Sparse NP-Turing-Hard/Complete Sets

Alan Beadle  Komail Dharsee  David Fink
Gene Kim  Joel Kottas
Xiangru Lian  Zahra Razavi

November 9, 2016
Sparse set

- We define the prefix set: for a set $A$, $A^{\leq n} = \{ x \in A \mid |x| \leq n \}$.
- We define the census function: for an set $A$, $c_A(n) = \| \{ x \in A \mid |x| \leq n \} \| = \| A^{\leq n} \|$.
- A language $A$ is sparse if there is a polynomial $p$ such that $c_A \leq p$.
A *Polynomial-time Turing reduction* ($\leq^p_T$) is a reduction computable by an oracle Turing machine that halts for all inputs within a polynomial bound on the number of steps with respect to the input.
Relevant Kleene Hierarchy Categories

\[ PH : \text{Polynomial (Kleene) Hierarchy} \]
\[ \Sigma^p_0 = P \]
\[ \Sigma^p_1 = NP \]
\[ \Sigma^p_2 = NP^{NP} \]

Let \[ \Theta^p_2 = P^{NP[\mathcal{O}(\log(n))]} \] be the class of languages that can be computed with \[ \mathcal{O}(\log(n)) \] calls to a \[ NP \] oracle. (This will be relevant!)
Theorem 1.14 (Kadin) *If there exists an NP-complete sparse set w.r.t. polynomial time Turing reductions, then $PH = \Theta^p_2$*
Theorem 1.14 (Kadin) If there exists an \( NP \)-complete sparse set w.r.t. polynomial time Turing reductions, then \( PH = \Theta^p_2 \)

Proof

1. Let \( S \) be an \( NP \)-complete sparse set w.r.t. \( \leq^p_T \).
Theorem 1.14 (Kadin) If there exists an \( NP \)-complete sparse set w.r.t. polynomial time Turing reductions, then \( PH = \Theta_2^p \)

Proof

1. Let \( S \) be an \( NP \)-complete sparse set w.r.t. \( \leq_T^p \).
2. There exists a \( j \) that \( (\forall n)(c_S(n) \leq p_S(n) := n^j + j) \)
Theorem 1.14 (Kadin) If there exists an $NP$-complete sparse set w.r.t. polynomial time Turing reductions, then $PH = \Theta^p_2$

Proof

1. Let $S$ be an $NP$-complete sparse set w.r.t. $\leq^p_T$.
2. There exists a $j$ that $(\forall n)(c_S(n) \leq p_S(n) := n^j + j)$
3. Let $M$ be a deterministic polynomial-time TM that solves $SAT$ using the oracle $S$. (i.e. $SAT = L(M^S)$). – Since $S$ is $\leq^p_T - complete$ and thus $\leq^p_T - hard$ for $NP$, $M$ exists.
Theorem 1.14 (Kadin) If there exists an $NP$-complete sparse set w.r.t. polynomial time Turing reductions, then $PH = \Theta^p_2$

Proof

1. Let $S$ be an $NP$-complete sparse set w.r.t. $\leq^p_T$.
2. There exists a $j$ that $(\forall n)(c_S(n) \leq p_S(n) := n^j + j)$
3. Let $M$ be a deterministic polynomial-time TM that solves $SAT$ using the oracle $S$. (i.e. $SAT = L(M^S)$). – Since $S$ is $\leq^p_T$-complete and thus $\leq^p_T$-hard for $NP$, $M$ exists.
4. There exists a $k$ such that $p_M(n) = n^k + k$ bounds the runtime of $M^O$ for any oracle $O$. 

Proof (cont.)

5. Consider $L^* \in NP^{NP}$. Note, $NP^{NP} = NP^{SAT}$.
Proof (cont.)

5. Consider \( L^* \in NP^{NP} \). Note, \( NP^{NP} = NP^{SAT} \).

6. Consider \( N \), a non-deterministic polynomial-time TM where \( L^* = L(N^{SAT}) = L(N^{L(M^S)}) \).
Proof (cont.)

5. Consider $L^* \in NP^{NP}$. Note, $NP^{NP} = NP^{SAT}$.

6. Consider $N$, a non-deterministic polynomial-time TM
   where $L^* = L(N^{SAT}) = L(N^{L(M^S)})$.

7. $p_N(n) := n^l + l$ bounds the runtime of $N^O$ for any oracle $O$. 
Proof (cont.)

