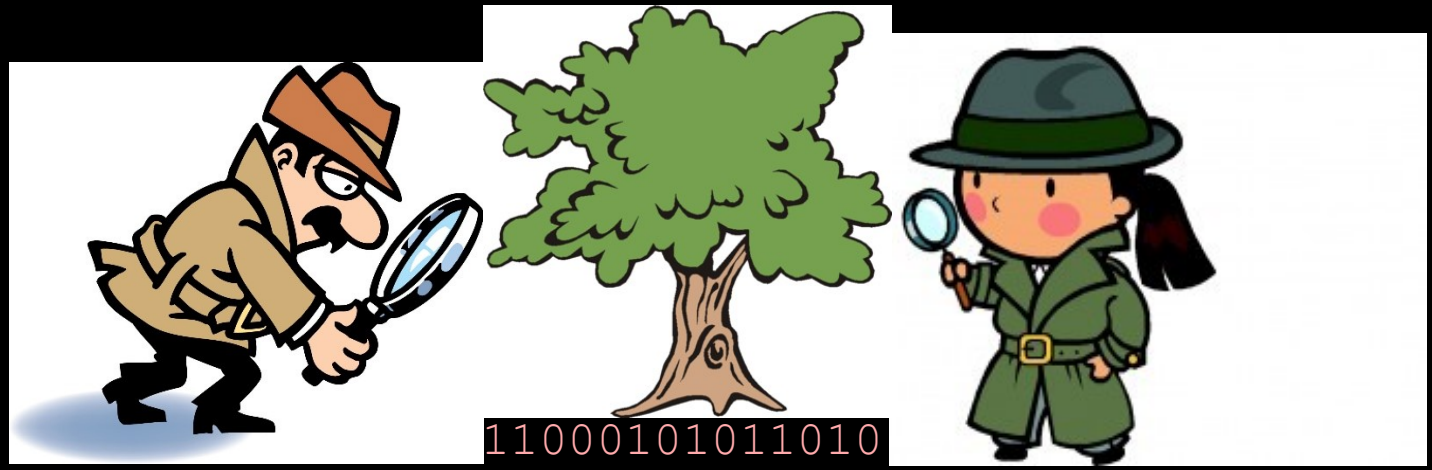
The Mystery of the Missing String



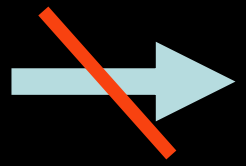
Ryan Williams
(joint work with Nikhil Vyas, ITCS'23)

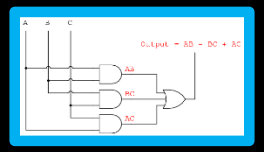
Which Functions Have High Circuit Complexity?

Nearly all of them… "Most" functions require *huge* circuits!

Theorem [Shannon '49, Lupanov '58]

With high probability,
a randomly chosen function $f : \{0,1\}^n \to \{0,1\}$
does not have circuits of size less than $\sim 2^n/n$
(and: every f has a circuit of size about $\sim 2^n/n$)





Which "natural" functions exhibit this exponential behavior?

Which Functions Have High Circuit Complexity?

Rephrase: What is the *"smallest"* complexity class containing a function of ***maximum*** $(2^n/n)$ circuit complexity?

Smallest known class: $E^{\Sigma_2 P} = TIME^{\Sigma_2 P}[2^{O(n)}]$ [1970s]

Theorem: There is a function $f \in E^{\Sigma_2 P}$ that requires ***maximum*** circuit complexity.

Recall: $f \in \Sigma_2 P \Leftrightarrow$ *there is a polynomial* $p(n)$ *and a* $p(n)$*-time verifier* $V(x, y, z)$ *such that for all* x, $f(x) = 1 \Leftrightarrow (\exists\ p\text{-length } y)(\forall\ p\text{-length } z)[V(x, y, z)\ accepts]$

A Canonical Problem in $\Sigma_2 P$:

Circuit Minimization

Input: Boolean circuit C

Decide: Is there a circuit smaller than C that computes the same function?

A Function With Max Circuit Complexity

Theorem: There is a function $f \in E^{\Sigma_2 P}$ that requires ***maximum*** circuit complexity.

