The Self-Reducibility Technique

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

Department of Computer Science
University of Rochester

November 2, 2015
Table of Contents

1  Tree Pruning Technique
   ■ Theorem 1.2
   ■ Theorem 1.4
Theorem 1.2
Theorem 1.2

Tally Set

A set $T$ is a tally set exactly if $T \subseteq 1^*$

Theorem 1.2

If there is a tally set that is $\leq_{p}^{m}$-hard for NP, then $P = NP$.

Corollary 1.3

If there is a tally set that is NP-complete, then $P = NP$.

- Let $T$ be a tally set that is $\leq_{p}^{m}$-hard. Then the NP-complete set $SAT \leq_{m}^{p} T$.
- Goal: We want to use $SAT \leq_{m}^{p} T$ to proof that $SAT$ can be decided in polynomial time. Thus, $SAT \in P$, then $P = NP$.
Tree Pruning For SAT Problem

- $F[v_i=True]$ denotes the resulting boolean formula when we assign True to variable $v_i$

- Boolean formula $F$ is satisfiable if and only if $F[v_1=True]$ is satisfiable or $F[v_1=False]$ is satisfiable.

- Find the satisfiable assignment by traversing the tree. If the traverse can be done in polynomial time, then $SAT \in P$. 
Tree Pruning For SAT Problem

- Traverse is done layer by layer. The number of nodes in $i^{th}$ layer is $2^i$.
- If during the traverse we can ignore some redundant nodes (tree pruning) so that for each layer we only traverse polynomial number of nodes, then the entire traverse is polynomial.
Example: Tree Pruning For SAT Problem

- (Rabbit says) What nodes/formulas are redundant?
- If a formula is not satisfiable, then all of its descendants are not satisfiable. Thus, this formula is redundant.
- If a formula is “identical” to another formula, then it is redundant.
- If $f_1$ is satisfiable if and only if $f_2$ is satisfiable, then $f_1$ and $f_2$ is identical.
- (Rabbit says) How do we identify the redundancy?
Tree Pruning For \textit{SAT} Problem

Layer 1

\begin{align*}
F[v_1=\text{True}] & \quad F[v_1=\text{False}] \\
\end{align*}

Layer 2

\begin{align*}
F[v_1=\text{True}, v_2=\text{True}] & \quad F[v_1=\text{True}, v_2=\text{False}] \\
\end{align*}

Layer m

\begin{align*}
F[v_1=\text{True}, v_2=\text{True}, \ldots, v_m=\text{True}] & \quad F[v_1=\text{True}, v_2=\text{True}, \ldots, v_m=\text{False}] \\
\end{align*}
Identify Redundancy

- Let \( g \) be the deterministic polynomial-time function such that \( \forall f \in SAT \) if and only if \( g(f) \in T \), where \( T \) is the \( \leq^p m \)-hard Tally set.
- Recall that \( T \subseteq 1^* \). If \( g(f) \notin 1^* \), then \( f \) is not satisfiable.
- For any two boolean formula \( f \neq h \), and \( g(f) = g(h) \), \( f \in SAT \iff h \in SAT \).

\[
f \in SAT \iff g(f) \in T \quad \parallel \quad h \in SAT \iff g(h) \in T
\]

- (Rabbit says) How do we make sure the number of remaining nodes/formulas in each layer is polynomial?
The length of the output of a polynomial-time function is bounded by some polynomial

Let $g(x)$ be a polynomial-time function, there exists an integer $k$ such that $\forall x, |g(x)| \leq |x|^k + k$

If $g(x) \in 1^*$, then the longest possible output is $1|x|^{k+k}$. Thus, the total number of possible outputs of $g(x)$ is $|x|^k + k + 1$.

Example

Given that $|g(x)| \leq |x|^k + k$ and $g(x) \in 1^*$, what are the possible outputs of $g(x)$?

$\epsilon, 1, 11, 111, 1111, 11111, \ldots, 11 \ldots 1$  
\[ |x|^{k+k} \]
Recall that for any two boolean formula $f, h$, if $g(f) = g(h)$, then $f$ and $g$ are “identical”. Similarly, if $g(f) \neq g(h)$, we say $f$ and $g$ are “distinct”.

Recall that the total number of possible outputs of $g(x)$ is $|x|^k + k + 1$.

