) = \text{perm}(M + x^*R)$
compute $g(1), g(2), \ldots, g(n+1)$ by using A
obtain $g(x)$ and compute $g(0) = \text{perm}(M)$

2. Worst-case vs. average-case

Conjecture

distNP $\not\subseteq$ heurP

by def.

[Gudfreund-TaShma07, etc.]

Conjecture

P $\neq$ NP

ご本家

Thm almost equivalent in high comp. classes

1. dist#P = heurP $\Leftrightarrow$ #P = BPP
2. distPSPACE = heurP $\Leftrightarrow$ PSPACE = BPP
3. distEXP = heurEXP $\Leftrightarrow$ EXP = BPP

3. Reducibility for the average-case

Idea for $A \leqq^{P}_{m} B$ *A* is reducible to *B*

$\boxed{\text{Solve } A \text{ by using algo. for } B}$

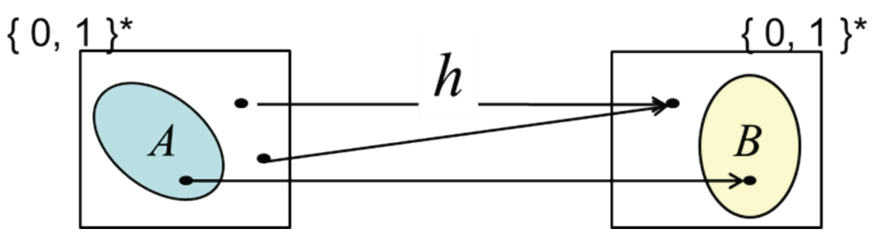
what else?

$B \in P \Rightarrow A \in P$ Then we have

$\boxed{\text{A's difficulty} \leqq \text{B's difficulty}}$

without answering A, B $\overset{?}{\in}$ P

	P	not P ?
B	○	×
A	○	○ or ×



Def

- x in $A \iff h(x)$ in B
- h is P-time computable

Program for B

入力例 x

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

tine

or 1

Thm

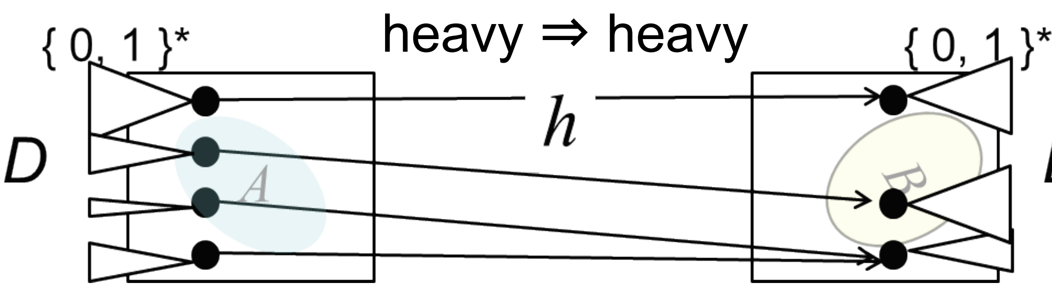
$\boxed{A \leqq^{P}_{m} B \ \&\ B \in P \Rightarrow A \in P}$

3. Reducibility for the average-case

Idea for $A \leq B$ $(A,D) \leq_m^{aP} (B,E)$ — *A* is ave. reducible to *B*

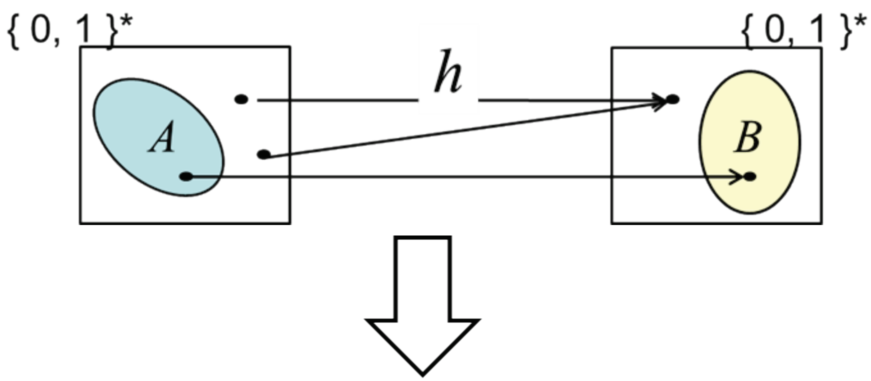
Solve *A* by using algo. for *B*

what else?



