Mahaney’s Theorem

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

November 2016
Definitions (cont.)

- Census function: 
  For any set $A$, $c_A(n) = \| \{ x \in A \mid |x| \leq n \} \|$
- Examples:
  - $c_{\emptyset}(n) = 0$
  - $c_{SAT}$ is exponential
- Density: Rate of growth of census function
Definitions (cont.)

- Finite sets have a bounded census function (if $|A| < \infty$, $\exists M \forall n \ c_A(n) \leq M$)

- Sparse sets:
  $A$ is sparse if there is a polynomial $p$ such that $c_A \leq p$

- Examples:
  - Any finite set
  - $\{n \mid n \text{ is a power of } 2\}$
  - $A$ such that $A$ contains the first $n$ lexicographically first strings of length $n$
P-Isomorphic Languages

- **Definition:** Languages $A \subseteq \Sigma_1^*$ and $B \subseteq \Sigma_2^*$ are p-isomorphic if there exists a bijection $f : \Sigma_1^* \rightarrow \Sigma_2^*$ such that:
  - $x \in A \iff f(x) \in B$
  - $f$ and $f^{-1}$ are both polynomial-time functions.

- Berman–Hartmanis conjecture: Any two NP-complete languages are p-isomorphic.
P-Isomorphic Languages

Definition: A set $A$ is paddable if there is an injective function $f$ such that $f$ and $f^{-1}$ are polynomial-time functions, and $\forall x, y : x \in A \iff f(x, y) \in A$.

All pairs of paddable NP-complete languages are p-isomorphic.

Equivalent conjecture: all NP-complete languages are paddable.
Theorem

If an NP-Complete sparse language exists such that its census function is computable in polynomial time, then $P=NP$.

(Later we will do away with the P-time census function requirement!)
**Theorem 5.6 Proof**

**Proof:** Let $S$ be an NP-Complete language whose census function $c_s$ is computable in polynomial time. Denote by $NT$ the NDTM which decides $S$ in polynomial time.
The following nondeterministic algorithm shows that the complement language $S^c$ of $S$ is also in NP:

begin \{input: $x$\}

\begin{align*}
  n &:= |x|; \\
  k &:= c_S(n); \\
  \text{guess } y_1, \ldots, y_k \text{ in set of } k\text{-tuples of distinct words} \\
  \text{each of which has length, at most, } n; \\
  \{\text{check whether the guessed } k\text{-tuple coincides with } S_{\leq n}\} \\
  \text{for } i = 1 \text{ to } k \text{ do} \\
  \hspace{1em} \text{if } NT(y_i) \text{ rejects then reject;} \\
  \{\text{check if } x \in S_{\leq n}\} \\
  \text{for } i = 1 \text{ to } k \text{ do} \\
  \hspace{1em} \text{if } y_i = x \text{ then reject;} \\
  \text{accept;}
\end{align*}

end.
So now we have a nondeterministic P-time algorithm to decide $S^c$, and we also know that $S$ is NP-complete. Then we know that both $SAT$ and $S^c$ can reduce to $S$, and also that $SAT^c$ can reduce to $S$.

This is true because we know $SAT \leq S$ and $S^C \leq S$, and from this property of reductions and complements:

$A \leq B \rightarrow A^C \leq B^C$, we can say $SAT^C \leq S^C$. By transitivity then $SAT^C \leq S^C$ and $S^C \leq S$, we know $SAT^C \leq S$.