Sketch: On $x \in \{0,1\}^\star$ of length n,

// first, find the max circuit complexity $s^\star$ of a function on n inputs

$s \coloneqq 2^n$

while ($\forall\, f : \{0,1\}^n \to \{0,1\}$, f has a circuit of size at most s)

// can formulate this as an $2^{O(n)}$-length query to a $\Sigma_2 P$ oracle

$s \coloneqq s - 1$

// at this point, max circuit complexity $s^\star = s + 1$

Use a "search-to-decision" reduction on the $\Sigma_2 P$ oracle to find the

lexicographically first truth table $T \in \{0,1\}^{2^n}$ that has circuits of size $s^\star$

Output x-th bit of T

Which Functions Have High Circuit Complexity?

Rephrase: What is the *smallest* complexity class containing a function of ***maximum*** $(2^n/n)$ circuit complexity?

Smallest known: $E^{\Sigma_2 P} = TIME^{\Sigma_2 P}[2^{O(n)}]$ [1970s]

Theorem: There is a function $f \in E^{\Sigma_2 P}$ that requires ***maximum*** circuit complexity.
This lower bound relativizes. Let $A: \{0,1\}^\star \to \{0,1\}$ **be an arbitrary "oracle"**

$f \in \Sigma_2 P^A \Leftrightarrow$ *there is a polynomial* $p(n)$ *and a* $p(n)$*-time verifier* $V^A(x, y, z)$ *such that for all* x, $f(x) = 1 \Leftrightarrow (\exists\ p\text{-length } y)(\forall\ p\text{-length } z)[V^A(x, y, z)\ accepts]$

Theorem: There's $f \in E^{\Sigma_2 P^A}$ that requires ***maximum*** A-oracle circuit complexity.
[A-oracle circuit: along with AND/OR/NOT, one can have unbounded fan-in gates in the circuit that compute A]

Which Functions Have High Circuit Complexity?

Rephrase: What is the *smallest* complexity class containing a function of ***maximum*** $(2^n/n)$ circuit complexity?

Smallest known: $E^{\Sigma_2 P} = TIME^{\Sigma_2 P}[2^{O(n)}]$ [1970s]

Theorem: There is a function $f \in E^{\Sigma_2 P}$ that requires ***maximum*** circuit complexity.

Corollary: *If $P = NP$ then there is an $f \in E$ that requires **maximum** circuit complexity.*

Proof: $P = NP \Rightarrow E^{\Sigma_2 P} = E^{NP^{NP}} = E^{NP^P} = E^{NP} = E^P = E$

Thus, one could conclude that $P \neq NP$, if one could show every function in E has circuit complexity at most $H(n) - 1$

[where $H(n)$=maximum circuit complexity for n-bit functions]

Which Functions Have High Circuit Complexity?

Rephrase: What is the *smallest* complexity class containing a function of ***maximum*** $(2^n/n)$ circuit complexity?

Smallest known: $E^{\Sigma_2 P} = TIME^{\Sigma_2 P}[2^{O(n)}]$ [1970s]

Theorem: There is a function $f \in E^{\Sigma_2 P}$ that requires ***maximum*** circuit complexity.

If we could show E^{NP} needs $2^{\Omega(n)}$-size circuits (for example),
then by PRG constructions [NW94,IW97] we would have $BPP \subseteq P^{NP}$
[major open problem!]

There doesn't seem to be ***any*** real barrier to improving the $E^{\Sigma_2 P}$ to $\boldsymbol{\Sigma_2 E}$

[Recall: $\Sigma_2 E = NTIME^{NP}[2^{O(n)}]$, the exponential-time analogue of $\Sigma_2 P$]
(one consequence would be $BPP \subseteq \Sigma_2 P$, but we already know this is true!)

Which Functions Have High Circuit Complexity?

Rephrase: What is the *smallest* complexity class containing a function of ***maximum*** $(2^n/n)$ circuit complexity?