∃ *k* such that ☆2

$$\sum_{h(x)=y} \frac{D(x)}{n^k} \leq E(y)$$




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

Def for $\leq_m^{aP}$

- x in A ⇔ $h(x)$ in B
- h is P-time computable
- preserve weights
  heavy ⇒ heavy

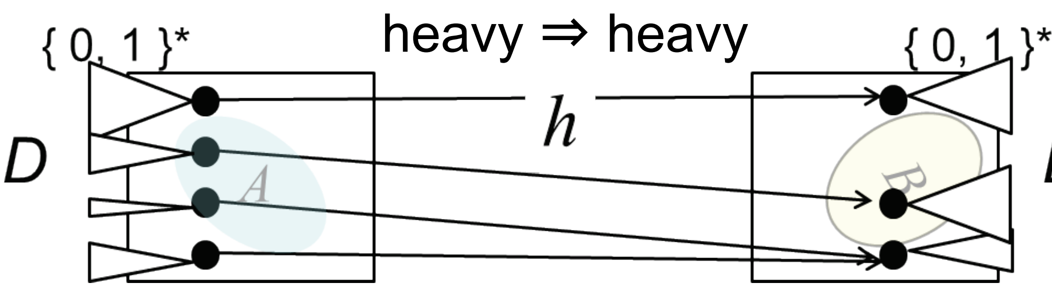
Thm

$(A,D) \leq_m^{aP} (B,E)$ & $B \in$ heurP
⇒ $A \in$ heurP

3. Reducibility for the average-case

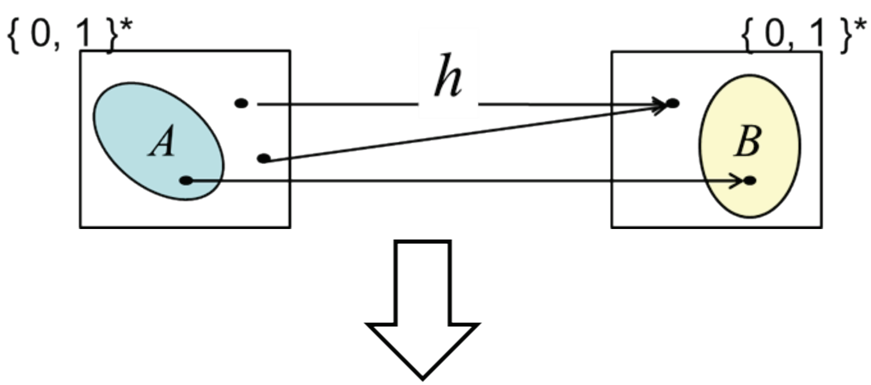
Idea for $(A,D) \leqq_m^{aP} (B,E)$ *A* is ave. reducible to *B*

Solve *A* by using algo. for *B*



∃ *k* such that

$$\sum_{h(x)=y} \frac{D(x)}{n^k} \leq E(y)$$




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

Proof

$D(\{\, x : \text{IsA errs} \,\})$
$\leq E(\{\, y : \text{IsB errs} \,\}) \times \text{poly}(n)$
$\leq 1 / \text{superpoly}(n)$

Thm

$(A,D) \leqq_m^{aP} (B,E)$ & $B \in \text{heurP}$
$\Rightarrow A \in \text{heurP}$

3. Reducibility for the average-case

Ex $\forall L$ in NP [$(L, \mathrm{U}) \leq_{\mathrm{m}}^{\mathrm{aP}} (\mathrm{K}, \mathrm{U_K})$]

- $\mathrm{U}(x) = 2^{-n}$
- $\mathrm{K} = \{ (M, x, 0^t) : M(x)$ has a length t acc. path $\}$
- $\mathrm{U_K}((M, x, 0^t)) = 2^{-m} \times 2^{-n}$, $m = | M |$, $n = | x |$

☆3

Proof. Consider any L in NP. Let M_L be its nondet. algo., and let p_0 be its poly. time bound.

