Computational Complexity Analysis: A Gentle Introduction

Todd Wareham

March 7, 2016

Computational Complexity Analysis: Why Bother?

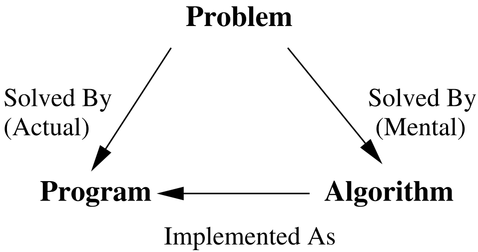
- Computations solve problems.
- Some computations fast, *e.g.*, Google search.
- Some seem hard, *e.g.*, getting good class schedules, or we hope they are, *e.g.*, cracking encrypted communications.

HOW DO WE SOLVE PROBLEMS QUICKLY?

HOW DO WE SHOW PROBLEMS ARE HARD?

HOW DO WE DEAL WITH HARD PROBLEMS?

Problems, Algorithms, and Programs



Problem: A set of inputs and their associated outputs.

Algorithm: A sequence of instructions that solves a problem, *i.e.*, computes the output for a given input.

Program: A sequence of instructions *in some computer language* that solves a problem.

Finding the Area of a Circle

Problem:

Input: A radius r.
Output: The area of a circle with radius r.

Algorithm:

area = 3.14159 * r * r
print area

Program:

```
import sys
r = sys.argv[1]
area = 3.14159 * r * r
print area
```

Summing a List

Problem:

Input: A list L of n numbers.
Output: The sum of the numbers in L.

Algorithm:

```
sum = 0
for i = 1 to n do
    sum = sum + L[i]
print sum
```

Program:

```
sum = 0
for i in range(1, n + 1):
      sum = sum + L[i]
print sum
```

Running Time Magnitudes

$\log n$	Logarithmic Time	(Binary Search)
n	Linear Time	(Summing a List)
n^2	Quadratic Time	(List Sort)
2^n	Exponential Time	(Bin Packing)

Polynomial Time $= n^c$ time for constant c

Table of Doom (1 Gigaflop/s Version)

Input Size (n)	Running Time				
	$\log_2 n$	n	n^2	n^3	2^n
10	< 1 second	< 1 second	< 1 second	< 1 second	< 1 second
50	< 1 second	< 1 second	< 1 second	< 1 second	13 days
100	< 1 second	< 1 second	< 1 second	< 1 second	4×10^{13} years
1000	< 1 second	< 1 second	< 1 second	1 second	4×10^{284} years
one million	< 1 second	< 1 second	2 minutes	30 years	–
300 million	< 1 second	< 1 second	10 days	9×10^5 years	–
five billion	< 1 second	5 seconds	8 centuries	4×10^{12} years	–

The Crux of the Matter

- Some problems are solvable in polynomial time, *e.g.*, summing a list, and can be solved in practice for large input sizes; some, *e.g.*, bin packing, cannot.
- With problems that are not known to be solvable in polynomial time, have we just not thought of a good algorithm yet, or are they genuinely intractable?

HOW CAN WE PROVE INTRACTABILITY?

Foundations of Complexity Analysis: Arm Wrestling

Arnold

Betty

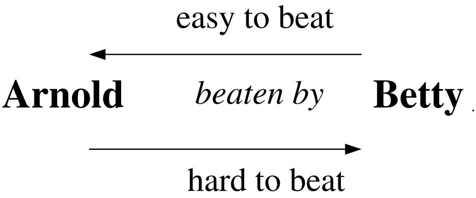
Best in Two?
The Logic of Pairwise Comparison

Arnold *beaten by* **Betty** .

- Establish better arm wrestler by a two-person match.
- If Arnold is beaten by Betty:

…??? …

Best in Two?
The Logic of Pairwise Comparison (Cont'd)



- Establish better arm wrestler by a two-person match.
- If Arnold is beaten by Betty:
    1. Arnold is no better than Betty
       (if Betty is easy to beat then Arnold is easy to beat)
    2. Betty is at least as good as Arnold
       (if Arnold is hard to beat then Betty is hard to beat)

Foundations of Complexity Analysis
Reductions between Problems

- A **reduction** from problem $\mathcal{A}$ to problem $\mathcal{B}$ ($\mathcal{A}$ **reduces to** $\mathcal{B}$) is an algorithm for solving $\mathcal{A}$ that uses an algorithm for solving $\mathcal{B}$.

Algorithm solveA:

```
blah blah
blah blah blah
blah blah
  ........

  x = solveB(x, y, z)
  .........
 return answer
```

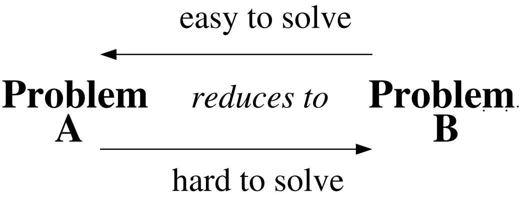
Hardest in Two?
The Logic of Reducibility

Problem A *reduces to* **Problem B**.

- Establish harder problem by poly-time reduction.
- If problem $\mathcal{A}$ reduces to problem $\mathcal{B}$:

… ??? …

Hardest in Two?
The Logic of Reducibility (Cont'd)



- Establish harder problem by poly-time reduction.
- If problem $\mathcal{A}$ reduces to problem $\mathcal{B}$:
  1. $\mathcal{A}$ is no harder than $\mathcal{B}$
     (if $\mathcal{B}$ is easy to solve then $\mathcal{A}$ is easy to solve)
  2. $\mathcal{B}$ is at least as hard as $\mathcal{A}$
     (if $\mathcal{A}$ is hard to solve then $\mathcal{B}$ is hard to solve)

Dealing with Intractability

- First poly-time intractable problem proven in 1971; thousands proven since (including Bin Packing and many other industrially-important problems).