Smallest known: $E^{\Sigma_2 P} = TIME^{\Sigma_2 P}[2^{O(n)}]$ [1970s]

Rephrase Again: Does $\Sigma_2 E$ contain a function requiring $2^{\Omega(n)}$-size circuits?
[Recall: $\Sigma_2 E = NEXP^{NP}$, the exponential-time analogue of $\Sigma_2 P$]

Kannan [1981] $\Sigma_2 E$ requires $n^{polylog\, n}$ -size circuits (in fact "sub-half-exponential")

We don't even know if there is an oracle A under which $\Sigma_2 E$ has $2^{o(n)}$-size circuits!

That is, the following is open:

Is there an oracle A such that $\Sigma_2 E^A$ has an A-oracle circuit of $2^{o(n)}$ size?

(note: there ***is*** an oracle A such that $\Sigma_2 E^A$ requires $2^{\Omega(n)}$-size A-oracle circuits)

So it's possible there is a ***relativizing proof*** that $\Sigma_2 E$ requires $2^{\Omega(n)}$-size circuits, i.e., one that works by standard "black-box" techniques!

We will give a new way of thinking about this question!

The MISSING STRING Problem

MISSING STRING: Given a list of M strings of length N ($M < 2^N$), find a string not on the list

If the list is truth tables of "easy" functions, we are asking you to find a "hard" function!

MISSING STRING can be viewed as an "exponential-sized" version of the following problem, which has seen some attention recently [KKMP'21] [Korten'21] [RSW'22] [GLW'22]

RANGE AVOIDANCE: Given a Boolean circuit $C : \{0,1\}^n \to \{0,1\}^m$ where $m > n$,
find a string $y \in \{0,1\}^m$ that is not in the range of C

A random y works with probability $\geq 1/2$ …
but to check that a given y works, requires a call to SAT *(there is no input x s.t. $C(x) = y$)*

Deterministic algorithms for RANGE AVOIDANCE naturally can be used to construct explicit functions that do not have ``small'' circuits
(imagine that C takes descriptions of ``small'' circuits, and maps them to their truth tables)

The Missing String Problem

Missing String: Given a list of M strings of length N ($M < 2^N$), find a string not on the list

Theorem [Folklore? K'81] There is an $\tilde{O}(M)$-time algorithm for Missing String.

Proof: *"Halving Algorithm".* Our missing string will be $x = x_1 x_2 \cdots x_N$

Probe the 1st bit of each string.

Set x_1= "minority" bit of those M bits — M probes

Probe the 2nd bits of all $t_2 \leq M/2$ strings with first bit = x_1

Set x_2= "minority" bit of those t_2 bits — $\leq M/2$ probes

Probe the 3rd bits of all $t_3 \leq M/4$ strings with first two bits = $x_1 x_2$

Set x_3= "minority" bit of those t_3 bits — $\leq M/4$ probes

After $(\log_2 m) + 1$ rounds, no strings are left. Fill any remaining x_i's with zeroes.

Total number of probes is $O(M)$.

MISSING STRING and High-Complexity Functions

We show a strong ***equivalence*** between questions like:

Is there an oracle A such that $\Sigma_2 E^A$ has an A-oracle circuit of $2^{o(n)}$ size?

and the *complexity* of finding a **MISSING STRING**

"Efficient" circuits for Missing String correspond to "efficient" constructions of hard functions!

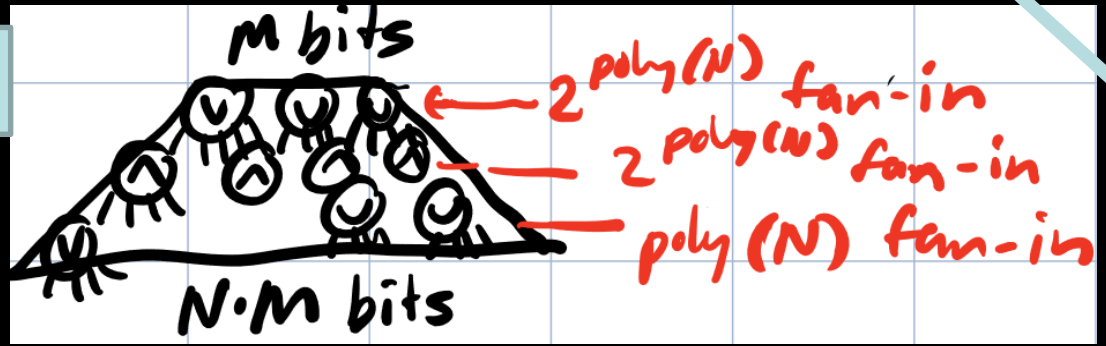
Example Theorem:

For every oracle A, $\boldsymbol{\Sigma_2 E^A}$ requires $2^{\varepsilon n}$-size A-oracle circuit complexity (a.e.)