Then h_L defined below satisfies the required conditions.

$$h_L(x) = (M_L, x, 0^{p_0(n)})$$

✔ x in $L \Leftrightarrow h_L(x)$ in K
✔ h_L is P-time computable
- preserve weights

x in $L \Leftrightarrow M_L(x)$ accepts x in $p_0(n)$ steps
$\Leftrightarrow (M_L, x, 0^{p_0(n)})$ in K $\Leftrightarrow h_L(x)$ in K

3. Reducibility for the average-case

Ex $\forall L$ in NP [$(L, \mathrm{U}) \leq^{\mathrm{aP}}_{\mathrm{m}} (\mathrm{K}, \mathrm{U}_{\mathrm{K}})$]

- $\mathrm{U}(x) = 2^{-n}$
- $\mathrm{K} = \{ (M, x, 0^t) : M(x)$ has a length t acc. path $\}$
- $\mathrm{U}_{\mathrm{K}}((M, x, 0^t)) = 2^{-m} \times 2^{-n}$, $m = |M|$, $n = |x|$

☆3

Proof. $h_L(x) = (M_L, x, 0^{p_0(n)})$

✔ x in $L \Leftrightarrow h_L(x)$ in K
✔ h_L is P-time computable
✔ preserve weights

$\mathrm{U}(x) = 2^{-n}$

$\mathrm{U}_{\mathrm{K}}((M_L, x, 0^{p_0(n)})) = 2^{-m_0} \times 2^{-n}$

$\therefore \mathrm{U}(x) \leq \mathrm{U}_{\mathrm{K}}(h_L(x)) \times 2^{m_0}$

$$\frac{\mathrm{U}(x)}{\text{const.}} \leq \mathrm{U}_{\mathrm{K}}(y), \text{ where } y = h_L(x)$$

$$\boxed{\sum_{h_L(x)=y} \frac{\mathrm{U}(x)}{n^{O(1)}} \leq \mathrm{U}_{\mathrm{K}}(y)}$$

3. Reducibility for the average-case

Ex $\forall L$ in NP, $\forall D$: P-dist $[\ (L, D) \leq_m^{aP} (K, U_K)\ ]$

☆3

- $K = \{ (M, x, 0^t) : M(x)$ has a length t acc. path $\}$
- $U_K((M, x, 0^t)) = 2^{-m} \times 2^{-n}$, $m = |M|$, $n = |x|$

Idea

$h_{L,D}(x) = (M_{L,D}, z_D(x), 0^{p_1(n)})$

where $z_D(x)$ is defined so that

(1) $D(x) = 2^{-|z_D(x)|}$ $\Rightarrow$ $h_{L,D}(x)$ is as heavy as x

(2) $z_D(x)$ determines x uniquely, and $\boxed{z_D(x) \to x}$ is easy.

✔ x in $L \Leftrightarrow h_{L,D}(x)$ in K

\- $h_{L,D}$ is P-time computable

✔ preserve weights

how

cumulative prob.

lex. order

$F(x_1) = \boxed{0.10}1101$

$F(x_2) = \boxed{0.100}101$

$F(x_3) = \boxed{0.}100010$

$F(x_4) = 0.011001$

$D(x_1) = 0.001 \leq 2^{-3}$

$D(x_2) = 0.000011 \leq 2^{-4}$

$D(x_3) = 0.001001 \leq 2^{-2} \leq 2^{-1}$

$z_D(x_1) = 010$

$z_D(x_3) = 0$

3. Reducibility for the average-case

Ex $\forall L$ in NP, $\forall D$: P-dist $[\ (L, D) \leq_m^{aP} (\mathrm{K}, \mathrm{U}_\mathrm{K})\ ]$

Thm [Levin86]

$(\mathrm{K}, \mathrm{U}_\mathrm{K})$ is complete for distNP.

4. Search vs. decision

Def. the standard NP problem = decision problem

NP problem is to decide $x \in L$ for a set L that can be characterized as follows

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

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

証拠 witness

witness checking predicate

Def. NP search problem

NP search problem (w.r.t. R_L) is to compute a witness for $x \in L$ ($\perp$ if no witness exists).

