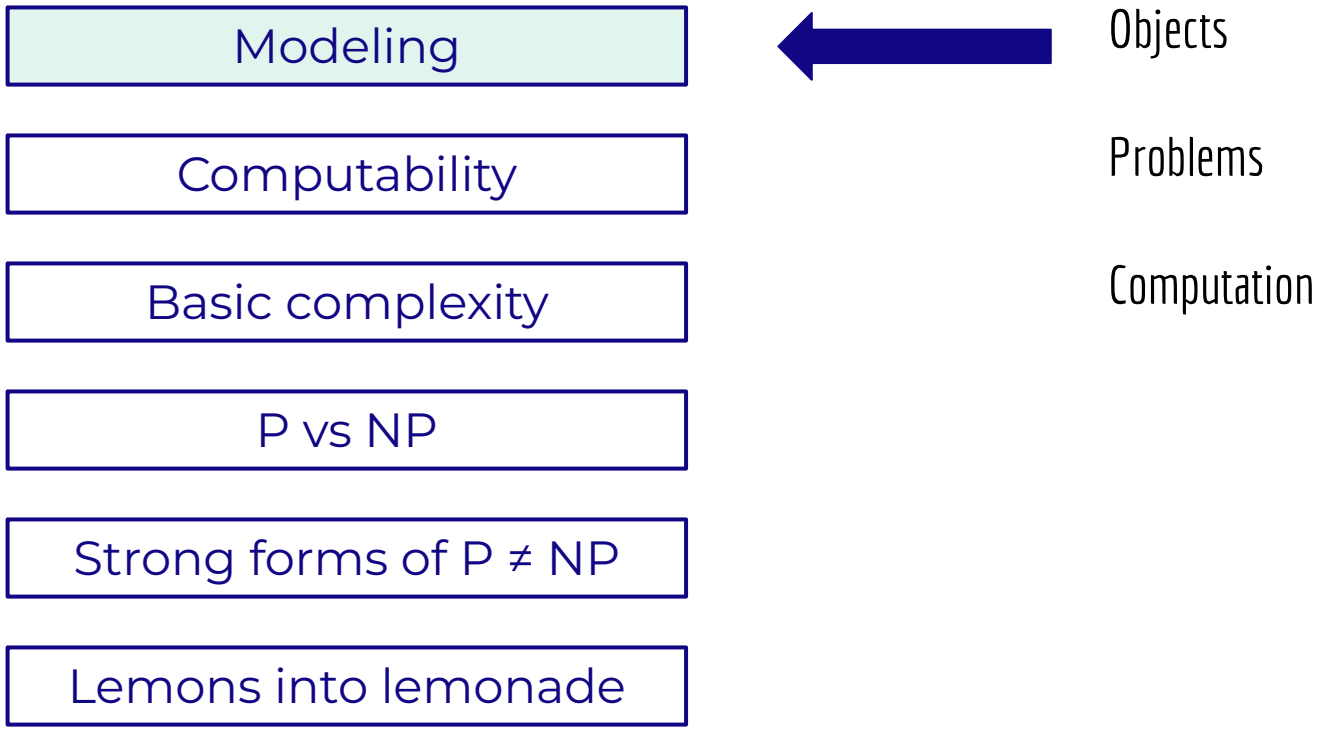
Complexity Theory

Instructor: Roei Tell

Modeling

Course outline



Representing objects

- We will use **Binary Strings** to represent everything
- Examples:

  Integer, matrix, graph

Representing objects

- Why binary?

  Choose the simplest way that is general!

Representing objects

- How many objects can we represents with n symbols from $\sum$?

$\lvert\sum\rvert = 1$	$\Rightarrow$	one	object
$\lvert\sum\rvert = 2$	$\Rightarrow$	2^n	objects
...			
$\lvert\sum\rvert = k$	$\Rightarrow$	k^n	objects

- Big gap between unary and binary, but not a big gap between binary and ternary, quaternary, and so on

Representing objects

- Observation:

  For $k \geq 2$, there is an injection from $[k]^n$ to $\{0,1\}^{\lceil \log_2(k) \rceil \cdot n}$

- Corollary:

  We can represent n-length objects over $\sum$ using O(n) bits

  (the constant hidden inside the O-notation is $\log(|\sum|)$)

- (later in the course: this does not affect efficient computation)

Representing problems

- Def:

  A **computational problem** is a function $\{0,1\}^* \to \{0,1\}^*$

Representing problems

- Examples:

1. f(G) = size of a largest clique in the graph G
2. f(M) = the determinant of the matrix M
3. f(x,y) = the sum of the integers x and y
4. f(S) = the best-fitting curve to explain the dataset S

...

