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

☆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.

**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**

\[
= \sum_x \text{time}_A(x) \times D(x) \quad \Pr[ x \text{ is given as an input }]
\]
1. Framework for average-case P

input distribution?

Notation

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

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

\[
D_U( x ) = \frac{c2^{-n}}{n^2} \quad \text{over} \{0, 1\}^* \\
D_U( x ) = 2^{-n} \quad \text{over} \{0, 1\}^n \text{ for each } n
\]

Note!

How shall we define input distribution range?

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

\[ D(\{0, 1\}^* ) = 1 \quad \text{or} \quad D(\{0, 1\}^n ) = 1 \]

unfriendly

less beautiful (mathematically)
1. Framework for average-case $\mathbf{P}$

**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 $\mathbf{P}$-dist (poly. time comp. dist.)

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

cumulative probability
1. Framework for average-case P

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

\[ D_U(x) = \frac{c2^{-n}}{n^2} \]

Def ??

expected. comp. time ??

A's expected time? = \( \Sigma \) time\(_A\)(x) \times D(x) \)

Algorithm \( A \) is polynomial time on average

\[ \left\{ \begin{array}{ll}
\Sigma_{|x|=n} \text{time}_A(x) \times D(x) = n^{O(1)} = n^{c}
\end{array} \right. \]

Ex

\[ \text{time}_A(x) = \left\{ \begin{array}{ll}
2^{0.5n} & \text{for } 2^{0.4n} \text{ instances, and } \\
4n^3 & \text{for the rest, i.e., } 2^n - 2^{0.4n} \text{ instances }
\end{array} \right. \]

\[ \Rightarrow \Sigma_{|x|=n} \text{time}_A(x) \times D_U(x) \leq 5n \]

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

\[ \Rightarrow \Sigma_{|x|=n} \text{time}_B(x) \times D_U(x) = \text{exponential} \]
1. Framework for average-case P

**Input distribution:** over \{0, 1\}^*

**Example:** \(D_U(x) = \frac{c2^{-n}}{n^2}\)

**Definition:** average-case poly. time computability [Levin86]

A is **poly. time on average**

\[\iff \exists \text{ some } k > 0 \text{ such that } \sum_x \frac{\text{time}_A(x)^{1/k} \times D(x)}{n} < \text{const.}\]

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

\[\iff \exists A \text{ such that (i) } A = L \text{ and (ii) } A \text{ is poly. time on average.}\]
1. Framework for average-case $\mathbb{P}$

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

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

Heuristic poly. time
Average-case poly. time computability [folklore]

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

\[ D\left( \{ x : L(x) \neq A(x) \} \right) < 1 / \nu(n) \]

for some superpoly \( \nu \).

Randomized

\[ \forall \text{poly. } \rho \forall n \left[ u(n) > \rho(n) \right] \]
Def

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

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

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

Conjecture

\[ \text{distNP} \not\subseteq \text{heurP} \]

by def.

Conjecture

\[ P \neq NP \]

Thm almost equivalent in high comp. classes

1. \[ \text{dist}\#P = \text{heurP} \iff \#P = \text{BPP} \]
2. \[ \text{distPSPACE} = \text{heurP} \iff \text{PSPACE} = \text{BPP} \]
3. \[ \text{distEXP} = \text{heurEXP} \iff \text{EXP} = \text{BPP} \]
2. Worst-case vs. average-case

Thm almost equivalent in high comp. classes

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

[IP = PSPACE, MIP = NEXP]

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

Proof 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:

- generate a random matrix R, and define \( g(x) = \text{perm}(M + xR) \)
- 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

\[
\text{distNP} \not\subseteq \text{heurP}
\]

by def.

by def.