4. Search vs. decision

Thm search vs. decision in the worst-case

$$\boxed{\text{NPsearch} \subseteq \text{P} \iff \text{P} = \text{NP}}$$

Proof Consider any R_L. Define prefix$L$ by

$$\text{prefix}L = \{\, (x, u) : \exists v\, [\, R_L(x, uv)\,]\,\}.$$

Then the following algorithm works (below ε is the empty str.).

program witness search for R_L(input x);
    if $(x, \varepsilon) \notin$ prefix$L$ then output $\perp$ & halt;
    $w \leftarrow \varepsilon$;
    while true do {
        if $R_L(x, w)$ then output w & halt;
        if $(x, w0) \in$ prefixL then $w \leftarrow w0$
                                        else $w \leftarrow w1$
    }

this may not work in the average-case

4. Search vs. decision

Thm search vs. decision in the worst-case

$$\boxed{\text{NPsearch} \subseteq \text{P} \;\Leftrightarrow\; \text{P} = \text{NP}}$$

Thm search vs. decision in the average-case

$$\boxed{\text{NPsearch} \subseteq \text{BPP} \;\Leftrightarrow\; \text{distNP} \subseteq \text{heurP}}$$

[BenDavid-Chor-Goldreich-Luby92]

how

By Isolation technique

4. P-samplable dist. & one-way func.

Def poly. time computable distribution

A distribution D is **P-comp** (= poly. time comp. dist.)

$\Leftrightarrow$ $F(x) = \Sigma_{y<x} D(y)$ is poly. time computable

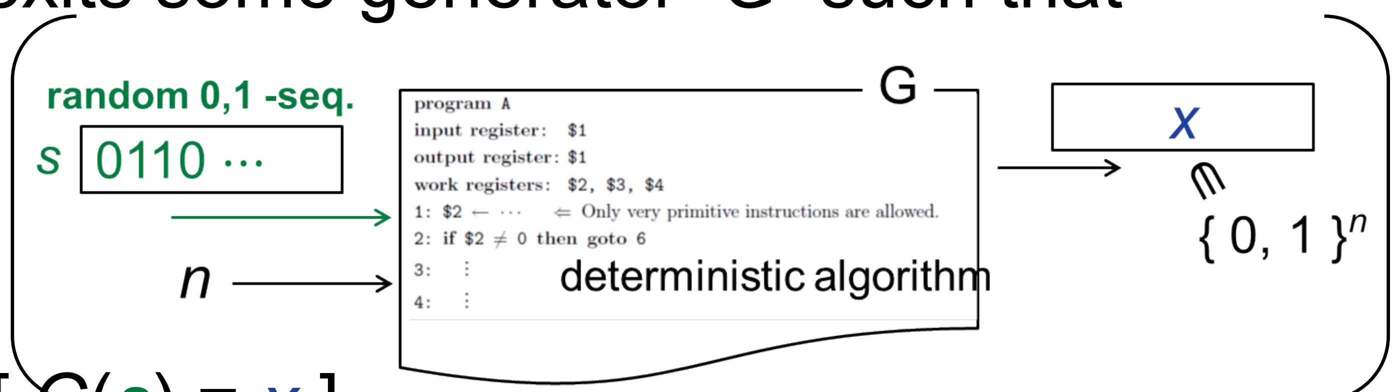
natural?

Def poly. time samplable distribution

A distribution D is **P-samplable**

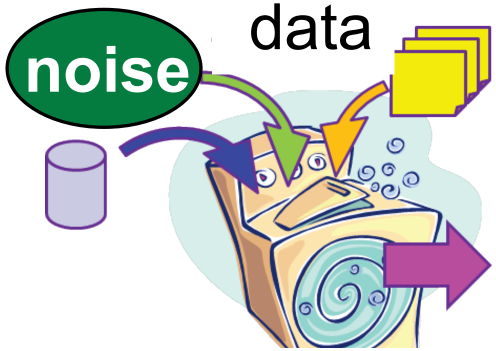
$\Leftrightarrow$ there exits some generator G such that

$D(x) = \Pr_{s}$



$= \Pr[G(s) = x]$

4. P-samplable dist. & one-way func.



our input data is P-samplable

Let us consider the search version!

machine learning

planted solution (specified by G, poly. time computable)