Representing problems

- Def (simplest case):

  A **decision problem** is a function $\{0,1\}^* \rightarrow \{0,1\}$

  (i.e., with one output bit)

Representing problems

- Examples:

1. f(G) = 1 iff G has a Eulerian cycle
2. f(M) = 1 iff the matrix M is invertible
3. f(x) = 1 iff x is prime
4. f(S) = 1 iff there's a quadratic fitting all points in S

...

Representing problems

- Our focus will usually be decision problems

  ⇒ simplest case

  ⇒ “general enough”: questions about search often reduce to questions about decision (e.g., P vs NP)

Representing problems

- Def (simplest case):

  A **language** is a subset of $\{0,1\}^*$

Representing problems

- Thm:

  There is a bijection between languages and decision problems

Representing problems

- Thm:

  There is a bijection between languages and decision problems

- Pf:

  Map $L \subseteq \{0,1\}^*$ to $f_L:\{0,1\}^* \to \{0,1\}$ st $f_L(x)=1 \Leftrightarrow x \in L$.

  Exercise: show that this is a bijection!

Representing problems

- Computing a decision problem f_L

  $\Leftrightarrow$

  “Deciding” the language L

Representing problems

- Exercise:

  Take all the examples of decision problems, and present them as languages.

Representing problems

- More general representation (capturing more problems)
- Def:

  A **search problem** is a relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$.

  For every x we define $R_x = \{ y : (x,y) \in R \}$ as the set of solutions for x.
- The problem is: Given input x, output some solution $y \in R_x$

Representing problems

- Search problems generalize functions:
  - There can be inputs x with multiple valid solutions, i.e. $|R_x| > 1$
  - There can also be inputs x with no solutions, i.e. $R_x = \emptyset$

Representing problems

- Examples:

  1. Given an integer, output a factor (or “NONE” if it’s prime)
  2. Given an n-vertex graph, output a clique of size ≥ n/10 (or “NONE”)
  3. Given a system of linear equations, output a solution (or “NONE”)

  …

Representing computation

- Intuition:
  - Process that changes an environment
  - Applies a sequence of simple actions
  - Each action interacts with a small part of the environment

Representing computation

- Intuition:
  - Process that changes an environment
  - Applies a sequence of simple actions
  - Each action interacts with a small part of the environment
- Contrast “big environment” with “simple actions, small part”

Representing computation

- A convenient model:

  Turing Machine (Alan Turing, 1930s)

- Predates computers
- General & simple to analyze

Representing computation

- Definition 1:

  A Turing Machine is $(\sum, S, \delta)$:

  - **Alphabet symbols:** A finite set $\sum \supseteq \{0,1\}$, a special $\beta \notin \sum$
  - **States:** A finite set S, with special $s_{start}, s_{end} \in S$
  - **Transition function:** $\delta : S \times (\sum \cup \{ \beta \}) \rightarrow S \times (\sum \cup \{ \beta \}) \times \{ -1, 1 \}$

Representing computation

- Definition 2:

  An instantaneous configuration of a TM is (w,h,s):

  - **Worktape content:** string $w \in (\sum \cup \{ \beta \})^*$
  - **Head location:** integer $h \geq 1$
  - **Current state:** state $s \in S$

Representing computation

› Definition 3:

The computation of a TM (∑, S, δ) on input x is a sequence of **configurations**. The first configuration has:

worktape content: $w^{(1)} = x$

head location: $h^{(1)} = 1$

state: $s^{(1)} = s_{start}$

Representing computation

› Definition 3 (cont'd):

To go from configuration $(w^{(i)},h^{(i)},s^{(i)})$ to the $(i+1)^{th}$ configuration:

Let $(s^{(i+1)}, b, d) \leftarrow \delta(s^{(i)}, w^{(i)}_{h\_i})$ // if $h_i > |w^{(i)}|$ then $w^{(i)}_{h\_i} = \beta$

Let $w^{(i+1)}$ be $w^{(i)}$ with the symbol at location h_i replaced by b

Let $h^{(i+1)} = \max\{ 1, h^{(i)} + d \}$

If $s^{(i+1)} = s_{end}$, halt and **output $w^{(i+1)}$**

Representing computation

› Definition 4:

The running time of a Turing Machine M on input x is the length of the computation sequence of M(x).

The running time of M is a function

$$T(n) = \max_{x \in \{0,1\}^n} \{ \text{runtime of } M(x) \}$$

Representing computation

› Definition 4:

The running time of a Turing Machine M on input x is the length of the computation sequence of M(x).

The running time of M is a function

$$T(n) = \max_{x \in \{0,1\}^n} \{ \text{run}$$

this is the time complexity of a specific algorithm M, not the “complexity of a problem”

Representing computation

- Church-Turing thesis:

  A function is computable by any process if and only if it is computable by a Turing Machine.

Representing computation

- Cobham–Edmonds thesis:

  A function is **efficiently** computable by any process if and only if it is **efficiently** computable by a Turing Machine.