We will now use this to show that $SAT$ is in P.
Let $x$ be a problem instance of $SAT$. We construct a binary tree $A_x$ by the following rules:

1. Label the root with the original formula $x$

2. If a node is labeled with $y$, then label its children ($y_0$ and $y_1$) with the two formulas obtained by setting the value of one variable in $y$ to \textbf{false} and \textbf{true} and performing all resulting simplifications.
Theorem 5.6 Proof continued

Example of resulting tree:

\[(x_1 \lor x_2 \lor \neg x_3) \land (\neg x_2) \land (\neg x_2 \lor x_3)\]

\[
x_1 = t \quad x_1 = f
\]

\[
(\neg x_2) \land (\neg x_2 \lor x_3) \quad (x_2 \lor \neg x_3) \land (\neg x_2) \land (\neg x_2 \lor x_3)
\]

\[
x_2 = t \quad x_2 = f \quad x_2 = t \quad x_2 = f
\]

\[
false \quad true \quad false \quad (\neg x_3)
\]

\[
x_3 = t \quad x_3 = f
\]

\[
false \quad true
\]
Clearly $x$ is satisfiable iff at least one leaf of $A_x$ is labeled with \textbf{true}. Checking all of them would take exponential time, but we don’t need to do that.

We can prune the tree using a reduction $f$ from $SAT^c$ to $S$. When we visit a node labeled $y$, we will check whether $f(y) \in S$. If that is the case, then $y$ is not satisfiable and we eliminate the subtree under $y$. 
Since $S$ is NP-complete, instead of checking it directly we will build and maintain a list of words in $S$ while visiting the nodes, and will instead check for membership in that list. Initially the list only contains $f(\text{false})$, and whenever both children of a node are in the list, then the label of that node is added to the list. Since $S$ is sparse, the list contains at most a polynomial number of words with respect to the length of $x$. 
To prove that this \emph{SAT} algorithm requires a polynomial number of steps, consider two unsatisfiable formulas, $y$ and $z$ such that $f(y) = f(z)$ and such that the corresponding nodes are inner ones of the visited subtree (That is, they are not leaves).

\begin{itemize}
  \item These nodes must be contained in a common path from the root to one of the leaves.
  \item Otherwise, one of the nodes (for instance $y$) would have been visited first and $f(y)$ would have been added to the list and the search would have stopped at the node labeled $z$, contrary to the assumption that the node is an inner one of the visited subtree.
\end{itemize}
Sooooo, why is it polynomial time?

- Oh right. Well, the number of distinct paths from the root to the inner nodes of the visited subtree is at most $p(q(|x|))$ where $p$ is a polynomial witnessing the sparsity of $S$ and $q$ is a polynomial limiting the length of $f$.

- Since $A_x$ has height $n$, the visited subtree has at most $np(q(|x|))$ inner nodes labeled with unsatisfiable formulas. An assignment satisfying $x$ may require the visit of $n - 1$ additional inner nodes.

- Thus the visited subtree includes at most $np(q(|x|)) + n - 1$ inner nodes. If we consider the leaves, this only increases by a factor of 2.
In summary, we just did the following:

1. Given a census function for $S$ which is computable in polynomial time, prove that $S^c \in NP$

2. Using the previous result, reduce $SAT^c$ to $S^c$ and thus to $S$

3. Using a reduction from $SAT^c$ to $S$ as a pruning function, visit the tree $A_x$ of possible assignments for $x$ and show that the pruning is drastic enough to cut the number of visited nodes from an exponential number to a polynomial one
Theorem 5.7: Mahaney’s Theorem

**Theorem**

*If a sparse NP-complete language exists, then P=NP.*
Theorem 5.7: Mahaney’s Theorem

Theorem

If a sparse NP-complete language exists, then $P=NP$.

Proof:
Let $S$ be a sparse NP-complete language, $p$ a polynomial witnessing the sparsity of $S$ and $NT$ a NDTM which decides $S$ in polynomial time.
Theorem 5.7: Mahaney’s Theorem

If a sparse NP-complete language exists, then $P=NP$.

Proof:
Let $S$ be a sparse NP-complete language, $p$ a polynomial witnessing the sparsity of $S$ and $NT$ a NDTM which decides $S$ in polynomial time.