Define:

\[ V_1 = \{0#1^n#1^q \mid ||S^{\leq n}|| \geq q\} \]

\[ V_2 = \{1#x#1^n#1^q \mid (\exists Z \subseteq S^{\leq n})[||Z|| = q \land x \in L(N^L(M^Z))]\} \]

\[ V = V_1 \cup V_2 \]

\[ V \in NP. \]
Proof (cont.)

\[ V_1 = \{0\#1^n\#1^q \mid \|S\leq n\| \geq q \} \]

\[ V_1 \in NP \] since we can solve it with the following NDTM:
On input \(0\#1^n\#1^q\) non-deterministically generate all strings of length at most \(n\). For each branch, simulate NDTM, \(N_1\), where \(L(N_1) = S\), to determine if the generated string is in \(S\). If \(N_1\) accepts, non-deterministically generate all strings of length at most \(n\) except those already generated in the current path and repeat above. Repeat this up to \(q\) times. If there is an accepting path, then we say \(0\#1^n\#1^q \in V_1\).
Proof (cont.)

\[ V_2 = \{1\#x\#1^n\#1^q \mid (\exists Z \subseteq S^{\leq n})[||Z|| = q \land x \in L(N^{L(M^Z)})]\} \]

\( V_2 \in NP \) since we can solve it with the following NDTM:
On input 0\#x\#1^n\#1^q non-deterministically generate all
\( Z \subseteq S^{\leq n} \) where \( ||Z|| = q \). Each branch simulates \( N^{L(M^Z)} \) on \( x \).
Since \( Z \) is finite, \( M^Z \in P \). Thus \( N^{L(M^Z)} \in NP \). If any branch
accepts, we accept the input.
Proof (cont.)

\[ V_2 = \{ 1 \# x \# 1^n \# 1^q \mid (\exists Z \subseteq S^{\leq n})[||Z|| = q \land x \in L(N^{L(M^Z)})] \} \]

\( V_2 \in NP \) since we can solve it with the following NDTM: On input 0\#x\#1^n\#1^q non-deterministically generate all \( Z \subseteq S^{\leq n} \) where \( ||Z|| = q \). Each branch simulates \( N^{L(M^Z)} \) on \( x \). Since \( Z \) is finite, \( M^Z \in P \). Thus \( N^{L(M^Z)} \in NP \). If any branch accepts, we accept the input.

\( V_1 \in NP \) and \( V_2 \in NP \).
So \( V = V_1 \cup V_2 \in NP \).
Proof (cont.)

Reminder:
\[ V_1 = \{0\#1^n\#1^q \mid \|S^{\leq n}\| \geq q\} \]
\[ V_2 = \{1\#x\#1^n\#1^q \mid (\exists Z \subseteq S^{\leq n})[\|Z\| = q \land x \in L(N^{L(M^Z)})]\} \]

Now we’ll define an algorithm that accepts \( L^* \) with \( O(\log|y|) \) calls to oracle \( V \).

Intuitive algorithm summary:
1. Binary search the census size with queries to the \( V_1 \) portion of \( V \) (Notice that strings in \( V_1 \) and \( V_2 \) are structurally different!).
2. If we know the census size, we can query the \( V_2 \) portion with \( y \) while forcing matching set to be \( L^* \).
Proof (cont.)

Reminder:
\[ V_1 = \{0\#1^n\#1^q \mid ||S^{\leq n}|| \geq q\} \]
\[ V_2 = \{1\#x\#1^n\#1^q \mid (\exists Z \subseteq S^{\leq n})[||Z|| = q \land x \in L(N^{L(M^Z)})]\} \]

Exact algorithm:

1. Binary search \( ||S^{\leq p_M(p_N(|y|))}|| \) by querying \( V \) (really just \( V_1 \)), with \( 0\#1^{p_M(p_N(|y|))}\#1^z \), varying \( z \). Let us simply call the final computed census value \( r \) for brevity.