Conjecture

\[
P \neq \text{NP}
\]

almost equivalent in high comp. classes

Thm

1. \(\text{dist}\#P = \text{heurP} \iff \#P = \text{BPP}\)
2. \(\text{distPSPACE} = \text{heurP} \iff \text{PSPACE} = \text{BPP}\)
3. \(\text{distEXP} = \text{heurEXP} \iff \text{EXP} = \text{BPP}\)

[Gudfreund-TaShma07, etc.]
3. Reducibility for the average-case

Idea for $A \leq^P_m B$  
A is reducible to $B$

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

If $B \in P \Rightarrow A \in P$

Then we have:

<table>
<thead>
<tr>
<th>P or not P?</th>
<th>$B$</th>
<th>$A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\times$</td>
<td>$O$</td>
<td>$O$</td>
</tr>
<tr>
<td>$O$</td>
<td>$O$</td>
<td>$O$ or $\times$</td>
</tr>
</tbody>
</table>

Def

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

Thm

$A \leq^P_m B \& B \in P \Rightarrow A \in P$
3. Reducibility for the average-case

Idea for $\langle A, \leq \rangle \leq_{aP} \langle B, E \rangle$:

- $A$ is average reducible to $B$
- Solve $A$ by using algo. for $B$

\[ \exists k \text{ such that } \frac{\sum D(x)}{n^k} \leq E(y) \]

Def for $\leq_{aP}$:
- $x \in A \iff h(x) \in B$
- $h$ is P-time computable
- Preserve weights

Thm:
- $\text{heavy} \Rightarrow \text{heavy}$

\[ (A, D) \leq_{aP} (B, E) \& B \in \text{heurP} \Rightarrow A \in \text{heurP} \]
3. Reducibility for the average-case

Idea for \((A,D) \preceq^\text{ap}_m (B,E)\)  \(\Rightarrow\) \(A\) is ave. reducible to \(B\)

Solve \(A\) by using algo. for \(B\)

\[
\exists k \text{ such that } \sum_{h(x)=y} \frac{D(x)}{n^k} \leq E(y)
\]

Proof

\[
D( \{ x : \text{IsA errs} \} ) \leq E( \{ y : \text{IsB errs} \} ) \times \text{poly}(n) \leq \frac{1}{\text{superpoly}(n)}
\]

Thm

\[(A,D) \preceq^\text{ap}_m (B,E) \land B \in \text{heurP} \Rightarrow A \in \text{heurP}\]
3. Reducibility for the average-case

\[ \forall L \text{ in NP } [ (L, U) \leq_{ap}^m (K, U_K) ] \]

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

**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 \( L \iff M_L(x) \) accepts \( x \) in \( p_0(n) \) steps

\( \iff (M_L, x, 0^{p_0(n)}) \) in \( K \iff h_L(x) \) in \( K \)

- \( x \) in \( L \iff h_L(x) \) in \( K \)
- \( h_L \) is P-time computable
- preserve weights
3. Reducibility for the average-case

Ex\(\forall L \text{ in NP} \ [ \ ( L, U ) \leq^a_{m} ( K, U_{K} ) ] \)

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

Proof. \(h_{L}( x ) = ( M_{L}, x, 0^{p_{0}(n)} )\)

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

\(U_{K}( ( M_{L}, x, 0^{p_{0}(n)} ) ) = 2^{-m_{0}} \times 2^{-n}\)

\(\therefore U( x ) \leq U_{K}( h_{L}( x ) ) \times 2^{m_{0}}\)

\(\frac{U( x )}{\text{const.}} \leq U_{K}( y ), \text{ where } y = h_{L}( x )\)

\[\sum_{h_{L}(x)=y} \frac{U(x)}{n^{O(1)}} \leq U_{K}( y )\]
3. Reducibility for the average-case

\[ \forall L \text{ in NP}, \forall D: \text{P-dist } [ (L, D) \leq_{aP} (K, U_K) ] \]

- \( K = \{ (M, x, 0^t) : M(x) \text{ has a length } t \text{ acc. path} \} \)

- \( U_K((M, x, 0^t)) = 2^{-m} \times 2^{-n}, m = |M|, n = |x| \)

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 \( z_D(x) \rightarrow x \) is easy.

