Computational Models

Introduction to the Theory of Computing

Instructor: Prof. Benny Chor  (benny@cs.tau.ac.il)

Teaching Assistants: Mr. Udi Boker  (udiboker@tau.ac.il)
                   Mr. Yuval Inbar  (inbaryuv@post.tau.ac.il)

Tel-Aviv University


http://www.cs.tau.ac.il/~bchor/CM07/compute.html

Site is our sole means of disseminating messages (no mailing list or forum).
Course Requirements:

- 6 problem sets (10% of grade, best 5-out-of-6).
- Readable, concise, correct answers expected.
- Late submission will not be accepted.
- Solving assignments is crucial for exam.
- Final exam (90% of grade).
- Midterm exam (10% of midterm grade added to weighted average of final exam and homework).
- Midterm tentatively scheduled to Fri., April 13.
- Second final exam (Moed B): Same material and same averaging applies. Format may differ (e.g. proportion of closed/open problems),
Prerequisites:
- Extended introduction to computer science (aka Scheme course).
- Discrete mathematics course.
- Students from other disciplines with some mathematical background are encouraged to contact the instructor.

Textbook (extensively used, highly recommended):
Why Study Theory?

- Basic Computer Science Issues
  - What is a computation?
  - Are computers omnipotent?
  - What are the fundamental capabilities and limitations of computers?

- Pragmatic Reasons
  - Avoid intractable or impossible problems.
  - Apply efficient algorithms when possible.
  - Learn to tell the difference.

Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University.
Course Topics

• **Automata Theory:** What is a computer?
• **Computability Theory:** What can computers do?
• **Complexity Theory:** What makes some problems computationally hard and others easy?
• **Coping with intractability:**
  • Approximation.
  • Randomization.
  • Fixed parameter algorithms.
  • Heuristics.
Automata Theory - Simple Models

- **Finite automata.**
  - Related to controllers and hardware design.
  - Useful in text processing and finding patterns in strings.
  - Probabilistic (Markov) versions useful in modeling various natural phenomena (e.g. speech recognition).

- **Push down automata.**
  - Tightly related to a family of languages known as **context free languages**.
  - Play important role in compilers, design of programming languages, and studies of **natural languages**.

Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University.
In the first half of the 20th century, mathematicians such as Kurt Gödel, Alan Turing, and Alonzo Church discovered that some fundamental problems cannot be solved by computers.

- **Proof verification** of statements can be automated.
- It is natural to expect that determining validity can also be done by a computer.
- **Theorem:** A computer cannot determine if mathematical statement true or false.
- Results needed theoretical models for computers.
- These theoretical models helped lead to the construction of real computers.
Computability Theory

a simplicial complex
Paths and Loops

- **a path** is a sequence of vertices connected by edges
- **a loop** is a path that ends and ends at the same vertex
Paths and Loops can be Deformed

\[(v_0, v_1) \Leftrightarrow (v_0, v_2, v_1)\]

\[(v_0, v_0) \Leftrightarrow (v_0)\]
No algorithm can determine whether an arbitrary loop in an arbitrary finite complex is contractible.
Some Other Undecidable Problems

- Does a program run forever?
- Is a program correct?
- Are two programs equivalent?
- Is a program optimal?
- Does an equation with one or more variables and integer coefficients ($5x + 15y = 12$) have an integer solution (Hilbert’s 10th problem).
- Is a finitely-presented group trivial?
- Given a string, $x$, how compressible is it?
Complexity Theory

Key notion: **tractable** vs. **intractable** problems.

- **A problem** is a general computational question:
  - description of parameters
  - description of solution

- **An algorithm** is a step-by-step procedure
  - a recipe
  - a computer program
  - a mathematical object

- **We want the most efficient algorithms**
  - fastest (usually)
  - most economical with memory (sometimes)
  - expressed as a function of problem size
Example: Traveling Salesman Problem

Input:
- set of cities
- set of inter-city distances

(not drawn to scale)
Example: Traveling Salesman Problem

Goal:
- want the shortest tour through the cities
- example: a, b, d, c, a has length 27.
What is an appropriate measure of problem size?
- $m$ nodes?
- $m(m + 1)/2$ distances?

Use an encoding of the problem alphabet of symbols:

Measures
- **Problem Size**: length of encoding (here: 23 ascii characters).
- **Time Complexity**: how long an algorithm runs, as function of problem size?
Time Complexity - What is tractable?

- We say that a function \( f(n) \) is \( O(g(n)) \) if there is a constant \( c \) such that for large enough \( n \),
  \[ |f(n)| \leq c \cdot |g(n)|. \]

- A polynomial-time algorithm is one whose time complexity is \( O(p(n)) \) for some polynomial \( p(n) \), where \( n \) denotes the length of the input.

