Sparse Sets and Turing Reductions

Adam Scrivener, Haofu Liao, Nabil Hossain, Shupeng Gui, Thomas Lindstrom-Vautrin

Department of Computer Science
University of Rochester

November 9, 2015
**Introduction**

- We’ve seen: if a sparse set is NP-hard or NP-complete w.r.t many-one reductions, then $P = NP$.
- Today we investigate whether any sparse set can be NP-hard or NP-complete w.r.t. Turing reductions, a weaker assumption (compared to many-one reductions).
- **Open Question**: do these Turing reduction based hypotheses imply that $P = NP$?
- We can show that the Polynomial Hierarchy **collapses**, given these assumptions hold.
- Intuitively, PH **collapses** when all polynomial classes above a certain order are shown to be equal.
The Polynomial Hierarchy

\[\Sigma_i^p = \Pi_i^p = \Delta_i^p = \Pi_i^{p-1} \]

E.g., \( \Sigma_1^p = \Pi_1^p = \Delta_1^p = \Pi_0^p = \mathsf{NP} \)

\( \Pi_i^p = \defn \text{co} \cdot \Sigma_i^{p-1} \quad i \geq 0 \)

Examples:
- \( \Sigma_2^p = \mathsf{NP} \cdot \mathsf{NP} \)
- \( \Pi_1^p = \{ \exists \mathbf{z} \in \Sigma_2^p \} = \text{co-NP} \)

\( \text{So not a Venn-Diagram! Almost Hasse...} \)

\( \exists_p \forall_p \exists_p \neq_3 \Sigma_3^p \)

\( \mathsf{NP} \cap \text{co-NP} \)

\( \Pi_2^p \quad \forall_p \exists_p \)

\( \Pi_3^p \quad \forall_p \exists_p \forall_p \)

NP-complete

\( \exists_p \neq_1 \Sigma_1^p \)

\( \mathsf{NP} \cap \text{co-NP} \)

\( \Pi_1^p \quad \forall_p \)

Nondem note: \( \mathsf{co-NP} \neq (\text{co-NP}) \)

\( \Pi_2^p \) is often expressed as \( \mathsf{NP} \cap \text{co-NP} \)
Polynomial Hierarchy Collapse

\[ PH = \bigcup_i \Sigma_i^P \]

\[ = P \cup NP \cup NP^{NP} \cup NP^{NP^{NP}} \cup NP^{NP^{NP^{NP}}} \cup \ldots \]

Theorem (Gold) ¹

If \( \Sigma_i^p = \Pi_i^p \), then \( PH = \Sigma_i^p \)

Example

if \( NP = coNP \), then \( PH = NP \)

- An implication: any problem, that can be solved using an NP machine with access to an NP oracle, can also be solved with a non-det poly-time TM with no access to oracle

Can sparse sets be NP-complete or NP-hard w.r.t Turing reductions? What are the implications?

**Theorem 1.14 [Hemaspaandra-Ogihara]**

If NP has sparse Turing-complete sets, then the Polynomial Hierarchy collapses to $P^{NP}[\log n]$.

**Theorem 1.15 [Hem-Ogi] (also called Karp-Lipton Thm)**

If NP has sparse Turing-hard sets, then the Polynomial Hierarchy collapses to $NP^{NP}$.
**Bounded/restricted query classes**

- **Motivation:** add new classes to PH to capture problems that can be solved by restricting the number of queries made to oracle to $O(\log n)$ (instead of polynomial)

**E.g. Odd colorability in graphs**

- it is in $P^{NP}$ and is NP-hard, but not known whether it is in NP
- but it can be solved in $P^{NP}[\log n]$ by using a $P^{NP}$ machine making only $O(\log n)$ (rather than polynomially many) queries to an NP oracle.

