Universal Algebra and Computational Complexity
Lecture 1

Ross Willard

University of Waterloo, Canada

Třešť’, September 2008
Lecture 1: Decision problems and Complexity Classes

Lecture 2: Nondeterminism, Reductions and Complete problems

Lecture 3: Results and problems from Universal Algebra
Three themes: problems, algorithms, efficiency

A *Decision Problem* is . . .

- A *YES/NO question*
  - parametrized by one or more *inputs*.
    - Inputs must:
      - range over an *infinite* class.
      - be “finitistically described”

What we seek:

- An *algorithm* which correctly answers the question for every possible inputs.

What we ask:

- How *efficient* is this algorithm?
- Is there a better (more efficient) algorithm?
Directed Graph Reachability problem (PATH)

INPUT:

- A finite directed graph $G = (V, E)$
- Two distinguished vertices $v_{start}, v_{end} \in V$.

QUESTION:

- Does there exist in $G$ a directed path from $v_{start}$ to $v_{end}$?
  (i.e., a sequence $v_{start} = v_0, v_1, v_2, \ldots, v_k = v_{end}$ of vertices such that $(v_i, v_{i+1}) \in E$ for all $0 \leq i < k$.)
An Algorithm for PATH

$v_{start} =$

Answer: “NO”

$v_{end} =$
Efficiency of this algorithm

How long does this algorithm take?

- I.e., how many steps?
- ...as a function of the size of the input graph.

I’ll give three answers to this.
Only action is changing a vertex’s color.

Only changes possible are

- white ⇒ red
- red ⇒ blue
- blue ⇒ green.

So if $n = |V|$, then the algorithm requires at most $3n$ vertex-color changes.
Second answer – pseudo-code

Assume that $V = \{0, 1, \ldots, n - 1\}$ and $E$ is encoded by the adjacency matrix $M_E = [e_{i,j}]$ where

$$e_{i,j} = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{else.} \end{cases}$$

For $i < n$ let $c_i$ be a variable recording the color of vertex $i$.

Also let $\text{Greenvar}$ be a variable storing whether there are green-colored vertices.
Algorithm:

- Input $n$, $M_E$, $start$ and $end$.
- For $i = 0$ to $n - 1$ set $c_i := white$.
- Set $c_{start} = green$.
- Set $GreenVar := yes$.
- While $GreenVar = yes$ do:
  - For $i = 0$ to $n - 1$; for $j = 0$ to $n - 1$
    - if $e_{i,j} = 1$ and $c_i = green$ and $c_j = white$ then set $c_j := red$.
  - For $i = 0$ to $n - 1$
    - If $c_i = green$ then set $c_i := blue$
  - Set $GreenVar := no$
  - For $i = 0$ to $n - 1$
    - If $c_i = red$ then (set $c_i := green$ and set $GreenVar := yes$)
  - If $c_{end} = blue$ then output YES; else output NO.

$n$ loops
$n^2$ cases

$O(n^3)$ steps
if $n = |V|$
Again assume $V = \{0, 1, \ldots, n - 1\}$.

Assume also that $v_{\text{start}} = 0$ and $v_{\text{end}} = 1$.

Assume the adjacency matrix is presented as a binary string of length $n^2$.

Implement the algorithm on a *Turing machine*.
Turing machine

Input (ROM): 001101111001001010110011001

R/W Tape 1: aacecaeeacacecececa

R/W Tape 2: 70325300167230151500067231

R/W Tape 3: xoxoxxooxoxxo

R/W Tape 4: ACHXXO01PLEAS*SEND*HELP*A

R/W Tape 5: 00110111100100101011001

Output bit: 

<table>
<thead>
<tr>
<th>Tape</th>
<th>In</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td></td>
<td>c</td>
<td>1</td>
<td>x</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>1st?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
Implementing the algorithm for \textit{PATH}

\begin{align*}
\text{Input:} & \quad \begin{array}{c}
\begin{array}{cccccccccccccccc}
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}
\end{array} \\
\end{align*}

\begin{align*}
\text{Tape 1:} & \quad \begin{array}{c}
\begin{array}{cccccccccccccccc}
\text{GreenVar} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \end{array}
\end{array} \\
\end{align*}

\begin{align*}
\text{Tape 2:} & \quad \begin{array}{c}
\begin{array}{cccccccccccccccc}
\text{GreenVar} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \end{array}
\end{array} \\
\end{align*}

\begin{align*}
\text{Tape 3:} & \quad \begin{array}{c}
\begin{array}{cccccccccccccccc}
\text{GreenVar} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \end{array}
\end{array} \\
\end{align*}

\begin{align*}
\text{Tape 4:} & \quad \begin{array}{c}
\begin{array}{cccccccccccccccc}
\text{GreenVar} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \end{array}
\end{array} \\
\end{align*}

\begin{align*}
\text{Tape 5:} & \quad \begin{array}{c}
\begin{array}{cccccccccccccccc}
\text{GreenVar} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \end{array}
\end{array} \\
\end{align*}

\begin{align*}
\text{Main loop:} & \quad \text{For } i, j = 0 \text{ to } n - 1 \ldots
\end{align*}
Pseudo-code revisited

Point: overhead needed to keep track of $i, j, c_i, c_j$.

Thus:

- While $\text{GreenVar} = \text{yes}$ do:
  - For $i = 0$ to $n - 1$; for $j = 0$ to $n - 1$
    - if $e_{i,j} = 1$ and $c_i = \text{green}$ and $c_j = \text{white}$
      then set $c_j := \text{red}$.