- An exponential-time algorithm is one whose time complexity cannot be bounded by a polynomial (e.g., \( n^{\log n} \)).
Tractability – Basic distinction:

- **Polynomial time** = tractable.
- **Exponential time** = intractable.

<table>
<thead>
<tr>
<th>$n$</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>.00001</td>
<td>.00002</td>
<td>.00003</td>
<td>.00004</td>
<td>.00005</td>
<td>.00006</td>
</tr>
<tr>
<td></td>
<td>second</td>
<td>second</td>
<td>second</td>
<td>second</td>
<td>second</td>
<td>second</td>
</tr>
<tr>
<td>$n^2$</td>
<td>.00001</td>
<td>.00004</td>
<td>.00009</td>
<td>.00016</td>
<td>.00025</td>
<td>.00036</td>
</tr>
<tr>
<td></td>
<td>second</td>
<td>second</td>
<td>second</td>
<td>second</td>
<td>second</td>
<td>second</td>
</tr>
<tr>
<td>$n^3$</td>
<td>.00001</td>
<td>.00008</td>
<td>.027</td>
<td>.064</td>
<td>.125</td>
<td>.216</td>
</tr>
<tr>
<td></td>
<td>second</td>
<td>second</td>
<td>second</td>
<td>second</td>
<td>second</td>
<td>second</td>
</tr>
<tr>
<td>$n^5$</td>
<td>.1</td>
<td>3.2</td>
<td>24.3</td>
<td>1.7</td>
<td>5.2</td>
<td>13.0</td>
</tr>
<tr>
<td></td>
<td>second</td>
<td>seconds</td>
<td>seconds</td>
<td>minute</td>
<td>minutes</td>
<td>minutes</td>
</tr>
<tr>
<td>$2^n$</td>
<td>.001</td>
<td>1.0</td>
<td>17.9</td>
<td>12.7</td>
<td>35.7</td>
<td>366</td>
</tr>
<tr>
<td></td>
<td>second</td>
<td>second</td>
<td>minutes</td>
<td>days</td>
<td>years</td>
<td>centuries</td>
</tr>
<tr>
<td>$3^n$</td>
<td>.059</td>
<td>58</td>
<td>6.5</td>
<td>3855</td>
<td>$2 \cdot 10^8$</td>
<td>$1.3 \cdot 10^{13}$</td>
</tr>
<tr>
<td></td>
<td>second</td>
<td>minutes</td>
<td>years</td>
<td>centuries</td>
<td>centuries</td>
<td>centuries</td>
</tr>
</tbody>
</table>
Effect of Speed-Ups

Let’s wait for faster hardware! Consider maximum problem size you can solve in an hour.

<table>
<thead>
<tr>
<th></th>
<th>present</th>
<th>100 times faster</th>
<th>1000 times faster</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>$N_1$</td>
<td>$100N_1$</td>
<td>$1000N_1$</td>
</tr>
<tr>
<td>$n^2$</td>
<td>$N_2$</td>
<td>$10N_2$</td>
<td>$31.6N_2$</td>
</tr>
<tr>
<td>$n^3$</td>
<td>$N_3$</td>
<td>$4.64N_3$</td>
<td>$10N_3$</td>
</tr>
<tr>
<td>$n^5$</td>
<td>$N_4$</td>
<td>$2.5N_4$</td>
<td>$3.98N_4$</td>
</tr>
<tr>
<td>$2^n$</td>
<td>$N_5$</td>
<td>$N_5 + 6.64$</td>
<td>$N_5 + 9.97$</td>
</tr>
<tr>
<td>$3^n$</td>
<td>$N_6$</td>
<td>$N_6 + 4.19$</td>
<td>$N_6 + 6.29$</td>
</tr>
</tbody>
</table>
NP-Completeness / NP-Hardness

Your boss says:

“Get me an efficient traveling-salesman algorithm, or else...”

What are you going to do?
“Yes Ma’am, expect it this afternoon!”

Problem is

- All known algorithms (essentially) check all possible paths.
- Exhaustive checking is exponential.
- Good luck!
Response

“Hah! I will prove that no such algorithm is possible”

Problem is, proving intractability is very hard.
Many important problems have
- no known tractable algorithms
- no known proof of intractability.
“I can’t find an efficient algorithm.
I guess I’m just a pathetic loser. ”

Bad for job security.
Response

“The problem is NP-hard. I can’t find an efficient algorithm, but neither can any of these famous people . . .”