$\Leftrightarrow$

There are *uniform* ***depth-3*** AC0 circuits for **MISSING STRING** of $2^{poly(N)}$ size and $poly(N)$ bottom fan-in, on all lists of length $M \approx 2^{N^\varepsilon}$

Depth 3 is crucial here!



Note: since $M \approx 2^{N^\varepsilon}$ we're asking for a circuit that has size *quasi-polynomial* in its input length!

MISSING STRING and High-Complexity Functions

We show a strong ***equivalence*** between questions like:

Is there an oracle A such that $\Sigma_2 E^A$ has an A-oracle circuit of $2^{o(n)}$ size?

and the *complexity* of finding a **MISSING STRING**

"Efficient" circuits for Missing String correspond to "efficient" constructions of hard functions!

Example Theorem:

For every oracle A, NE^A requires $2^{\varepsilon n}$-size A-oracle circuit complexity (a.e.)

Known to be FALSE [Wilson'84]

$\Leftrightarrow$

There are *uniform* ***depth-2*** AC0 circuits for **MISSING STRING** of $2^{poly(N)}$ size and $poly(N)$ bottom fan-in, on all lists of length $M \approx 2^{N^{\varepsilon}}$

Therefore, no depth-2 circuits!

Depth 3 is crucial here!

Note: there are depth-2 circuits for ***"succinct"*** instances $\Leftrightarrow$ NE requires exp size circuits

MISSING STRING and High-Complexity Functions

We show a strong ***equivalence*** between questions like:

Is there an oracle A such that $\Sigma_2 E^A$ has an A-oracle circuit of $2^{o(n)}$ size?

and the *complexity* of finding a **MISSING STRING**

"Efficient" circuits for Missing String correspond to "efficient" constructions of hard functions!

Example Theorem:

For every oracle A, $\boldsymbol{\Sigma_3 E^A}$ requires $2^{\varepsilon n}$-size A-oracle circuit complexity (a.e.)

Known to be TRUE [Kannan'81]

$\Leftrightarrow$

There are *uniform* ***depth-4*** AC0 circuits for **MISSING STRING** of $2^{poly(N)}$ size and $poly(N)$ bottom fan-in, on all lists of length $M \approx 2^{N^{\varepsilon}}$

Therefore, depth-4 circuits exist!

Depth 3 is crucial here!

MISSING STRING and High-Complexity Functions

Let $G : \{0,1\}^\star \to \{0,1\}$.

Say $\boldsymbol{f \in G \cdot E}$ if there is a constant $c > 0$ and Turing machine $M(x, y)$ running in 2^{cn} time when $n = |x|$ and $2^{cn} = |y|$, such that for all x,

$$f(x) = 1 \Leftrightarrow G\left(M\left(x, 0^{2^{2^{cn}}}\right), \ldots, M\left(x, 1^{2^{2^{cn}}}\right)\right) = 1$$

Example: $OR \cdot E = NTIME[2^{O(n)}]$

Generic Theorem [Informal]:

For every oracle A, $\boldsymbol{G \cdot E^A}$ requires $2^{\varepsilon n}$-size A-oracle circuit complexity (a.e.)

$\Leftrightarrow$

MISSING STRING has $poly(N)$-time uniform circuits of $2^{poly(N)}$ size on lists of length $M \approx 2^{N^\varepsilon}$, each circuit is a **"$G$ of Decision Trees of $poly(N)$ Depth"**

MISSING STRING and High-Complexity Functions

Theorem [Easier Case]: ***Also works for AND-OR-AND circuits***

There are *uniform* ***OR-AND-OR*** circuits for **MISSING STRING** of $2^{poly(N)}$ size and $poly(N)$ bottom fan-in, on all lists of length $M = 2^{O(N^{\varepsilon} \log N)}$

$\Rightarrow \Sigma_2 E$ requires $2^{\varepsilon n}$-size circuit complexity

This is the "algorithms imply lower bounds" direction!