2. Query \( V \) (really just \( V_2 \)) with \( 0\#y\#1^{p_M(p_N(|y|))}\#1^r \) and accept if and only if \( V \) accepts the query.
Proof (cont.)

Reminder:

\[ V_1 = \{0\#1^n\#1^q \mid \|S^{\leq n}\| \geq q\} \]

Why does it work?:

1. Binary search \(\|S^{\leq p_M(p_N(|y|))}\|\) by querying \(V\) (really just \(V_1\)), with \(0\#1^{p_M(p_N(|y|))}\#1^z\), varying \(z\). Let us simply call the final computed census value \(r\) for brevity.

   - \(\|S^{\leq p_M(p_N(|y|))}\|\) is bounded by \(p_S(p_M(p_N(|y|)))\).
   - \(p_S(p_M(p_N(|y|)))\) is polynomial in \(|y|\), so \(O(\log|y|)\) queries is sufficient for binary search of \(\|S^{\leq p_M(p_N(|y|))}\|\).
Proof (cont.)

Reminder:

\[ V_2 = \{1\#x\#1^n\#1^q \mid (\exists Z \subseteq S^{\leq n})[||Z|| = q \land x \in L(N^{L(M_Z)})]\} \]

Why does it work?:

2. Query \( V \) (really just \( V_2 \)) with \( 0\#y\#1^{p_M(p_N(|y|))}\#1^r \) and accept if and only if \( V \) accepts the query.
   - \( r = ||S^{p_M(p_N(|y|))}|| \) so \( Z = S^{\leq p_M(p_N(|y|))} \).
   - Query is checking \( y \in L(N^{L(M^{S^{\leq p_M(p_N(|y|))}})}) \).
   - \( N \) bounded by \( p_N(|y|) \) steps.
   - The input of \( M \) is bounded by \( p_N(|y|) \) takes at most \( p_M(p_N(|y|)) \) steps.
   - \( M \) will never query oracle of \( S^{\leq p_M(p_N(|y|))} \) with anything larger than \( p_M(p_N(|y|)) \), so for our purposes the oracle for \( S^{\leq p_M(p_N(|y|))} \) is equivalent to the oracle for \( S \).
Proof (cont.)

Reminder:

\[ V_2 = \{1\#x\#1^n\#1^q \mid (\exists Z \subseteq S^{\leq n})[||Z|| = q \land x \in L(N^L(M^Z))]\} \]

Why does it work?:

2. Query \( V \) (really just \( V_2 \)) with \( 0\#y\#1^{pM(pN(|y|))}\#1^r \) and accept if and only if \( V \) accepts the query.

- For any of our queries,
  \[ L(N^L(M^{S_{\leq pM(pN(|y|))}})) = L(N^L(M^S)) = L^* \]
- Thus \( x \in L(N^L(M^Z)) \) solves \( x \in L(N^L(M^S)) \) and thus \( x \in L^* \)
Now that we see why the algorithm works...

- Since this algorithm uses $O(\log |y|)$ calls to an $NP$ oracle to solve $L^*$, $L^* \in \Theta_2^p$.
- $L$ is an arbitrary $\Sigma_2^p$ set, so $\Sigma_2^p = \Theta_2^p$.
- $\Theta_2^p$ is closed under complementation so $\Sigma_2^p = \Pi_2^p$.
- Thus $PH$ collapses on $\Theta_2^p$. $PH = \Theta_2^p$.

This concludes the proof for:

**Theorem 1.14 (Kadin)** If there exists an $NP$-complete sparse set w.r.t. Turing reductions, then $PH = \Theta_2^p$.
Theorem 1.16 (Karp-Lipton) If $NP$ has sparse $\leq^p_T$-hard sets then $PH = NP^{NP}$.
Theorem 1.16 (Karp-Lipton) If $NP$ has sparse $\leq^p_T$-hard sets then $PH = NP^{NP}$.