---

Theorem 1.14

If \( NP \) has sparse \( NP \leq_{T}^{P} \)-complete sets, then

\[ PH = P^{NP}[\log n] \]

Proof Strategy

- proof uses “census” approach
- For a sparse set, the census approach is to first obtain the exact number of elements in the set up to some given length, and then exploit that information.
Proof of Theorem 1.14

1) Let $S$ be a sparse set s.t. $S$ is \( \leq^p_T \) -complete for NP
2) For any \( \ell \), let \( p_\ell(n) = n^\ell + \ell \)
3) Let \( j \) be s.t. \( (\forall n)[|S^{\leq n}| \leq p_j(n)] \)
4) Let \( M \) be deterministic poly-time TM s.t. SAT = \( L(M^S) \)
   - \( M \) exists since \( S \) is Turing-hard for NP
5) Let \( k \) be s.t. \( p_k(n) \) bounds runtime of \( M \) regardless of \( M \)'s oracle
6) Let \( L \) be an arbitrary set in \( \Sigma^p_2 \) (i.e. \( NP^{NP} \))
   - Since SAT is NP-complete, we have \( \Sigma^p_2 = NP^{SAT} \)
7) \( \exists \) non-det poly-time TM \( N \) s.t. \( L = L(N^{SAT}) \)
8) Let \( \ell \) be s.t. \( p_\ell(n) \) bounds non-det runtime of \( N \) for all oracles
9) Note that \( L = L(N^{L(M^S)}) \)
Proof of Theorem 1.14 (Continued)

\[ V = \{0\#1^n\#1^q \mid |S| \geq q\} \]
\[ \cup \]
\[ \{1\#x\#1^n\#1^q \mid (\exists Z \subseteq S) [|Z| = q; \land x \in L(N^L(M^Z))]\} \]

- \( V \in NP \) since \( S \in NP \)
- 1st set (census): will be exploited to compute \( |S| \)
- 2nd set: will be used to check whether \( x \in L \)
The following $P^{\text{NP}[\log n]}$ algorithm accepts $L$ by making $O(\log n)$ calls to the NP oracle $V$, for each input string $y$.

### Step 1

- In $O(\log |y|)$ sequential queries to $V$, compute $|S^{\leq p_k(p_\ell(|y|))}|$
  - queries of the form $0^\# p_k(p_\ell(|y|)) \# 1^z$
  - vary $z$ as in binary search until we find $|S^{\leq p_k(p_\ell(|y|))}|$
- Since $|S^{\leq p_k(p_\ell(|y|))}|$ is bounded by $p_j(p_k(p_\ell(|y|)))$, thus $O(\log |y|)$ queries are sufficient to find census value
- Let the census value obtained be $r$
Proof of Theorem 1.14 (Continued)

Step 2

Ask $V$ the query $1\#y\#1^{p_k(p_\ell(|y|))}\#1^r$, and accept if and only if this query $\in V$

- Clearly this is a $P^{NP}[\log n]$ algorithm
- Algorithm accepts $L$
- Since $L \in \Sigma_2^P (= NP^{NP})$, we have $\Sigma_2^P = P^{NP}[\log n]$.
- Since $P^{NP}[\log n]$ is closed under complementation, we have $\Sigma_2^P = \Pi_2^P$
- Therefore by Theorem (Gold), $PH = \Sigma_2^P = P^{NP}[\log n]$
THE KARP-LIPTON THEOREM

If there is a sparse NP-Turing-Hard set \((\exists S \in \text{SPARSE} \forall T \in \text{NP} \exists^* S)\),

the polynomial hierarchy collapses to \(\text{NP}^{\text{NP}}\) \((\text{PH} = \text{NP}^{\text{NP}})\)

(also congratulations if you did the reading, it is almost impossible)
A LITTLE REVIEW

What the heck does this mean?

SPARSE SETS:

$p(n) = n^2 + 1$

alphabet size $|\Sigma| = 3$

It's easy to see why they're called sparse.
As a proportion of possible elements, sparse sets contain almost nothing.

$(|\Sigma^*|^n = |\Sigma|^n = 3^n)$
A LITTLE REVIEW

A sparse $\text{NP}$-Turing-Hard set:
We have a sparse set $S$

![Diagram showing relationship between NP, M, and S]

every set in NP can be solved by a machine in polynomial time with an oracle for $S$.