Input:	$v = G(u, s)$ for random $s \in \{0, 1\}^n$
task:	compute the planted solution u

inverting poly. time computable function f

cryptography

Input:	$y = f(x)$ for random $x \in \{0, 1\}^n$
task:	compute the inverse image x

4. P-samplable dist. & one-way func.

inverting poly. time computable function f cryptography

Input: $y = f(x)$ for random $x \in \{0, 1\}^n$
task: compute the inverse image x

Conjecture

one-way function exits:
$\exists$ poly. time computable function f such that
no poly. time algorithm A satisfies

$$\Pr[\ A(\boxed{f(x)}) = x\ ] > 1 - \nu(n)$$

f^{-1} is not poly. time computable on average

P-samplable dist.

distNP $\not\subseteq$ heurP

4. P-samplable dist. & one-way func.

Thm

- P-comp ⇒ P-samplable
- P-samplable $\not\Rightarrow$ P-comp unless #P = P

Thm [Impagliazzo-Levin90]

$\forall L$ in NP, $\forall D$: P-samp. $[\ (L, D) \leqq^{aP}_{m} (K, U_K)\ ]$

Cor

distNP ＝heurP ⇒ distNP = heurP under P-samp

Cor

distNP ＝heurP ⇒ no one-way function exists

4. P-samplable dist. & one-way func.

Thm   Proof Idea: ①, ②

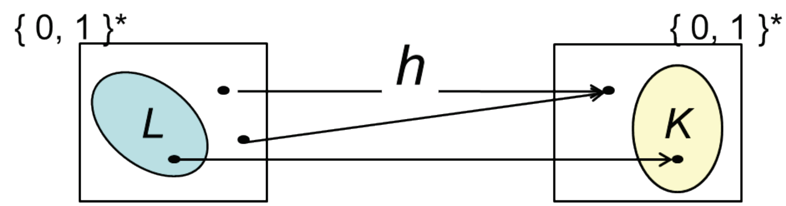
$\forall L$ in NP, $\forall D$: P-samp. [$(L, D) \leqq^{\mathrm{aP}}_{\mathrm{m}} (\mathrm{K}, \mathrm{U}_\mathrm{K})$]

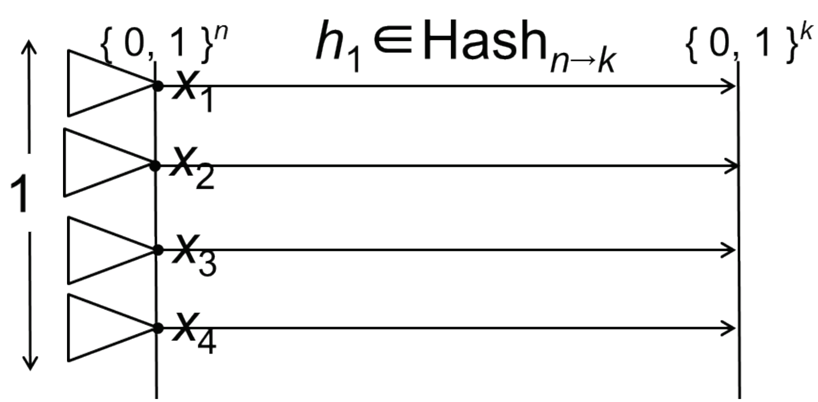
☆3

- $\mathrm{K} = \{ (M, x, 0^t) : M(x)$ has a length t acc. path $\}$
- $\mathrm{U}_\mathrm{K}((M, x, 0^t)) = 2^{-m} \times 2^{-n}$, $m = |M|$, $n = |x|$

Consider a heavy input x for L.
I.e., $\Pr[G(s) = x] = 2^{-k}$ for $k << n$

Idea:① Map such inputs to a string of length k by a random hash.





✔ x in $L \Leftrightarrow h_{L,D}(x)$ in K
\- $h_{L,D}$ is P-time computable
✔ preserve weights

heavy ⇒ heavy

at most 2^k ⇒ almost 1-1

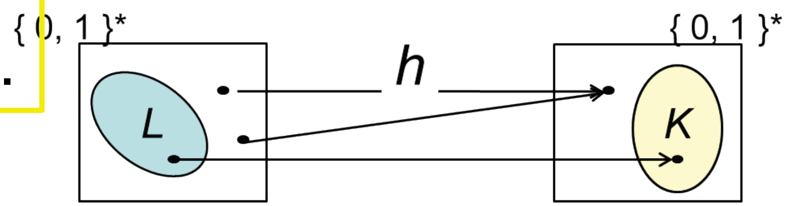
4. P-samplable dist. & one-way func.