Proof Sketch:

Our $\Sigma_2 E$ function will evaluate the depth-3 circuit for **MISSING STRING** with its input fixed to be:
the list of all 2^n-bit truth-tables of functions $f: \{0,1\}^n \to \{0,1\}$ with circuits of size $2^{\varepsilon n}$

$N = 2^n$, $M = 2^{O(2^{\varepsilon n} n)}$ [there are $2^{O(2^{\varepsilon n} n)}$ circuits of size $2^{\varepsilon n}$]

On input $x \in \{0,1\}^n$, our $\Sigma_2 E$ function evaluates the x-th output of the MISSING STRING circuit:

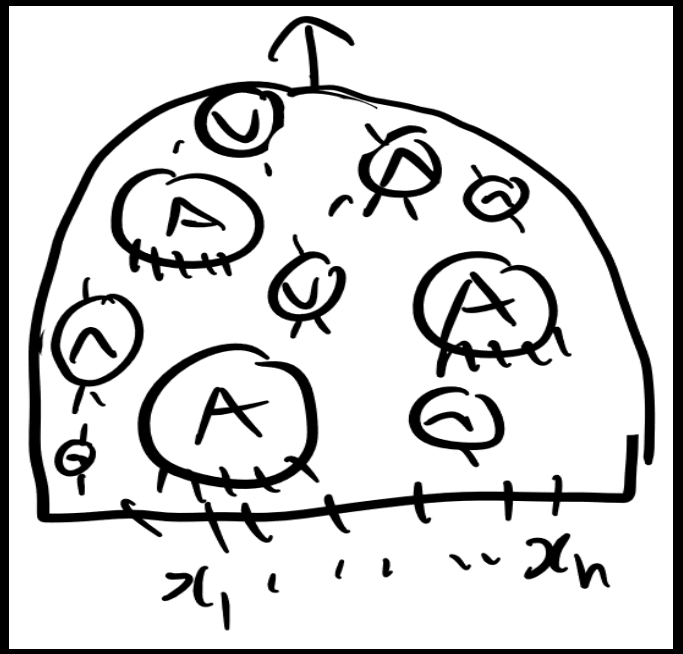
Existentially guess an input wire to the output OR gate which is true, coming from an AND gate g
Universally try all input wires to the AND gate g, coming from an OR gate h
Evaluate all $poly(N)$ literals with wires into h, **accept** iff at least one literal is true.

Both the existential and universal guesses take $poly(N)$ bits to write down (fan-in is $2^{poly(N)}$)

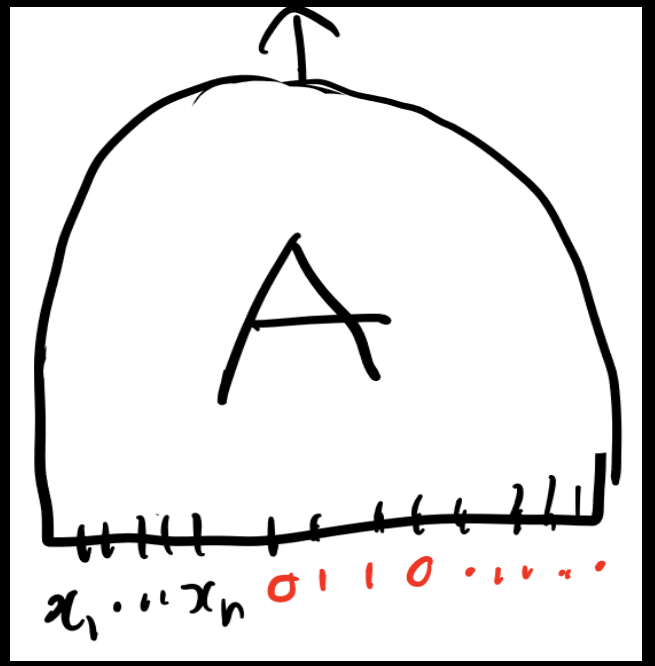
Assuming we can compute local information about the gates of the circuit in $poly(N)$ time, the above process can be implemented in $poly(N) = 2^{O(n)}$ time on a Σ_2 machine.

