15-451
Algorithms

Theory of NP-Completeness

Randal E. Bryant
April 11, 2001

Topics:
  • Turing Machines
  • Cook’s Theorem
  • Implications
Turing Machine

Formal Model of Computer

- Very primitive, but computationally complete
Turing Machine Components

Tape
- Conceptually infinite number of “squares” in both directions
- Each square holds one “symbol”
  - From a finite alphabet
- Initially holds input + copies of blank symbol ‘B’

Tape Head
- On each step
  - Read current symbol
  - Write new symbol
  - Move Left or Right one position
Components (Cont.)

Controller

- Has state between 0 and m-1
  - Initial state = 0
  - Accepting state = m-1
- Performs steps
  - Read symbol
  - Write new symbol
  - Move head left or right

Program

- Set of allowed controller actions
- Current State, Read Symbol $\rightarrow$ New State, Write Symbol, L|R
Turing Machine Program Example

Language Recognition
  • Determine whether input is string of form $0^n1^n$

Input Examples

- Should reach state $m-1$

  
  ```
  \[
  \cdots B \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ B \ B \ \cdots
  \]
  ```

- Should never reach state $m-1$

  
  ```
  \[
  \cdots B \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ B \ \cdots
  \]
  ```
Algorithm

- Keep erasing 0 on left and 1 on right
- Terminate and accept when have blank tape
# Program

## States
- 0 Initial
- 1 Check Left
- 2 Scan Right
- 3 Check Right
- 4 Scan Left
- 5 Accept

<table>
<thead>
<tr>
<th>Current State</th>
<th>B</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1,B,R</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>5,B,R</td>
<td>2,B,R</td>
<td>—</td>
</tr>
<tr>
<td>2</td>
<td>3,B,L</td>
<td>2,0,R</td>
<td>2,1,R</td>
</tr>
<tr>
<td>3</td>
<td>—</td>
<td>—</td>
<td>4,B,L</td>
</tr>
<tr>
<td>4</td>
<td>1,B,R</td>
<td>4,0,L</td>
<td>4,1,L</td>
</tr>
<tr>
<td>5</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

'—' means no possible action from this point

Deterministic TM: At most one possible action at any point
Non Deterministic Turing Machine

Language Recognition
- Determine whether input is string of form \( xx \)
- For some string \( x \in \{0,1\}^* \)

Input Examples

\[
\begin{array}{cccccc}
& & & & & \\
\cdots & B & 0 & 1 & 1 & 0 & 1 & 1 & B & B & \cdots \\
\cdots & B & 0 & 1 & 1 & 0 & 1 & 1 & 1 & B & \cdots \\
\end{array}
\]

- Should reach state \( m-1 \)
- Should never reach state \( m-1 \)
Nondeterministic Algorithm

- Record leftmost symbol and set to B

- Scan right, stopping at arbitrary position with matching symbol, and mark it with 2

- Scan left to end, and run program to recognize $x2^x$
Nondeterministic Algorithm

- Might make bad guess

- Program will never reach accepting state

Rule

- String accepted as long as reach accepting state for some sequence of steps
# Nondeterministic Program

## States
- 0 Initial
- 1 Record
- 2 Look for 0
- 3 Look for 1
- 4 Scan Left
- 5+ Rest of program

## Read Symbol

<table>
<thead>
<tr>
<th>Current State</th>
<th>B</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1,B,R</td>
<td>2,B,R</td>
<td>3,B,R</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>2,0,R</td>
<td>4,2,L</td>
</tr>
<tr>
<td>4</td>
<td>5,B,R</td>
<td>4,0,L</td>
<td>4,1,L</td>
</tr>
</tbody>
</table>

Nondeterministic TM: \( \geq 2 \) possible actions from single point
Turing Machine Complexity

Machine $M$ Recognizes Input String $x$
- Initialize tape to $x$
- Consider all possible execution sequences
- Accept in time $t$ if can reach accepting state in $t$ steps
  - $t(x)$: Length of shortest accepting sequence for input $x$

Language of Machine $L(M)$
- Set of all strings that machine accepts
- $x \notin L$ when no execution sequence reaches accepting state
  - Might hit dead end
  - Might run forever

Time Complexity
- $T_M(n) = \text{Max} \{ t(x) \mid x \in L \mid x \mid = n \}$
  - Where $|x|$ is length of string $x$
P and NP

Language L is in P
• There is some deterministic TM M
  - L(M) = L
  - T_M(n) = p(n) for some polynomial function p

Language L is in NP
• There is some nondeterministic TM M
  - L(M) = L
  - T_M(n) = p(n) for some polynomial function p
• Any problem that can be solved by intelligent guessing
Example: Boolean Satisfiability