Thm   **Proof Idea: ①, ②**

$\forall L$ in NP, $\forall D$: P-samp. $[\ (L, D) \leqq^{aP}_{m} (\mathrm{K}, \mathrm{U}_{\mathrm{K}})\ ]$

Problem may occur if small weight inputs are mapped to some length k inputs of K.

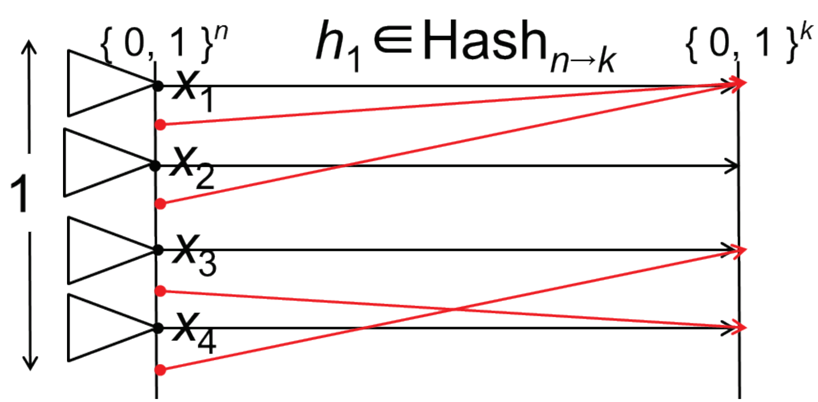
Idea ② Use the 2nd random h_2 to confirm the weight is large enough.



- x in $L \Leftrightarrow h_{L,D}(x)$ in K
- $h_{L,D}$ is P-time computable
- preserve weights

heavy ⇒ heavy

Idea ① Map such inputs to a string of length k by a random hash.



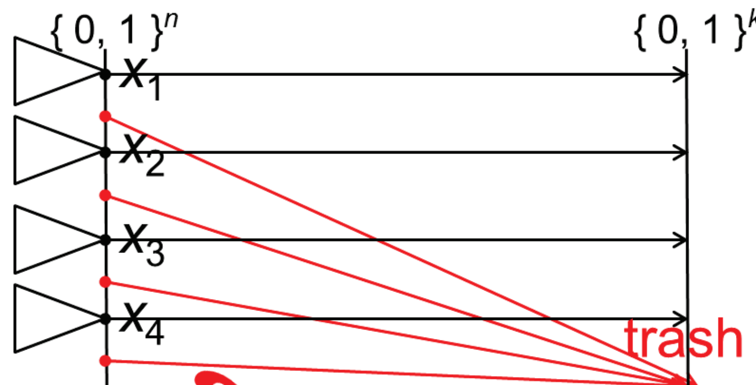
at most 2^k ⇒ almost 1-1

4. P-samplable dist. & one-way func.

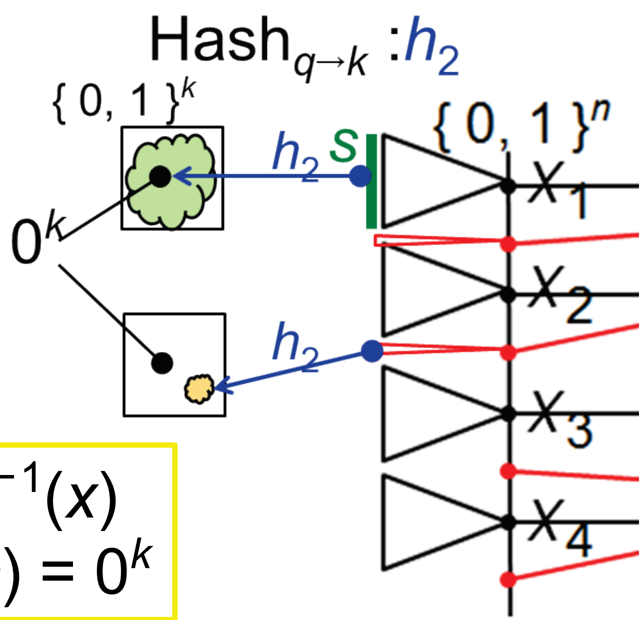
Thm   Proof Idea: ①, ②

$\forall L$ in NP, $\forall D$: P-samp. [$(L, D) \leqq^{aP}_{m} (K, U_K)$]

Problem may occur if small weight inputs are mapped to some length k inputs of K.