\( F(x_1) = 0.101101 \)
(\( z_D(x_1) = 010 \))

\( F(x_2) = 0.100101 \)

\( F(x_3) = 0.100010 \)

\( F(x_4) = 0.011001 \)

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

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

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

\( z_D(x_3) = 0 \)
3. Reducibility for the average-case

Ex \( \forall L \in \text{NP}, \ \forall D: \text{P-dist} \ [ (L, D) \leq^a_{m} (K, U_K) ] \)

Thm [Levin86]

(\( K, U_K \)) is complete for \text{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

\[
\forall x \in \{0, 1\}^* \quad \begin{cases} 
    x \in L & \Rightarrow \exists w : |w| \leq q_L(|x|) \left[ R_L(x, w) = 1 \right] \\
    x \notin L & \Rightarrow \forall w : |w| \leq q_L(|x|) \left[ R_L(x, w) = 0 \right]
\end{cases}
\]

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

**Def.** NP search problem

NP search problem (w.r.t. \( R_L \)) is to compute a witness for \( x \in L \) (⊥ if no witness exists).
4. Search vs. decision

Thm search vs. decision in the worst-case

NPsearch \subseteq P \iff P = NP

Proof Consider any \( R_L \). Define prefixL by

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

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

program witness search for \( R_L(\text{input } x) \);
if \( (x, \epsilon) \notin \text{prefixL} \) then output \( \bot \) & halt;
\( w \leftarrow \epsilon \);
while true do {
if \( R_L(x, w) \) then output \( w \) & halt;
if \( (x, w0) \in \text{prefixL} \) 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

\[ \text{NPsearch} \subseteq \text{P} \iff \text{P} = \text{NP} \]

Thm search vs. decision in the average-case

\[ \text{NPsearch} \subseteq \text{BPP} \iff \text{distNP} \subseteq \text{heurP} \]

[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.)

\[ F(x) = \sum_{y < x} D(y) \] is poly. time computable

**Def** poly. time samplable distribution

A distribution $D$ is **P-samplable**

\[ D(x) = \Pr_{s \in \{0,1\}^n} \left[ \text{random 0,1 seq.} s \right. \]

\[ \rightarrow \text{n \ deterministic algorithm} \]

\[ \left. \rightarrow G(s) = x \right] \]
Let us consider the search version!

**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** \( f \)

**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 $f$

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

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

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

P-samplable dist.

distNP $\subseteq$ heurP
4. P-samplable dist. & one-way func.

Thm

- P-comp ⇒ P-samplable
- P-samplable ⇒ P-comp unless #P = P

Thm [Impagliazzo-Levin90]

∀ L in NP, ∀ D: P-samp. [ (L, D) \leq^a_{\text{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  ∀ \( L \in \text{NP} \), ∀ \( D: \text{P-samp.} \) [ \( (L, D) \leq_{m}^{aP} (K, U_{K}) \) ]

\[ K = \{ (M, x, 0^{t}) : M(x) \text{ has a length } t \text{ acc. path} \} \]

\[ U_{K}( (M, x, 0^{t}) ) = 2^{-m} \times 2^{-n}, \quad m = |M|, \quad n = |x| \]

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

Idea: Map such inputs to a string of length \( k \) by a random hash.

\[ x \in L \iff h_{L,D}(x) \in K \]
- \( h_{L,D} \) is P-time computable
- preserve weights

at most \( 2^{k} \Rightarrow \) almost 1-1
4. P-samplable dist. & one-way func.

Thm

Proof Idea: ①, ②

∀ \mathcal{L} \text{ in NP}, \forall \mathcal{D}: \text{ P-samp. } [ ( \mathcal{L}, \mathcal{D} ) \leq_{m}^{aP} ( \mathcal{K}, \mathcal{U}_K ) ]

Problem may occur if small weight inputs are mapped to some length \( k \) inputs of \( \mathcal{K} \).

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

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

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

\[ \text{heavy} \Rightarrow \text{heavy} \]

at most \( 2^k \Rightarrow \text{almost 1-1} \)
4. P-samplable dist. & one-way func.

Thm

Proof Idea: ①, ②

∀ \text{L in NP}, \ ∀ \text{D: P-samp.} \ [ \ (L, D) \leq_{m}^{aP} (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.

∃ \( s \in G^{-1}(x) \) s.t. \( h_2(s) = 0^k \)
4. P-samplable dist. & one-way func.