Let $L$ be an arbitrary set in $\Sigma^p_3$. We will give a $\Sigma^p_2$ algorithm for $L$. This establishes that $\Sigma^p_2 = \Sigma^p_3$, which implies that $PH = \Sigma^p_2$, thus proving the theorem.
Theorem 1.16 (Karp-Lipton) If \( NP \) has sparse \( \leq^p_T \)-hard sets then \( PH = NP^{NP} \).

Let \( L \) be an arbitrary set in \( \Sigma^p_3 \). We will give a \( \Sigma^p_2 \) algorithm for \( L \). This establishes that \( \Sigma^p_2 = \Sigma^p_3 \), which implies that \( PH = \Sigma^p_2 \), thus proving the theorem.

The idea: For each input \( y \) if we can find a set \( U \) where its finite prefix \( U \leq^n \) behaves as a \( SAT \leq^n \) oracle in (nondeterministic) polynomial time and \( n \) bounds the possible input length, we can safely replace \( SAT \) with \( SAT \leq^n \) and \( SAT \leq^n \) with \( U \leq^n \). Then \( NP^{NP^{SAT}} \) can be computed with \( NP^{NP^{U \leq^n}} = NP^{NP} \).
Let $S$ be a sparse set that is $\leq^p_T$-hard for NP.
Let $S$ be a sparse set that is $\leq^p_T$-hard for NP.

Let $j$ be such that $(\forall n)[\|S - \leq^n\| \leq p_S(n) := n^j + j]$. 

Note that $\Sigma^p_3 = \text{NP}$. 

Let $k$ be such that $p_M(n) := n^k$ bounds the runtime of $M$ for all oracles.
Let $S$ be a sparse set that is $\leq_{T}^{p}$-hard for NP.

- Let $j$ be such that $(\forall n)[||S - \leq^{n}|| \leq p_{S}(n) := n^{j} + j]$.

- Let $M$ be a deterministic polynomial-time Turing machine that $\text{SAT} = L(M^{S})$. 
Let $S$ be a sparse set that is $\leq^p_T$-hard for NP.

- Let $j$ be such that $(\forall n)[\|S - \leq^n\| \leq p_S(n) := n^j + j]$.

Let $M$ be a deterministic polynomial-time Turing machine that $SAT = L(M^S)$.

- Let $k$ be such that $p_M(n) := n^k + k$ bounds the runtime of $M$ for all oracles.
NP-hard sparse set w.r.t. Turing Reduction

- Let $S$ be a sparse set that is $\leq^p_T$-hard for NP.
  - Let $j$ be such that $(\forall n)[\|S - \leq^n \| \leq p_S(n) := n^j + j]$.
- Let $M$ be a deterministic polynomial-time Turing machine that $SAT = L(M^S)$.
  - Let $k$ be such that $p_M(n) := n^k + k$ bounds the runtime of $M$ for all oracles.
- Note that $\Sigma^p_3 = NP^{NP^SAT}$. 
NP-hard sparse set w.r.t. Turing Reduction

- Let $S$ be a sparse set that is $\leq^p_T$-hard for NP.
  - Let $j$ be such that $(\forall n)[\|S - \leq n \| \leq p_S(n) := n^j + j]$.
- Let $M$ be a deterministic polynomial-time Turing machine that $SAT = L(M^S)$.
  - Let $k$ be such that $p_M(n) := n^k + k$ bounds the runtime of $M$ for all oracles.

- Note that $\Sigma^p_3 = NP^{NP^{SAT}}$.
- $\implies$ there are two nondeterministic polynomial-time Turing machines $N_1$ and $N_2$ such that $L(N_1^{L(N_2^{SAT})}) = L$. 
Let $S$ be a sparse set that is $\leq_T^p$-hard for NP.
- Let $j$ be such that $(\forall n)[\|S - \leq n\| \leq p_S(n) := n^j + j]$.