Let $n$ be the size of formulas on the $i^{th}$ layer. Thus, among the $2^i$ formulas in this layer, at most $n^k + k + 1$ of them are “distinct”.
Proof Sketch

The Self-Reducibility Technique

Group A

Tree Pruning Technique

Theorem 1.2

Theorem 1.4

Layer 1

Layer 2

Layer m

F[v_1=True]  
F[v_1=\text{False}]  

F[v_1=True, v_2=True]  
F[v_1=True, v_2=\text{False}]  

F[v_1=True, v_2=True, ... v_m=\text{True}]  
F[v_1=True, v_2=\text{False}, ... v_m=\text{False}]  

F[v_1=\text{False}, v_2=True, ... v_m=\text{False}]  
F[v_1=\text{False}, v_2=\text{False}, ... v_m=\text{False}]
Proof Sketch

- The input of layer $i$ are the output formulas from layer $i - 1$.
- Expand each formula by assigning True and False value to $v_i$ (Get the corresponding formulas in layer $i$).
- For each expanded formula $f$ in layer $i$, calculate $g(f)$. If $g(f) \not\in 1^*$, remove $f$. If $f \in 1^*$ but exists expanded formula $h \neq f$ such that $g(f) = g(h)$, remove $f$.
- Output the resulting formulas in layer $i$. 
### Proof

**Stage 0**

Outputs $C = F$ where $F$ is the original formula.

**Stage i**

**Input** $C = \{F_1, \ldots, F_l\}$

**Step 1:** Replace $v_i$ by True or False to get

$$C = \{F_1[v_i = \text{True}], F_2[v_i = \text{True}], \ldots, F_l[v_i = \text{True}], F_1[v_i = \text{False}], F_2[v_i = \text{False}], \ldots, F_l[v_i = \text{False}]\}$$

**Step 2:** $C' = \emptyset$

**Step 3:** For each $f$ in $C$ do

1. Compute $g(f)$
2. If $g(f) \in 1^*$ and for no formula $h \in C'$ does $g(f) = g(h)$, then add $f$ to $C'$.

**Output of stage $i$** : $C = C'$

**Stage $m+1$**

Input is $C$ which is now a variable-free formula collection. $F$ is satisfiable if an element in $C$ is true.
Questions?
Theorem 1.4
Problem

The Self-Reducibility Technique

Group A

Tree Pruning Technique

Theorem 1.2

Theorem 1.4

If there is a sparse set that is $\leq_{m}^{p}$-hard for coNP, then $P=NP$.

Corollary 1.5

If there is a sparse coNP-complete set, then $P=NP$. 
Observation

Theorem 1.4

If there is a sparse set that is $\leq_m^P$-hard for $\text{coNP}$, then $\text{P=NP}$.

Definition

A set $S$ is sparse if it contains at most polynomially many elements at each length, i.e.,

$$(\exists \text{ polynomial } p)(\forall n)[\|\{x | x \in S \land |x| = n\}\| \leq p(n)].$$

Definition

A language $A$ is $\text{coNP}$-hard, if $\forall L \in \text{coNP}, L \leq_m^P A$. 
Observation

Idea

Utilize Tree-pruning trick and the definition of coNP-hard to construct a polynomial-time algorithm for SAT. (SAT is NP-complete)

Explanation

- $\forall L \in NP, L \leq_P SAT$
- SAT solved in polynomial-time by deterministic Turing machine (DTM).
- $\Leftrightarrow$ All NP problems solved in polynomial-time by DTM.
- $\Leftrightarrow P = NP.$
Observation

Let $S$ be a sparse set and also coNP-hard.

**Definition**

$\forall \ell, \, p_\ell(n)$ denotes the polynomial $n^\ell + \ell$.

**Definition**

$\|S^{\leq n}\|$ denotes the number of strings with length less than $n$ in $S$. 
Observation

Corollary

∀n, ∥S^≤n∥ ≤ p_d(n).

Proof.

- S is sparse ⇒ ∥{x|x ∈ S ∧ |x| = n}∥ ≤ p(n)
- We can obtain the upper bound \(p_{\text{max}} = \max_n p(n)\), where \(p_{\text{max}}\) is bounded by polynomial.
- ∥S^≤n∥ = \(\sum_{i=0}^{n} p(i)\) ≤ \(\sum_{i=0}^{n} p_{\text{max}} = (n + 1)p_{\text{max}}\), which is bounded by polynomial.
Observation

Recall

SAT ∈ NP ⇒ \overline{SAT} ∈ coNP and \overline{SAT} ∈ coNP ⇒ \overline{SAT} \leq^p_m S, since S ∈ coNP-hard.