A Normal Form For Relativized Circuit Complexity

A-oracle circuit:



Say an A-oracle circuit is ***"trivial"*** if it has the form:



measure size by number of inputs

Theorem: Suppose ∃ oracle A s.t. $CTIME^A[2^{O(n)}]$ has A-oracle circuits of size $s(n)$

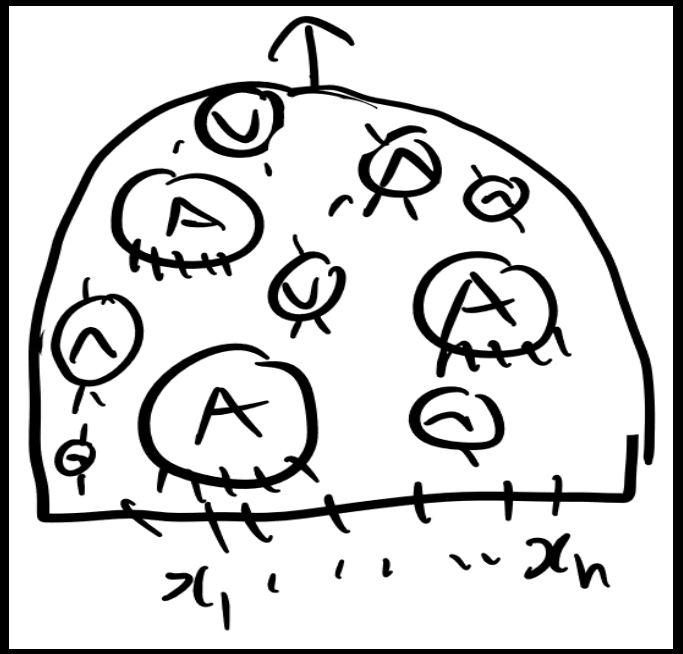
(where $CTIME$ could be $DTIME, NTIME, BPTIME, \Sigma_2 TIME$, …)

Then ∃ oracle B s.t. $CTIME^B[2^{O(n)}]$ has ***"trivial"*** B-oracle circuits of size $\tilde{O}(s(n))$

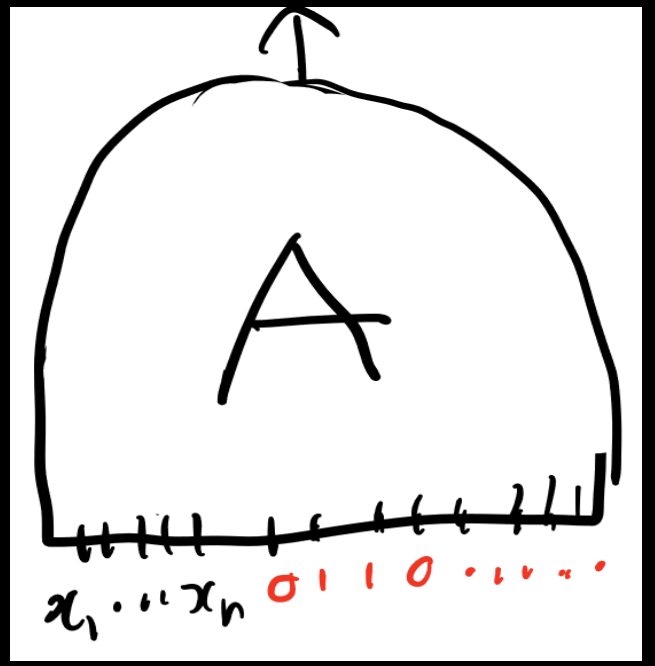
Thus WLOG, we can work with "trivial" circuits when considering such statements.

A Normal Form For Relativized Circuit Complexity

A-oracle circuit:



Say an A-oracle circuit is ***"trivial"*** if it has the form:



Theorem: Suppose ∃ oracle A s.t. $CTIME^A[2^{O(n)}]$ has A-oracle circuits of size $s(n)$

(where $CTIME$ could be $DTIME, NTIME, BPTIME, \Sigma_2 TIME$, …)

Then ∃ oracle B s.t. $CTIME^B[2^{O(n)}]$ has ***"trivial"*** B-oracle circuits of size $\tilde{O}(s(n))$

Proof Idea: Given an oracle A, set the oracle B to be "A-oracle Circuit Evaluation" (!)