…but we still need to solve these problems!!!

HOW DO WE SOLVE INTRACTABLE PROBLEMS?

Tractability under Restrictions: Fixed-Parameter Tractability

- Let's relax our notion of tractability:
  1. Focus on a set P of one or more problem-aspects (**parameters**) whose values are small in practice.
  2. Only consider inputs with small values for P.
  3. Relax poly-time to fixed-parameter (fp-)time, *i.e.*, run-time $f(P)n^c$ for some function f.
- When the parameters in P are small, fp-time is effectively poly-time, *e.g.*, when $P = \{k\}$ and $k = 3$,

$$2^k n^2 \Rightarrow 2^3 n^2 \Rightarrow 8n^2 \Rightarrow \sim n^2$$

- Can prove fp-intractability with appropriate reductions.

Computational Complexity Analysis: The *Reader's Digest* Version

	good	bad
classical complexity	poly-time solvable (Best)	pt-intractable
parameterized complexity	fp-tractable (Still OK)	fp-intractable

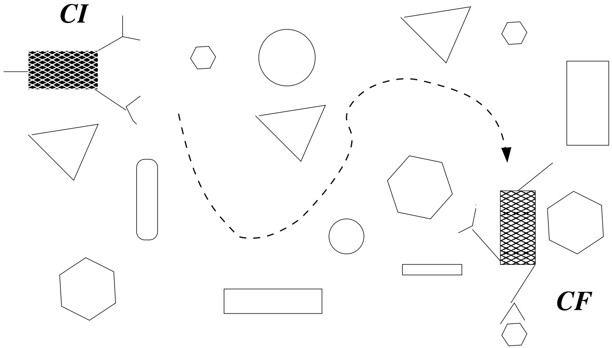
Complexity Analysis of Important Problems

> **The Tractable Computation Thesis**:
> WHERE POSSIBLE, IMPORTANT PROBLEMS SHOULD BE SOLVED QUICKLY.

- Two conceptions of "quickly":
  - quick in general (poly-time solvability)
  - quick under restrictions (fp-tractability relative to P)
- If a problem is intractable, look for restrictions to make it tractable.
- One way to do this is to look for parameters whose values are small in practice and then see if these restrictions yield fp-tractability.

Robot Motion Planning

- Consider 3D motion planning in an obstacle-filled environment where we have to totally plan out a collisionless path from some initial robot-configuration c_I to a final robot-configuration c_F, *e.g.*,



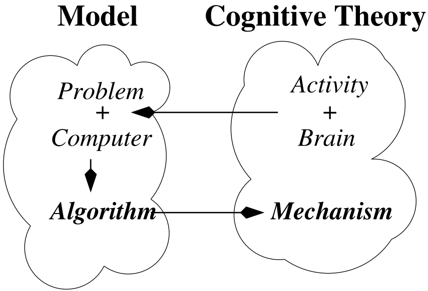
Robot Motion Planning (Cont'd)

3D ROBOT MOTION PLANNING
Input: An environment E with obstacles, a robot R, and initial and final configurations CI and CF of R in E.
Output: A sequence of moves of R from c_I to c_F in E that does not collide with an obstacle, if such a sequence exists, and special symbol $\perp$ otherwise.

- Is poly-time intractable in general; however, robots often have a small number k of joints (3 for robot arm, $\leq$ 20 for robot hand).
- Unfortunately, is fp-intractable for parameter-set $\{k, X\}$, where X is **lots** of other problem-aspects (Cesati and Wareham, 1995).

Computational Models of Cognition

- Goal is to develop theories of cognitive activities stated in terms of models, problems, and algorithms.



- Each cognitive theory has an associated model whose computations can be stated as a problem.

Complexity Analysis of Cognitive Theories

The Tractable Cognition Thesis:
AS COGNITION IS FAST, PROBLEMS ASSOCIATED WITH COGNITIVE MODELS SHOULD BE SOLVABLE QUICKLY.

- Two conceptions of “quickly”:
  - quick in general (poly-time solvability)
  - quick under restrictions (fp-tractability relative to P)
- If the problem associated with a model is intractable, revise mechanisms in model to make it tractable.
- One way to do this is to look for restrictions that yield fp-tractability, and then see if these restrictions hold in actual cognition.

Analogy Derivation

- Given two concepts, an analogy is essentially a mapping between common parts of both concepts.
- Analogies can be good, *e.g.*, “Genghis Khan is like Adolf Hitler”, or bad, *e.g.*, “An orange is like Adolf Hitler”.
- Analogy derivation underlies many cognitive processes, *e.g.*, memory retrieval, problem solving, learning.
- Sometimes, deriving analogies is easy; sometimes, it is hard. What characterizes these situations?

Analogy Derivation (Cont'd)

ANALOGY MAPPING **Input**:Two concepts B and T. **Output**: The best analogy between B and T.

- Is poly-time intractable in general; however, various conjectures have been made about what restrictions do and do not make this problem easy, *e.g.*, fp-tractable.
- **All** published conjectures have been proven wrong (van Rooij et al, 2008)!
- Lots of work remains to be done ...

Computational Complexity Analysis: What Next?

- New application areas for CCA, *e.g.*, the design and reconfiguration of robot swarms (Wareham (2015)) and software systems (Wareham and Sweers (2015)).
- New ways of designing efficient algorithms.
- New conceptions of tractability (and hence new methods for assessing such (in)tractability).
- New ways of resolving conjectures underlying intractability, *e.g.*, $P = NP$?

. . . Lots of work remains to be done . . .
(Thank goodness!)