(not necessarily the same machine $M$)
A LITTLE REVIEW

The Polynomial Hierarchy (PH):

\[ \text{PH} = \text{P} \cup \text{NP} \cup \text{NP}^2 \cup \text{NP}^3 \cup \cdots \]

Note: \( \text{NP}^0 = \text{NP} \)

and

\[ P \subseteq \text{NP} \subseteq \text{NP}^2 \subseteq \text{NP}^3 \subseteq \text{NP}^4 \subseteq \cdots \]
A LITTLE REVIEW

Polynomial time hierarchy collapse:

essentially when all subset-equals signs become equals signs above a certain order.

OR in other words

All polynomial classes above a certain order are shown to be equal

\[ P \subseteq \mathsf{NP} \subseteq \mathsf{NP}^{\mathsf{NP}} \subseteq \mathsf{NP}^{\mathsf{NP}^{\mathsf{NP}}} \subseteq \mathsf{NP}^{\mathsf{NP}^{\mathsf{NP}^{\mathsf{NP}}}} \subseteq \ldots \]

(a collapse of \( \mathsf{PH} \) to \( \mathsf{NP}^{\mathsf{NP}} \))

\[ \mathsf{PH} = \mathsf{P} \cup \mathsf{NP} \cup \mathsf{NP}^{\mathsf{NP}} \cup \mathsf{NP}^{\mathsf{NP}^{\mathsf{NP}}} \cup \mathsf{NP}^{\mathsf{NP}^{\mathsf{NP}^{\mathsf{NP}}}} \subseteq \ldots \]
LINES OF INQUIRY

How do we show a polynomial hierarchy collapse to $\text{NP}^{\text{NP}}$?

Notice if we show $\text{NP}^{\text{NP}} = \text{NP}^{\text{NP}^{\text{NP}}}$, all other inductively higher order polynomial classes will collapse recursively:

$$\ldots \leq \text{NP}^{\text{NP}} = \text{NP}^{\text{NP}^{\text{NP}}} ? \text{NP}^{\text{NP}^{\text{NP}}} ? \text{NP}^{\text{NP}^{\text{NP}}} ? \ldots$$

So $\text{NP}^{\text{NP}} = \text{NP}^{\text{NP}^{\text{NP}}}$ is all we need to show.
AVAILABLE TOOLS

Remember we need to use our sparse set $S$ and the fact that any language in NP Turing reduces to $S$.

Since SAT is in NP,

$$SAT \leq^p_T S$$

So we have a machine $M$ such that

$$SAT = L(M^S)$$

IMPORTANT: $M$ is a deterministic Turing machine that runs in polynomial time.

In other words, this machine $M$ given an oracle for $S$ is a machine for SAT.
LINES OF INQUIRY

We want to show that an arbitrary element of $\text{NP}^{\text{NP}^{\text{NP}^{\text{NP}}}}$ can be solved by an element of $\text{NP}^{\text{NP}}$.

First we must see how to represent these:

$L \leq \text{NP}^{\text{NP}^{\text{NP}^{\text{NP}}}}$

$L = L(N_1, L(N_2^{L(N_3)})$ for machines $N_1, N_2, N_3$

(nondeterministic poly time)
LINES OF INQUIRY

What the heck does this mean?

$N_1$

$L = L(N_1^{L(N_2^{L(N_3)})})$

$N_2$

$N_3$

Note: since SAT is NP complete,

we can make $N_2$ do the polynomial work of $f$ and replace $N_1$ with a machine for SAT
POLYNOMIAL BOUNDS

Here are the polynomial bounds we will use and keep track of:

First, define $P_k(n)$ for natural numbers $k$ and $n$

$$P_k(n) = n^k + k$$

$\mathbf{j}$: $\forall n \left[ \|S^k\| \leq P_j(n) \right]$ (we can polynomially bound any prefix of our sparse set)

$\mathbf{k}$: runtime($M^A(n)$) $\leq P_k(1^w)$ For all oracles $A$

(M is our deterministic polynomial TM such that $L(M^A) = \text{SAT}$)

$\mathbf{l}$: max(runtime($N_1^A(n)$), runtime($N_2^A(n)$)) $\leq P_l(1^w)$ For all oracles $A$

($N_1$ and $N_2$ are nondeterministic polynomial TM, we will be investigating in our proof)
**Pause to Ponder 1.15**  
*Show why this "without loss of generality claim" holds.*