Advantage is:

- The problem is “just as hard” as other problems smart people can’t solve efficiently.
- So it would do your boss no good to fire you and hire a Technion/Hebrew Univ./MIT graduate.
Response

“Would you settle for a pretty good, but not the best, algorithm?”

Intractability isn’t everything.

- Find an approximate solution (is a solution within 10% of optimum good enough, ma’am?).
- Use randomization.
- Fixed parameter algorithms may be applicable.
- Heuristics can also help.

- Approximation, randomization, etc. are among the hottest areas in complexity theory and algorithmic research today.
Next Subject
A Very Short Math Review

- Graphs
- Strings and languages
- Mathematical proofs
- Mathematical notations (sets, sequences, . . . ) ✓
- Functions and predicates ✓

✓ = will be done in recitation.
Graphs

\[ G = (V, E), \text{ where} \]
\[ V \text{ is set of nodes or vertices, and} \]
\[ E \text{ is set of edges} \]
\[ \text{degree of a vertex is number of edges} \]
Directed Graph and its Adjacency Matrix

\[
\begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]
Strings and Languages

- **an alphabet** is a finite set of **symbols**
- a **string** over an alphabet is a finite sequence of symbols from that alphabet.
- the **length** of a string is the number of symbols
- the **empty string** \( \varepsilon \)
- **reverse**: \( abcd \) reversed is \( dcba \).
- **substring**: \( xyz \) in \( xyzzy \).
- **concatenation** of \( xyz \) and \( zy \) is \( xyzzy \).
- \( x^k \) is \( x \cdots x \), \( k \) times.
- a **language** \( L \) is a set of strings.
Proofs

We will use the following basic kinds of proofs.

- by construction
- by contradiction
- by induction
- by reduction
- we will often mix them.
Proof by Construction

A graph is \( k \)-regular if every node has degree \( k \).

**Theorem:** For every even \( n > 2 \), there exists a 3-regular graph with \( n \) nodes.
Proof by Construction

Proof: Construct $G = (V, E)$, where $V = \{0, 1, \ldots, n - 1\}$ and

$$E = \{\{i, i + 1\} \mid \text{for } 0 \leq i \leq n - 2\} \cup \{n - 1, 0\} \cup \{\{i, i + n/2\} \mid \text{for } 0 \leq i \leq n/2 - 1\}.$$ 

Note: A picture is helpful, but it is not a proof!
Proof by Contradiction

**Theorem:** $\sqrt{2}$ is irrational.

**Proof:** Suppose not. Then $\sqrt{2} = \frac{m}{n}$, where $m$ and $n$ are relatively prime.

\[
\begin{align*}
  n\sqrt{2} &= m \\
  2n^2 &= m^2
\end{align*}
\]
Proof by Contradiction (cont.)

So \( m^2 \) is even, and so is \( m = 2k \).

\[
\begin{align*}
2n^2 & = (2k)^2 \\
& = 4k^2 \\
& = 2k^2
\end{align*}
\]

Thus \( n^2 \) is even, and so is \( n \).

Therefore both \( m \) and \( n \) are even, and not relatively prime!

*Reductio ad absurdum.*
Proof by Induction

Prove properties of elements of an infinite set.

\[ \mathcal{N} = \{1, 2, 3, \ldots\} \]

To prove that \( \varphi \) holds for each element, show:

- **base step**: show that \( \varphi(1) \) is true.
- **induction step**: show that if \( \varphi(i) \) is true (the induction hypothesis), then so is \( \varphi(i + 1) \).
Induction Example

**Theorem:** All cows are the same color.

**Base step:** A single-cow set is definitely the same color.

**Induction Step:** Assume all sets of $i$ cows are the same color. Divide the set $\{1, \ldots, i + 1\}$ into $U = \{1, \ldots, i\}$, and $V = \{2, \ldots, i + 1\}$.

All cows in $U$ are the same color by the induction hypothesis.

All cows in $V$ are the same color by the induction hypothesis.

All cows in $U \cap V$ are the same color by the induction hypothesis.
Induction Example (cont.)

Ergo, all cows are the same color.

*Quod Erat Demonstrandum* (QED).

(cows’ images courtesy of [www.crawforddirect.com/cows.htm](http://www.crawforddirect.com/cows.htm))
Proof by Reduction

We can sometime solve problem A by reducing it to problem B, whose solution we already know.

Example: Maximal matching in bipartite graphs:
Proof by Reduction

Reducing bipartite matching to MAX FLOW:

Reduction: Put capacity 1 on each edge. Maximum flow corresponds to maximum matching. So if we have an algorithm that produces max flow, we can easily derive a maximum bipartite matching from it.