Similar to Theorem 5.6, we will show $PC(S) \in NP$ where $PC(S)$ refers to the pseudo-complement of $S$. 
Theorem 5.7: Mahaney’s Theorem

Theorem

If a sparse NP-complete language exists, then $P = NP$.

Proof:
Let $S$ be a sparse NP-complete language, $p$ a polynomial witnessing the sparsity of $S$ and $NT$ a NDTM which decides $S$ in polynomial time.

Similar to Theorem 5.6, we will show $PC(S) \in NP$ where $PC(S)$ refers to the pseudo-complement of $S$. 
Theorem 5.7 Proof: Part 1 – Building a NDTM to decide $PC(S)$

This NDTM clearly runs in nondeterministic polynomial time, so $PC(S) \in NP$. 

```
begin {input: x, k, 0^n}
if |x| > n \lor k > p(n) then reject;
guess y_1, \ldots, y_k in set of k-tuples of distinct values
each of which is of length, at most, n;
for i = 1 to k do
  if NT(y_i) rejects then reject;
for i = 1 to k do
  if y_i = x then reject;
accept;
end.
```
Theorem 5.7 Proof: Part 1 – Building a NDTM to decide $PC(S)$

- Note: Since $PC(S) \in NP$, we know $PC(S) \leq S$. 
Theorem 5.7 Proof: Part 2 – Use our NDTM that decides $PC(S)$ to show $SAT \in P$

- Let $h$ be the reduction from $SAT$ to $S$, and $p_h$ be its bounding polynomial
- Let $g$ be the reduction from $PC(S)$ to $S$, and $p_g$ be its bounding polynomial
- if $k = c_S(m)$ (i.e. correctly guessed $k$),

Mahaney’s Theorem 20/24
Theorem 5.7 Proof: Part 2 – Use our NDTM that decides $PC(S)$ to show $SAT \in P$

Case 1: Correctly guessed $k$
- if $k = c_S(m)$, then for an unsatisfiable $y$, $g(h(y), k, 0^m) \in S$

Case 2: Incorrectly guessed $k$
- if $k \neq (m)$, we cannot be certain about $g(h(y), k, 0^m)$’s relationship with $S'$
Theorem 5.7 Proof: Part 2 – Use our NDTM that decides $PC(S)$ to show $SAT \in P$

- Let us define $f_{n,k}(y) = g(h(y), k, 0^{p_h(n)})$.
- For each $k$, the function $f_{n,k}$ is computable in polynomial time with respect to the length of $y$, where $|y| \leq n$.
- This is our set of polynomially many pruning functions.
Theorem 5.7 Proof: Part 2 – Use our NDTM that decides $PC(S)$ to show $SAT \in P$

- if $k = c_s(p_h(n))$, then a constant $c_1$ depending on $p$ and an integer $n_0$ exists such that
  \[
  |\langle y, k, 0^{p_h(n)} \rangle| \leq 2p_h(n) + c_1 \log(p_h(n)) \leq 3p_h(n), \text{ for all } n \geq n_0.
  \]
- The unsatisfiable formulas of length, at most, $n$ are transformed from $f_{n,k}$ in, at most $p(p_g(3p_h(n)))$ distinct words of $S$, for each $n \geq n_0$.
Theorem 5.7 Proof: Decide SAT in \( P \) time

begin \{ \text{input: } x \} \\
\text{for } k = 0 \text{ to } p(p_h(|x|)) \text{ do} \\
\text{begin} \\
\text{execute the tree-visiting algorithm described in the} \\
\text{proof of Theorem 5.6 using } f_{|x|,k} \text{ as a pruning function} \\
\text{and visiting, at most, } |x|p(p_g(3p_h(|x|))) + |x| - 1 \text{ inner nodes;} \\
\text{if the algorithm accepts then accept;} \\
\text{end;} \\
\text{reject;} \\
\text{end.}

\begin{itemize}
  \item This clearly runs in polynomial time, therefore \( SAT \in P \)
\end{itemize}