Problem

• Variables: $x_1, \ldots, x_k$
• Literal: either $x_i$ or $\neg x_i$
• Clause: Set of literals
• Formula: Set of clauses
• Example: $\{x_3, \neg x_3\} \{x_1, x_2\} \{\neg x_2, x_3\} \{x_1, \neg x_3\}$
  - Denotes Boolean formula $x_3 \lor \neg x_3 \land x_1 \lor x_2 \land \neg x_2 \lor x_3 \land x_1 \lor \neg x_3$
Encoding Boolean Formula

Represent each clause as string of 2k 0's and 1's

- 1 bit for each possible literal
- First bit: variable, Second bit: Negation of variable

- \{x_3, \neg x_3\}: 000011 {x_1, x_2}: 101000
- \{\neg x_2, x_3\}: 000110 {x_1, \neg x_3}: 100001

Represent formula as clause strings separated by '$'

- 000011$101000$000110$100001
SAT is NP

Claim

• There is a NDTM $M$ such that $L(M) =$ encodings of all satisfiable Boolean formulas

Algorithm

• Phase 1: Determine $k$ and generate some string $\{01, 10\}$
  - Append to end of formula
  - This will be a guess at satisfying assignment
  - E.g., $000011$\$101000$\$000110$\$100001$\$100110$

• Phase 2: Check each clause for matching 1
  - E.g., $000011$\$101000$\$000110$\$100001$\$100110$
SAT is NP-complete

Cook's Theorem

- Can generate Boolean formula that checks whether NDTM accepts string in polynomial time

Translation Procedure

- Given
  - NDTM $M$
  - Polynomial function $p$
  - Input string $x$

- Generate formula $F$
  - $F$ is satisfiable iff $M$ accepts $x$ in time $p(|x|)$
  - Size of $F$ is polynomial in $|x|$
  - Procedure generates $F$ in (deterministic) time polynomial in $|x|$
Construction

Parameters

- $|x| = n$
- $m$ states
- $v$ tape symbols (including B)

Formula Variables

- $Q[i,k] \quad 0 = i = p(n), \ 0 = k = m-1$
  - At time $i$, $M$ is in state $k$
- $H[i,j] \quad 0 = i = p(n), \ -p(n) = j = p(n)$
  - At time $i$, tape head is over square $j$
- $S[i,j,k] \quad 0 = i = p(n), \ -p(n) = j = p(n), \ 1 = k = v$
  - At time $i$, tape square $j$ holds symbol $k$

Key Observation

- For bounded computation, can only visit bounded number of squares
Clause Groups

- Formula clauses divided into "clause groups"

Uniqueness
- At each time $i$, $M$ is in exactly one state
- At each time $i$, tape head over exactly one square
- At each time $i$, each square $j$ contains exactly one symbol

Initialization
- At time 0, tape encodes input $x$, head in position 0, controller in state 0

Accepting
- At some time $i$, state = $m-1$

Legal Computation
- Tape/Head/Controller configuration at each time $i+1$ follows from that at time $i$ according to some legal action
Implications of Cook’s Theorem

Suppose There Were an Efficient Algorithm for Boolean Satisfiability

• Then could take any problem in NP, convert it to Boolean formula and solve it quickly!
• Many “hard” problems would suddenly be easy

Big Question $P =? NP$

• Formulated in 1971
• Still not solved
• Most believe not
Complements of Problems

Language Complement

• Define $\sim L = \{ x \mid x \notin L \}$
• E.g., $\sim$SAT
  - Malformed formulas (easy to detect)
  - Unsatisfiable formulas

P Closed Under Complementation

• If L is in P, then so is $\sim$L
  - Run TM for L on input $x$ for $p(|x|)$ steps
    » Has unique computation sequence
  - If haven’t reached accepting state by then, then $x \notin L$
NP vs. co-NP (cont.)

Is NP = co-NP?

- Having NDTM for \( \sim L \) doesn't help for recognizing \( L \)
  - Would have to check all computation sequences of length \( p(|x|) \).
  - Could have exponentially many sequences

Proper Terminology

- Generally want algorithm that can terminate with "yes" or "no" answer to decision problem
- If underlying problem (or its complement) is NP, then full decision problem is "NP-Hard"
Showing Problems NP-Complete

To show Problem X is NP-complete

1. **Show X is in NP**
   - Can be solved by “guess and check”
   - Generally easy part

2. **Show known NP-complete problem Y can be reduced to X**
   - Devise translation procedure
   - Given arbitrary instance y of Y, can generate problem x in X such that $y \in L_Y$ iff $x \in L_X$
     - $L_X$: set of all strings x for which decision problem answer is “yes”
   - Size of x must be polynomial in y, and must be generated by (deterministic) polynomial algorithm.