Representing computation

› Cobham–Edmonds thesis:

A function is **efficiently** computable by any process if and only if it is **efficiently** computable by a Turing Machine.

"efficiently" = polynomial time, more on that later

Representing computation

- Cobham–Edmonds thesis:

  A function is **efficiently** computable by any process if and only if it is **efficiently** computable by a Turing Machine.

- Is it possible to mathematically prove this?

Representing computation

- Cobham–Edmonds thesis:

  A function is **efficiently** computable by any process if and only if it is **efficiently** computable by a Turing Machine.

- Is it possible to mathematically prove this?

  This is a falsifiable scientific hypothesis.

Representing computation

- Example 1:

  Computing f(x) = x+1 in linear time.

Representing computation

- Example 2:

  Deciding $L = \{ w,w : w \in \{0,1\}^* \}$ in quadratic time.

- Are we allowed the alphabet symbol “,” in the input?

Representing computation

- The “criss-cross” technique feels artificial
- Must we use ad-hoc techniques each time to be convinced that TMs are a general model of computation?

Variations of Turing Machines

- What we'll see next:

  Natural variations on TMs (making them seem like computers we know) don't add significant computational power.

- Sanity check for the Cobham–Edmonds Thesis

Variations of Turing Machines

- Variations:

1. Two-way infinite tape, multidimensional tape
2. k-tape machines
3. RAM machines
4. k-head machines
5. Clocked machines, head-aware machines

Toy example

- Define “lazy machines”, wherein at each step the head can go left, right, or stay put.
- Thm:

  Any lazy machine running in time T(n) can be simulated by a (standard, non-lazy) machine running in time O(T(n)).

[1] M’ simulates M if for all x we have M(x) = M’(x).

Two-tape machines

- Definition 1a:

  A two-tape Turing Machine is

  - **Alphabet symbols:** A finite set $\sum \supseteq \{0,1\}$, a special $\beta \notin \sum$
  - **States:** A finite set S, with special $s_{start}, s_{end} \in S$
  - **Transition function:** $\delta : S \times (\sum \cup \{ \beta \})^2 \rightarrow S \times (\sum \cup \{ \beta \})^2 \times \{-1, 1\}^2$

Two-tape machines

› Definition 2a:

A configuration of a 2-tape TM is

Worktape content:	two strings	$w_1, w_2 \in (\sum \cup \{ \beta \})^*$
Head locations:	two integers	$h_1, h_2 \geq 1$
Current state:	state	$s \in S$

Two-tape machines

› Definition 3a:

The computation of a 2-tape TM (∑, S, δ) on input x is a sequence of **configurations**. The first configuration has:

worktapes content: $w^{(1)}_1 = x, \quad w^{(1)}_2 = \beta$

head locations: $h^{(1)}_1 = 1, \quad h^{(1)}_2 = 1$

state: $s^{(1)} = s_{start}$

Two-tape machines

› Definition 3a (cont'd):

To go from configuration i to configuration i+1:

Let $(s^{(i+1)}, (b_1,b_2), d) \leftarrow \delta(s^{(i)}, ((w^{(i)}_1)_{h\wedge(i)\_1}, (w^{(i)}_2)_{h\wedge(i)\_2})$

Let $w^{(i+1)}_\sigma$ be $w^{(i)}_\sigma$ with the symbol at location $h^{(i)}_\sigma$ replaced by b

Let $h^{(i+1)} = \min\{ 1, h^{(i)} + d \}$

If $s^{(i+1)} = s_{end}$, halt and **output $w^{(i+1)}$**

Two-tape machines

- Thm:

  Any two-tape machine running in time T(n) can be simulated by a (standard, one-tape) machine running in time poly(T(n)).

Two-tape machines

- Proof idea.
- We will often use this type of high-level description, make sure you know how to formalize this as a TM.

k-tape machines

- The argument extends to any constant integer k .
- Thm:

  For any constant k, any k-tape machine running in time T(n) can be simulated by a (standard) machine running in time poly(T(n)).

RAM machines

- Random access:

  Special “address” tape

  Special state s_{jump} that teleports the head to the address

RAM machines

- Thm:

  Any RAM machine running in time T(n) can be simulated by a standard machine running in time poly(T(n)).

- Proof idea.

- Important point from the proof: Composition of poly is poly.

Representing efficient computation

- Cobham–Edmonds thesis (formal version):

  A function is computable by any process in polynomial time if and only if it is computable in polynomial time by a Turing Machine.

Representing efficient computation

- Why focus on polynomial time?
  - insensitive to reasonable variations in modeling
  - closed under composition (A(x) efficient ⇒ A(B(x)) efficient)
  - ruling out polytime ⇒ ruling out all fast algorithms
  - first-order approximation of “efficient computation”
    (we also care about the precise poly!)