Thm

∀ L in NP, ∀ D: P-samp. [ ( L, D ) \leq^a P ( K, U_K ) ]

Proof Idea: ①, ②

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.

\[
h_{L,D}(x) = ( M_{L,D}, (k, h_1(x), h_1, h_2), 0^t )
\]

What \( M_{L,D} \) checks on \( (k, h_1(x), h_1, h_2) \)

\[\exists s: \text{a seed for } G \text{ such that}
\]

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

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

NP comp.
= nondet. and
in some poly. time \( t \)
4. P-samplable dist. & one-way func.

Thm
\[ \forall L \text{ in } NP, \forall D: \text{P-samp.} \ [ (L, D) \leq_{m}^{aP} (K, U_{K}) ] \]

Cor
\[ \text{distNP} = \text{heurP} \Rightarrow \text{distNP} = \text{heurP} \text{ under P-samp} \]

Cor
\[ \text{distNP} = \text{heurP} \Rightarrow \text{no one-way function exists} \]

Big Open Queston
\[ \text{distNP} \neq \text{heurP} \Rightarrow \text{one-way function exists} \]

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

key reference
1. Jie Wang's Average-case complexity forum
2. Du and Ko, Theory of Computational Complexity

☆ See slides from the next for additional explanation.
Appendix: Detail Explanations

1. $(\text{Perm, U})$ in heurP $\Rightarrow$ 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 $n \times n$ 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 $n \times n$ 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 $\mathbb{Z}$.) Then we can follow the same proof outline to show that the reduction works to use the assumed heurP algorithm for Perm under $\{U_{n,k}\}_{n,k}$.
average-case reducibility (for length-wise dist.)

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 \(n\).

\[
\exists k \text{ such that } \sum_{h(x)=y} \frac{D(x)}{n^k} \leq E(y)
\]

extending the notion of dominancy: motivation

This topic is for preparation for \(\star 3\)

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

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.

\[
\exists k \text{ such that } \sum_{h(x)=y} \frac{D(x)}{n^k} \leq E(y)
\]

- \( x \) in \( A \) ⇔ \( h(x,s) \) in \( B \) for all \( s \)
- \( h \) is P-time computable
- dominancy cond.

\[
\exists k \text{ such that } \sum_{h(x,s)=y} \frac{D(x)}{n^k} 2^{-|s|} \leq E(y)
\]

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).
• $K = \{ (M, x, 0^t) : M(x) \text{ has a length } t \text{ acc. path } \}$

• $U_K( (M, x, 0^t) ) = 2^{-m} \times 2^{-n} , m = |M|, n = |x|$

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.

$$m \quad n \quad t \quad M \quad x \quad pad$$

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

$m, n, t \text{ are some positive integers,}$

$M \text{ is a nondet. Turing machine,}$

$|M| = m, |x| = n, |pad| = t + d, \text{ and}$

$M(x) \text{ has an accepting path of length } t \}$

• $m, n, t, M, x, pad$ 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 $110011011100001101110001Mxpad$.

• By choosing $d$ appropriately, to adjust the total length (denoted by $n_0$) to any number satisfying

$$n_0 > 2(\log m + \log n + \log t + 3) + m + n + t \quad \leftarrow (1)$$
\( K = \{ (M, x, 0^t) : M(x) \text{ has a length } t \text{ acc. path } \} \)

\( U_K( (M, x, 0^t) ) = 2^{-m} \times 2^{-n} \), \( m = |M|, \ n = |x| \)

\[
K = \{ (m,n,t,M, x, \text{pad}) : \\
\begin{array}{llllll}
m, n, t & \text{are some positive integers,} \\
M & \text{is a nondet. Turing machine,} \\
|M| = m, |x| = n, |\text{pad}| = t + d, \text{ and} \\
M(x) & \text{has an accepting path of length } t \}
\end{array}
\]

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.

\[
\begin{array}{llllll}
h( x ) = (M_L, x, 0^{p_0(n)} ) & \quad \Rightarrow \quad & h( x, s ) = \begin{array}{llllll}
m & n & p_0(n) & M_L & x & \text{pad}
\end{array} \\
& & \begin{array}{llllll}
n' \quad n'_0
\end{array}
\end{array}
\]

Here \( \text{pad} \) is the prefix of \( s \) of length \( n_0 - n'_0 \).