As long as $CTIME^A[2^{O(n)}] = CTIME^B[2^{O(n)}]$, we are OK…

Local Algorithms for Finding a Missing String

MISSING STRING: Given a list of M strings of length N ($M < 2^N$), find a string not on the list

Cantor's Diagonal Argument shows:

If $M \leq N$, then we can find a missing string by taking the diagonal of the list

N

M

	1	2	3	…	k	…
x_1	1	0	1		0	
x_2	1	0	1		1	
⋮						
x_k	1	0	0		1	
⋮						

Local Algorithms for Finding a Missing String

MISSING STRING: Given a list of M strings of length N ($M < 2^N$), find a string not on the list

Cantor's Diagonal Argument shows:

If $M \leq N$, then we can find a missing string by taking the diagonal of the list

N

M

	1	2	3	…	k	…
x_1	0	0	1		0	
x_2	1	1	1		1	
⋮						
x_k	1	0	0		0	
⋮						

For all $i = 1, \ldots, N$, we can obtain the i-th bit of a missing string with ***only one bit probe*** into the input

[probe the i-th bit of the i-th string, output the opposite bit]

That is, when $M \leq N$, there's a "1-probe" algorithm for MISSING STRING

Under what conditions can we use only k probes?

Local Algorithms for Finding a Missing String

MISSING STRING: Given a list of M strings of length N ($M < 2^N$), find a string not on the list

If $M \leq N$, then we can find a missing string by taking the diagonal of the list

For all $i = 1, \ldots, N$, we can obtain the i-th bit of a missing string with ***only one bit probe***

That is, there is a "1-probe" algorithm for MISSING STRING

What can we do with k probes?

Theorem 1 (easy): For $M = kN$, there is **no** $k - 1$ probe algorithm for MISSING STRING.

Proof: Suppose for each $i = 1, \ldots, N$,
your algorithm makes only $k - 1$ probes and computes a missing string y.

The total number of different strings you probed, over all i, is $\leq (k-1)N < M$.

So there's some string on the list you haven't probed at all.

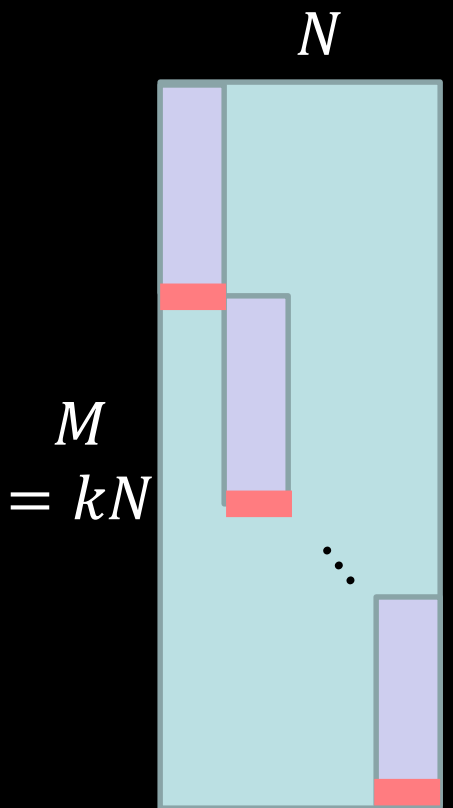
Your algorithm fails on any input that includes y among those non-probed strings.

Local Algorithms for Finding a Missing String

MISSING STRING: Given a list of M strings of length N ($M < 2^N$), find a string not on the list

Theorem 2: For $M \leq kN$, there is an $O(k \log k)$-probe algorithm for MISSING STRING.

Idea: Combine the diagonal argument and the algorithm that makes $O(M)$ probes!



Divide the N bits of each string into $\sim N/b$ blocks of length b

Divide the M strings into $\sim M/t$ blocks of length t, where $t \leq 2^b - 1$.

If $\frac{M}{t} \leq \frac{N}{b}$, then if we find a missing string for each purple block,

their concatenation is a missing string for the entire set!