Let $M$ be a deterministic polynomial-time Turing machine that $SAT = L(M^S)$.
- Let $k$ be such that $p_M(n) := n^k + k$ bounds the runtime of $M$ for all oracles.

Note that $\Sigma_3^p = NP^{NP^{SAT}}$.

$\implies$ there are two nondeterministic polynomial-time Turing machines $N_1$ and $N_2$ such that $L(N_1^{L(N_2^{SAT})}) = L$.
- $p_N(n) := n^n + l$ bounds runtime of both $N_1, N_2$. 
Now we define, $V = V_0 \cup V_1$: (The $V_0$ checks if a given set fails to simulate SAT correctly.)

$$V_0 = \{0\#1^n\#S' \mid (\exists z \in (\Sigma^*)^{\leq n})[R]\},$$

$$V_1 = \{1\#S'\#z \mid z \in L(N_2^L(M^{S'}))\},$$
Now we define, $V = V_0 \cup V_1$: (The $V_0$ checks if a given set fails to simulate SAT correctly.)

$$V_0 = \{0\#1^n\#S' \mid (\exists z \in (\Sigma^*)^{\leq n})[R]\},$$

$$V_1 = \{1\#S'\#z \mid z \in L(N_2^{L(MS')})\},$$

For $V_1$, $L(MS')$ is in $P$ since $S'$ is finite and $M$ is in $P$. Thus $V_1 \in NP$.

We will see $V_0 \in NP$ so $V \in NP$. 


where $R :=$

1. $z$ is not a well-formed formula and $M^{S'}(z)$ accepts;
2. $z$ is a well-formed formula without free variables and either
   2.1 $M^{S'}(z)$ accepts and $z \notin SAT$ or
   2.2 $M^{S'}(z)$ rejects and $z \in SAT$
3. $z$ is a well-formed formula with variables $z_1, z_2, \ldots$ and it is not the case that: $M^{S'}(z)$ accepts iff

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

where $z[\ldots]$ denotes $z$ with the indicated variables assigned as noted.
NP-hard sparse set w.r.t. Turing Reduction

where $R :=$

1. $z$ is not a well-formed formula and $M^{S'}(z)$ accepts;
2. $z$ is a well-formed formula without free variables and either
   2.1 $M^{S'}(z)$ accepts and $z \not\in SAT$ or
   2.2 $M^{S'}(z)$ rejects and $z \in SAT$
3. $z$ is a well-formed formula with variables $z_1, z_2, \ldots$ and it is not the case that: $M^{S'}(z)$ accepts iff
   \[(M^{S'}(z[z_1 = True])) \text{ accepts } \lor M^{S'}(z[z_1 = False]) \text{ accepts},\]
   where $z[\ldots]$ denotes $z$ with the indicated variables assigned as noted.

For $V_0$, we can nondeterministically generate all $z$ and verify $R$. Note that $R$ can be verified by a deterministic polynomial time machine. Thus $V_0 \in NP$. 
From the self-reducibility of SAT, if $0\#1^n\#S'$ is not in $V_0$, we have $L(M^{S'})^\leq n = SAT^\leq n$.

**Reason:**
Obviously, either 1,2,3 satisfied means $L(M^{S'})^\leq n \neq SAT^\leq n$.
If for all $z$, 1,2,3 are unsatisfied, we have for all formula, $M^{S'}(z)$ accepts iff

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

Combined with requirement 2, we see, $M^{S'}(z)$ accepts one of its assignments $z' \in SAT$. 
Some facts: On input $y$

- the longest possible query to SAT that $N_2$ will make on queries $N_1$ asks to its oracle during the run of $N_1^{L(N_2^{SAT})}(y)$ is $p_N(p_N(|y|))$. 