Idea ② Use the 2nd random h_2 to confirm the weight is large enough.



- x in $L \Leftrightarrow h_{L,D}(x)$ in K
- $h_{L,D}$ is P-time computable
- preserve weights

heavy ⇒ heavy

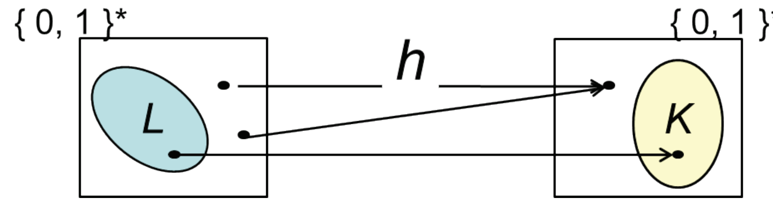
$\exists\, s \in G^{-1}(x)$
s.t. $h_2(s) = 0^k$

at most 2^k ⇒ almost 1-1

4. P-samplable dist. & one-way func.

Thm Proof Idea: ①, ②

$\forall L$ in NP, $\forall D$: P-samp. [$(L, D) \leq_m^{aP} (K, U_K)$]



Idea ① Map such inputs to a string of length k by a random hash.

Idea ② Use the 2nd random h_2 to confirm the weight is large enough.

- x in $L \Leftrightarrow h_{L,D}(x)$ in K
- $h_{L,D}$ is P-time computable
- preserve weights

$$h_{L,D}(x) = (\, M_{L,D}, (k, \underbrace{h_1(x)}_{k}, h_1, h_2), 0^t \,)$$

randomly chosen from { 0,...,n }

random hash

<u>What $M_{L,D}$ checks on ($k, h_1(x), h_1, h_2$)</u>

$\exists$ s: a seed for G such that

- $h_1(x)$ in L (where $x := G(s)$), and
- $h_2(s) = 0^m$

NP comp.
= nondet. and
in some poly. time t

4. P-samplable dist. & one-way func.

Thm

$\forall L$ in NP, $\forall D$: P-samp. $[\ (L, D) \leqq^{aP}_{m} (K, U_K)\ ]$

Cor

distNP $=$ heurP $\Rightarrow$ distNP = heurP under P-samp

Cor

distNP $=$ heurP $\Rightarrow$ no one-way function exists

Big Open Queston

distNP $\neq$ heurP $\Rightarrow$ one-way function exists

Let's try this !!

0. Contents

1. Framework for average-case NP vs. P
2. Worst-case vs. average-case
3. Reducibility for the average-case
4. Search vs. decision
5. P-samplable dist. and one-way function

Fin

key reference

1. Jie Wang's Average-case complexity forum
2. Du and Ko, Theory of Computational Complexity John-Wisely Sons, Inc., 2000.

☆k ∴ See slides from the next for additional explanation.

Appendix: Detail Explanations ☆k

☆1 (Perm, U) in heurP ⇒ Perm in BPP

what this means? For our length-wise distribution framework, it would be easier if we use two size parameters n and k, where n x n is the size of a given matrix and k is the bit length of each matrix element. Thus, our uniform distribution is defined by

$$U_{n,k}(M) = 2^{kn^2}$$

for all n x n matrices M whose entries are (at most) k bit nonnegative integers. (For simplicity, let us consider only nonnegative integers.)

how to design worst-to-average reduction ? The idea and the outline are the same as those explained in the talk. The difference is to choose R following, say, $U_{n,k}$ and use mod $2^{k+2\log n}$ computation for computing $M + x^*R$. (The permutation itself should be computed in Z.) Then we can follow the same proof outline to show that the reduction works to use the assumed heurP algorithm for Perm under $\{ U_{n,k} \}_{n,k}$

☆2 average-case reducibility (for length-wise dist.)