Have: $M = kN$, Want: $M \leq (2^b - 1)\, N/b$. We therefore set:

$$b = \log(k) + \operatorname{loglog}(k) + O(1)$$

Then, to get any particular bit of the missing string, number of probes is

$$O(t) \leq O\left(2^b\right) \leq O(k \log k)$$

What Good are Local Algorithms?

Theorem 2: For $M \leq kN$, there is an $O(k \log k)$-probe algorithm for MISSING STRING.

Corollary: New Time Hierarchy Theorems, against Non-Uniform Programs!

Time Hierarchy Theorem: For "reasonable" g, h where $h(n) >> g(n)$,

$\text{TIME}(h(n)) \not\subseteq \text{TIME}(g(n))$

The time hierarchy can be generalized to work for "small non-uniform advice"

Define $f \in \text{TIME}[g(n)]/a(n)$ if for every n, there is *some program* of length $a(n)$, running in time $g(n)$, that decides f.

Time Hierarchy Against Advice [Folklore]: For "reasonable" g, h where $h(n) >> g(n)$,

$\text{TIME}(h(n)) \not\subseteq \text{TIME}(g(n))/n$

Idea: The hard function running in $O(h(n))$ time treats its input of length n ***as a program,*** and simulates the program on its own code, outputting the opposite answer.

contains uncomputable stuff!

What Good are Local Algorithms?

Theorem 2: For $M \leq kN$, there is an $O(k \log k)$-probe algorithm for MISSING STRING.

Corollary: New Time Hierarchy Theorems, against Non-Uniform Programs!

Time Hierarchy Theorem: For "reasonable" g, h where $h(n) >> g(n)$,

$$\textbf{TIME}(h(n)) \not\subseteq \textbf{TIME}(g(n))$$

The time hierarchy can be generalized to work for "small non-uniform advice"

Define $f \in \textbf{TIME}[g(n)]/a(n)$ if for every n, there is *some program* of length $a(n)$, running in time $g(n)$, that decides f.

Time Hierarchy Against Advice, Version 2 [Folklore]: For "reasonable" g, h

$$\textbf{TIME}(2^{n+g(n)} \cdot h(n)) \not\subseteq \textbf{TIME}(h(n))/g(n)$$

Idea: The hard function enumerates all $2^{g(n)}$ programs of length $g(n)$,
and simulates each of them on every possible n-bit input.
Then it computes a missing string from the list of $M = 2^{g(n)}$ strings of length $N = 2^n$

What Good are Local Algorithms?

Theorem 2: For $M \leq kN$, there is an $O(k \log k)$-probe algorithm for MISSING STRING.

Corollary: New Time Hierarchy Theorems, against Non-Uniform Programs!

Old Time Hierarchies Against Advice: **For "reasonable" g, h where $h(n) >> g(n)$,**

$$\text{TIME}(h(n)) \not\subseteq \text{TIME}(g(n))/(n) \qquad \text{TIME}(2^{n+g(n)} \cdot h(n)) \not\subseteq \text{TIME}(h(n))/g(n)$$

Examples of New Hierarchies (using Theorem 2)

$$\text{TIME}(5^n) \not\subseteq \text{TIME}(2^n)/(2n)$$

program is LARGER than n (!)

$$\text{TIME}(n^{2c+1}) \not\subseteq \text{TIME}(n^c)/(n + c\log(n))$$

$$\text{TIME}(2^{cn} \cdot poly(n)) \not\subseteq \text{TIME}(O(n))/(cn + n)$$

Note: All the above hierarchies relativize,
and there is an oracle A such that $\text{TIME}^A[2^{cn}] \subseteq \text{TIME}^A[O(n)]/(cn + n)$!!

(Main) Open Problems

- Does MISSING STRING have small uniform depth-3 circuits, or not?

(it cannot have depth-2 circuits, and it does have depth-4 circuits)

- How many probes (as a function of k) are necessary and sufficient to find a missing string, when $M \leq k\ N$?
  The answer is somewhere between k and $O(k \log k)$

- Finding an average-case hard function corresponds to finding a REMOTE POINT (string "far" in Hamming dist from all on the list).
  Can we rule out depth-3 circuits for this harder problem?