(Answer sketch for Pause to Ponder 1.15: Given a machine $M$, let the machines $M_1, M_2, \ldots$, be as follows. $M_i^A(x)$ will simulate the action of exactly $p_i(|x|)$ steps of the action of $M^A(x)$, and then will halt in an accepting state if $M^A(x)$ halted and accepted within $p_i(|x|)$ steps, and otherwise will reject. Note that since the overhead involved in simulating one step of machine is at most polynomial, for each $i$, there will exist an $\hat{i}$ such that for every $A$ it holds that $M_i^A$ runs in time at most $p_i(n)$. Furthermore, in each relativized world $A$ in which $M^A$ runs in time at most $p_i$, it will hold that $L(M^A) = L(M_i^A)$. Relatedly, in our proof, given the machine $M$ such that SAT = $L(M^S)$, we will in light of whatever polynomial-time bound $M^S$ obeys similarly replace $M$ with an appropriate $M_j$ from the list of machines just described.)
HOW DO POLYNOMIAL BOUNDS WORK?

Consider $M^A(x)$. Its runtime is bounded by $P_k(n^A)$.

At each step $M^A$ can print at most one character/letter.
So the largest query $M$ can make to $A$ is one in which it uses all its steps to form the query and submits it at the last step.

So query size is bounded by $P_k(n^A)$. 
MAIN INSIGHTS

Can we simulate $N_2^{\text{SAT}}$ which works in $\text{NP}$ with a machine that works in $\text{NP}$?

$N_2^{\text{SAT}} \leq_l \text{NP}$ works in $\text{NP}$ since $\text{SAT} \leq_l \text{NP}$ is finite and in $\text{P}$, and $\text{NP} = \text{NP}$.

($x$ is the input to SAT, not to $N_2$. It is the query.)

(Since we cannot actually easily find $\text{SAT} \leq_l n$

this is assuming we can)
MAIN INSIGHTS

Simulating SAT:
Suppose you know the length of the input to M, $|x|$. M can make a query of length at most $\mathbb{P}_n(|x|)$. On inputs of these lengths,
$M \leq^\text{P}_{\mathbb{P}_n(|x|)} \text{ simulates SAT}$

Takeaway:
A finite prefix of $S$ is enough to simulate SAT given the input length.
Since $M$ is deterministic and polynomial and $S \leq \text{P}^{\text{NP}}$, being finite is in $\text{P}$. 

$M^{S \leq \text{P}^{\text{NP}}(x)} \in \text{P}^{\text{P}} = \text{P}$
Now that we have the main insights, I will walk you through the meat of the formal proof.
Define

\[ V_0 = \{ \#1^n\#S' | (\exists z \in (\Sigma^*)^{\leq n})((a) \text{ } z \text{ is not a well-formed formula and } M^{S'}(z) \text{ accepts; or (b) } z \text{ is a well-formed formula free variables and either (b1) } M^{S'}(z) \text{ accepts and } z \notin \text{ SAT or (b2) } M^{S'}(z) \text{ rejects and } z \in \text{ SAT; or (c) } z \text{ is a well-formed formula variables } z_1, z_2, \ldots \text{ and it is not the case that: } M^{S'}(z) \text{ accepts if and only if} \]

\[ (M^{S'}(z[z_1 = \text{True}]) \text{ accepts } \lor M^{S'}(z[z_1 = \text{False}]) \text{ accepts}) \} , \]

where, as defined earlier in this chapter, \( z[\ldots] \) denotes \( z \) with the indicated variables assigned as noted.

\[ V_1 = \{ \#1^n\#S'\#z | z \in L(N_2^L(M^{S'})) \}. \]

\[ V = V_0 \cup V_1. \]
Step 1  Nondeterministically guess a set $S' \subseteq (\Sigma^*)^{\leq p_k(p_{\ell}(|y|))}$ satisfying $||S'|| \leq p_j(p_k(p_{\ell}(|y|))))$. If $0\#^1p_k(p_{\ell}(|y|)))\#S' \in V$ then reject. Otherwise, go to Step 2.

Step 2  Simulate the action of $N_1(y)$ except that, each time $N_1(y)$ makes a query $z$ to its $L(N^2_{SAT})$ oracle, ask instead the query $1\#S'\#z$ to $V$. 