Let g denote the reduction function \overline{SAT} \leq^p_m S.

Corollary

∀x, |g(x)| \leq p_k(|x|).

Proof.

- Function g is computed by a DTM
- a DTM outputs at most 1 symbol in one step
⇒ |g(x)| is bounded by polynomial length, named p_k(|x|).
Observation

Corollary

Since $\forall n, \|S^{\leq n}\| \leq p_d(n)$ and $\forall x, |g(x)| \leq p_k(|x|)$, given $g$ and $S$,

$$\|S^{\leq |g(x)|}\| \leq p_d(p_k(|x|)).$$

Rabbit: Interesting! $S^{\leq |g(x)|}$ is a set with a polynomial number of elements.
Deterministic polynomial-time algorithm for SAT

**Input**

Boolean formula $F[v_1, v_2, ..., v_m]$, w.l.o.g, $m \geq 1$.

**Stage 0**

- Collection of boolean formulas, $C' = \{ F \}$
- Pass $C'$ to Stage 1.

Rabbit: That’s pretty easy. I can do it.
The Self-Reducibility Technique

Group A

Tree Pruning Technique

Theorem 1.2

Theorem 1.4

The Self-Reducibility Technique

Group A

Tree Pruning Technique

Theorem 1.2

Theorem 1.4

Deterministic polynomial-time algorithm for SAT

Rabbit: If we keep this procedure to Stage $m$, the number of strings in each level will grow larger and larger!!!
A collection of formulas: Hi, we are from Stage i-1.

Stage i

**Step 1:** \( C = \{ F_1[v_i = True], F_2[v_i = True], \ldots, F_\ell[v_i = True], F_1[v_i = False], F_2[v_i = False], \ldots, F_\ell[v_i = False] \} \).

**Step 2:** Set \( C' = \emptyset \).

Rabbit: lol, I can do it but where is my carrot?
Stage i

**Step 3:** For each formula $f$ in $C$ do:

1. Compute $g(f)$.
2. If for no formula $h \in C'$ does $g(f) = g(h)$, then add $f$ to $C'$

---

The Self-Reducibility Technique

Group A

Tree Pruning Technique

Theorem 1.2

Theorem 1.4

Deteministic polynomial-time algorithm for SAT
Stage i

**Step 4:** If $C'$ contains at least $p_d(p_k(|F|)) + 1$ elements, stop and immediately declare that $F \in SAT$.  

**Figure:** Reduction Mapping
Deterministic polynomial-time algorithm for SAT

Stage i

Step 4: If $C'$ contains at least $p_d(p_k(|F|)) + 1$ elements, stop and immediately declare that $F \in SAT$.

Explanation

- Only $p_d(p_k(|F|))$ strings are in $S_{\leq p_k(|F|)}$.
- There is at least one formula named $H$ maps to a string in $\overline{S}$, i.e., $g(H) \notin S$.
- Since $g$ is the reduction function from $\overline{SAT}$ to $S$, $H$ is satisfiable. It implies that $F$ is satisfiable.
Deterministic polynomial-time algorithm for SAT

Stage $i$

**End Stage $i$:** $C'$ is the collection that gets passed on to Stage $i + 1$. 
Stage $m+1$

If some member of the formula collection output by Stage $m$ evaluates to being true, $F \in SAT$, and otherwise $F \notin SAT$.

Rabbit: Oh, my carrot! The proof is done here. Wait, rabbit!
Discuss

Comment

Obviously, this algorithm is computed by deterministic Turing machine.

- **Step 4 never met**
  Upper bound number of strings $p_{\text{max}} = \max p_d(p_k(|F|))$.
  ⇒ time for whole algorithm $t \leq mp_{\text{max}}$

- **Step 4 invoked**
  This algorithm stops early before Stage $m+1$.
  ⇒ The algorithm is polynomial-time.

We construct a deterministic polynomial-time algorithm for SAT.

Rabbit: If I find a carrot like this set $S$, I will buy a million carrots (plus 9 millions).
Thank You!