$O(n \log n)$ steps

$O(n^4 \log n)$ steps (Time)

$O(n)$ memory cells (Space)

Turing machine

### SUMMARY:

on an input graph $G = (V, E)$ with $|V| = n$, our algorithm decides the answer to $\text{PATH}$ using:

<table>
<thead>
<tr>
<th>Heuristics</th>
<th>$3n$ color changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pseudo-code</td>
<td>$O(n^3)$ operations</td>
</tr>
<tr>
<td>Turing machine</td>
<td>$O(n^4 \log n)$ steps (Time)</td>
</tr>
<tr>
<td></td>
<td>$O(n)$ memory cells (Space)</td>
</tr>
</tbody>
</table>
Let $f : \mathbb{N} \to \mathbb{N}$ be given.

A decision problem $D$ (with a specified encoding of its inputs) is:

1. in $\text{TIME}(f(N))$ if there exists a Turing machine solving $D$ in at most $O(f(N))$ steps on inputs of length $N$.

2. in $\text{SPACE}(f(N))$ if there exists a Turing machine solving $D$ requiring at most $O(f(N))$ memory cells (not including the input tape) on inputs of length $N$. 
Complexity of \textit{PATH}

Recall that our Turing machine solves \textit{PATH} on graphs with \( n \) vertices in

- Time: \( O(n^4 \log n) \) steps
- Space: \( O(n) \) memory cells.

Since “length \( N \) of input” = \( n^2 \) (when \( n = |V| \)), we have

\[
\begin{align*}
\text{PATH} & \in \ \text{TIME}(N^{2+\epsilon}) \\
\text{PATH} & \in \ \text{SPACE}(\sqrt{N})
\end{align*}
\]

(Question: can we do better?...)
Another problem: Boolean Formula Value ($FVAL$)

INPUT:
- A boolean formula $\varphi$ in propositional variables $x_1, \ldots, x_n$.
- A sequence $\mathbf{c} = (c_1, \ldots, c_n) \in \{0, 1\}^n$.

QUESTION:
- Is $\varphi(\mathbf{c}) = 1$?
An algorithm for \( FVAL \)

\[
\varphi = (((x_2 \lor x_4) \lor (\neg (x_3))) \land ((x_1 \land x_4) \rightarrow (x_3 \lor x_2))) \rightarrow (\neg (x_3 \land (x_1 \lor x_3))), \quad c = (1,0,1,1).
\]

Seems to use \( TIME(N) \) and \( SPACE(N) \).

But space can be re-used. In this example, 3 memory bits suffice.
In general, a bottom-up computation, always computing a larger subtree first, can be organized to need only $O(\log |\varphi|)$ intermediate values.

A careful implementation on a Turing machine yields:

**Theorem (Nancy Lynch, 1977).**

\[
\begin{align*}
FVAL & \in \text{TIME}(N^{2+\epsilon}) \\
FVAL & \in \text{SPACE}(\log N).
\end{align*}
\]
A third problem: Graph 3-Colorability ($3\text{COL}$)

INPUT: a finite graph $G = (V, E)$.

QUESTION: Is it possible to color the vertices red, green or blue, so that no two adjacent vertices have the same color?

Equivalently: does there exist a homomorphism

$$\chi : G \rightarrow K_3$$
An algorithm for 3COL

Brute force search algorithm:

- For each function $\chi : V \rightarrow K_3$:
  - Test if $\chi$ works.

\[ 3|V| = 2^{O(\sqrt{N})} \] loops

\[ O(N^2) \] time,
\[ O(\sqrt{N}) \] space

This at least proves:

\[ 3COL \in \text{TIME}(2^{O(\sqrt{N})}) \]
\[ 3COL \in \text{SPACE}(\sqrt{N}) \]

Open problem: can 3COL be solved in polynomial time?
A fourth problem: Clone membership (CLO)

INPUT:
- A finite algebra $\mathbf{A} = \langle A; f_1, \ldots, f_k \rangle$.
- An operation $g : A^n \rightarrow A$.

QUESTION: Is $g$ a term operation of $\mathbf{A}$?

All known algorithms essentially generate the full $n$-generated free algebra in $V(\mathbf{A})$,

$$F_n \leq A^{(A^n)}$$

and test whether $g \in F_n$.

In the worst case this could require as much as $|A^{(A^n)}| = 2^O(|A|^n)$ time and space.

**Theorem:** We cannot solve CLO in polynomial time.
Some important complexity classes

1. \( P = \text{PTIME} = \bigcup_{k=1}^{\infty} \text{TIME}(N^k) = \text{TIME}(N^{O(1)}) \).

2. \( \text{PSPACE} = \bigcup_{k=1}^{\infty} \text{SPACE}(N^k) = \text{SPACE}(N^{O(1)}) \).

Problems known to be in \( P \) are said to be feasible or tractable.

3. \( \text{EXPTIME} = \bigcup_{k=1}^{\infty} \text{TIME}(2^{N^k}) = \text{TIME}(2^{N^{O(1)}}) \).

4. \( L = \text{LOGSPACE} = \text{SPACE}(\log(N)) \).

\[ L \subseteq P \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \]

\[ \cup \quad \cup \quad \cup \quad \cup \]

\[ \text{PATH} \quad \text{3COL} \quad \text{CLO} \quad \text{FVAL} \]
In tomorrow’s lecture I will:

- Introduce “nondeterministic” versions of these 4 classes.
- Introduce problems which are “hardest” for each class.