$\exists\, k$ such that

$$\sum_{h(x)=y} \frac{D(x)}{n^k} \leq E(y)$$

The right expression is the dominancy cond. for the reduction form (A, D) to (B, E) in the Levin's framework. For our length-wise dist. framework, we had better modify it slightly.
Well, the modification is easy. We simply need to consider the following policy.

> Design reductions so that its output length is uniquely (and easily) determined by input length *n*.

This topic is for preparation for ☆3

extending the notion of dominancy: motivation

Consider the case where inputs for the problem B is a random graph following the standard distribution G(n,1/2), that is, for a given n, we assume that a graph G with n vertices is generated randomly by adding each edge with prob. 1/2 independently. Then the prob. $E(G)$ of each graph G is quite small, and it is usually impossible to design a reduction satisfying the above dominancy condition. But still the problem seems hard and we want to show its hardness. Thus, we consider some extension of our dominancy condition.

☆2 average-case reducibility (for length-wise dist.)

extending the notion of dominancy: how to

We use randomized reductions to avoid this technical difficulty. That is, we allow a reduction h to take random seed s of length p(n) for some appropriately chosen polynomial p, and extend our goal as follows:

- x in A ⇔ $h(x)$ in B
- h is P-time computable
- dominancy cond.

  ∃ k such that

$$\sum_{h(x)=y} \frac{D(x)}{n^k} \leq E(y)$$

⇒

- x in A ⇔ $h(x,s)$ in B <u>for all s</u>
- h is P-time computable
- dominancy cond.

  ∃ k such that

$$\sum_{h(x,s)=y} \frac{D(x) \cdot 2^{-|s|}}{n^k} \leq E(y)$$

With this generalization, it is possible to handle the case where $E(y)$ gets exponentially smaller than $D(x)$.

We may even relax this condition to "many" (i.e., 2/3 of all).

☆3

- K = { (M, x, 0^t) : $M(x)$ has a length t acc. path }
- U_K((M, x, 0^t)) = $2^{-m} \times 2^{-n}$, $m = |M|$, $n = |x|$

This definition for K and U_K is not appropriate, though it tells the intuitive idea of what we want. Below we give some definition for our length-wise distribution framework, which implements this idea more appropriately.

m	n	t	M	x	pad

K = { (m,n,t,M, x,pad) :
m, n, t are some positive integers,
M is a nondet. Turing machine,
$|M| = m$, $|x| = n$, $|pad| = t + d$, and
$M(x)$ has an accepting path of length t }

- m, n, t, M, x, pad are all bin. strings, i.e., elements of $\{0, 1\}^*$
- Use double-bit-coding for m, n, t so that some delimiter (e.g., 01) can be used. For example, (3,5,2,M,x,pad) is encoded as 11001101111000001101110001$Mxpad$.
- By choosing d appropriately, to adjust the total length (denoted by n_0) to any number satisfying

$$n_0 > 2(\log m + \log n + \log t + 3) + m + n + t \qquad \leftarrow (1)$$

☆3

- $K = \{ (M, x, 0^t) : M(x)$ has a length t acc. path $\}$
- $U_K((M, x, 0^t)) = 2^{-m} \times 2^{-n}$, $m = |M|$, $n = |x|$

$K = \{ (m,n,t,M, x,pad)$:

m	n	t	M	x	pad

m, n, t are some positive integers,
M is a nondet. Turing machine,
$|M| = m$, $|x| = n$, $|pad| = t + d$, and
$M(x)$ has an accepting path of length t $\}$

Now U_K is simply the uniform distribution over $\{0, 1\}^{n_0}$ for each n_0.

The reduction from (L,U) to (K,U_K)

We also need to modify our reduction. Let M_L and p_0 be a nondet. TM its polynomial time bound. For each n, let n_0 be a number satisfying (1) of the previous slide; we may assume that $n_0 < q_0(n)$ for some poly. q_0. We change the definition of our reduction as follows. It is not so difficult to check that this reduction satisfies the extended dominancy cond.

$h(x) = (M_L, x, 0^{p_0(n)})$ $\Rightarrow$ $h(x,s) =$

m	n	$p_0(n)$	M_L	x	pad

$\longleftarrow n'_0 \longrightarrow$

Here pad is the prefix of s of length $n_0 - n'_0$.