Some facts: On input $y$

- the longest possible query to SAT that $N_2$ will make on queries $N_1$ asks to its oracle during the run of $N_1^{L(N_2^{SAT})}(y)$ is $p_N(p_N(|y|))$.
- $M$ on inputs of length of $p_N(p_N(|y|))$, asks its oracle only questions of length as most $p_M(p_N(p_N(|y|)))$. 
Some facts: On input $y$

- the longest possible query to SAT that $N_2$ will make on queries $N_1$ asks to its oracle during the run of $N_1^{L(N_2^{SAT})}(y)$ is $p_N(p_N(|y|))$.

- $M$ on inputs of length of $p_N(p_N(|y|))$, asks its oracle only questions of length as most $p_M(p_N(p_N(|y|)))$.

- there exists a sparse oracle $U$ such that $L(M^{U \leq p_M(p_N(p_N(|y|)))}) \leq p_N(p_N(|y|)) = SAT \leq p_N(p_N(|y|))$. (note that $S$ is such an oracle)
Some facts: On input $y$

- the longest possible query to SAT that $N_2$ will make on queries $N_1$ asks to its oracle during the run of $N_1^{L(N_2^{SAT})}(y)$ is $p_N(p_N(|y|))$.
- $M$ on inputs of length of $p_N(p_N(|y|))$, asks its oracle only questions of length as most $p_M(p_N(p_N(|y|)))$.
- there exists a sparse oracle $U$ such that $L(M(U^{\leq_{p_M(p_N(p_N(|y|))})})^{\leq_{p_N(p_N(|y|))}}) = SAT^{\leq_{p_N(p_N(|y|))}}$. (note that $S$ is such an oracle)

We now give a $NP^V$ algorithm that accepts $L$. 
Step 1: Non-deterministically guess a set
\( S' \subseteq (\Sigma^*)^{p \leq p_M(p_N(|y|))} \) satisfying
\[ \| S' \| \leq p_S(p_M(p_N(|y|))). \]
If \( 0 \# 1^{p_M(p_N(|y|))} \# S' \in V \) (actually \( V_0 \)) then reject. Otherwise go to step 2.
Step 1: Non-deterministically guess a set 
$S' \subseteq (\Sigma^*)^{p_M(p_N(|y|))}$ satisfying 
$||S'|| \leq p_S(p_M(p_N(|y|))))$. If $0\neq 1^{p_M(p_N(|y|))}\#S' \in V$ 
(actually $V_0$) 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 $V$ (actually only 
$V_1 = \{1\#S'\#z \mid z \in L(N_2^{L(MS')})\}$).
**NP-hard sparse set w.r.t. Turing Reduction**

**Step 1:** Non-deterministically guess a set $S' \subseteq (\Sigma^*)^{\leq p_M(p_N(p_N(|y|)))}$ satisfying $\|S'\| \leq p_S(p_M(p_N(p_N(|y|))))$. If $0\#1^{p_M(p_N(p_N(|y|)))} \# S' \in V$ (actually $V_0$) 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 $V$ (actually only $V_1 = \{1\#S'\#z \mid z \in L(N_2^{L(M_{S'}^{S'})})\}$).

Why it works:

\[
SAT^{\leq p_N(p_N(|y|))} = (L(M_{S'}))^{\leq p_N(p_N(|y|))}.
\]

\[
\Rightarrow (N_2(L(M_{S'}^{S'})))^{\leq p_N(|y|)} = (N_2^{SAT})^{\leq p_N(|y|)}
\]
## Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Reduction (poly)</th>
<th>Class</th>
<th>$PH =$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mahaney</td>
<td>Many-one</td>
<td>Sparse NP-complete</td>
<td>$P$</td>
</tr>
<tr>
<td></td>
<td>Many-one</td>
<td>Sparse NP-hard</td>
<td>$P$</td>
</tr>
<tr>
<td></td>
<td>Turing</td>
<td>Sparse NP-complete</td>
<td>$P^{NP[O(\log n)]}$</td>
</tr>
<tr>
<td></td>
<td>Turing</td>
<td>Sparse NP-hard</td>
<td>$NP^{NP}$</td>
</tr>
<tr>
<td>Kadin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Karp-Lipton</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table:** Sparse set theorems.