CMSC 28100

Introduction to Complexity Theory

Spring 2024
Instructor: William Hoza

Midterm exam

- Midterm exam will be in class on **Wednesday, April 17**
- To prepare for the midterm, you only need to study the material up to this point
- The midterm will be about decidability and undecidability

Which problems

can be solved

through computation?

Decompositions into squares

- Let $\mathrm{SQUARES} = \{xx : x \in \{0,1\}^*\}$
- We say that a string $w \in \{0,1\}^*$ can be decomposed into squares if there exist $y_1, y_2, \dots, y_k \in \mathrm{SQUARES}$ such that $w = y_1 y_2 \dots y_k$
- Example: $00100111 = (001\,001)(1\,1)$
- Example: 1001 cannot be decomposed into squares

Decompositions into squares

- Let $\mathrm{DECOMPOSABLE\text{-}INTO\text{-}SQUARES} = \{w \in \{0,1\}^* : w \text{ can be decomposed into squares}\}$

Is $\mathrm{DECOMPOSABLE\text{-}INTO\text{-}SQUARES}$ **decidable?**

A: Yes

B: No

C: It depends on the encoding

D: It's not a language, so the question doesn't make sense

Respond at PollEv.com/whoza or text "whoza" to 22333

Decompositions into squares

- Let $\text{DECOMPOSABLE-INTO-SQUARES} = \{w \in \{0,1\}^* : w \text{ can be decomposed into squares}\}$
- **Claim:** $\text{DECOMPOSABLE-INTO-SQUARES}$ is decidable
- **Proof sketch:** Given $w \in \{0,1\}^*$, try all possible decompositions of w into substrings: $w = y_1 y_2 \dots y_k$
- Check whether $y_i \in \text{SQUARES}$ for each i
- If we find a decomposition into squares, accept; otherwise, reject.

Decompositions into squares

- $\mathrm{DECOMPOSABLE\text{-}INTO\text{-}SQUARES}$ is decidable
- So… Can we actually decide it?

Our algorithm is so slow that it’s worthless

- Can the following string be decomposed into squares?

0011001100101001100000001010011000000111111101111111101011010101110100

100100111010010010011010101011111110001011111110001010011010100110101

- Checking all possible decompositions would take longer than a lifetime
- One begins to feel that DECOMPOSABLE-INTO-SQUARES might as well be undecidable…

Which problems

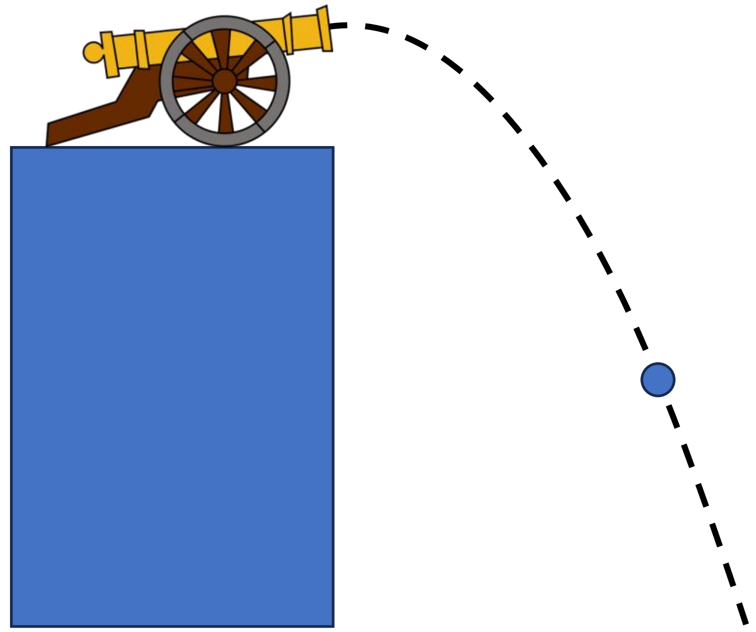
can be solved

through computation?

Refining our model

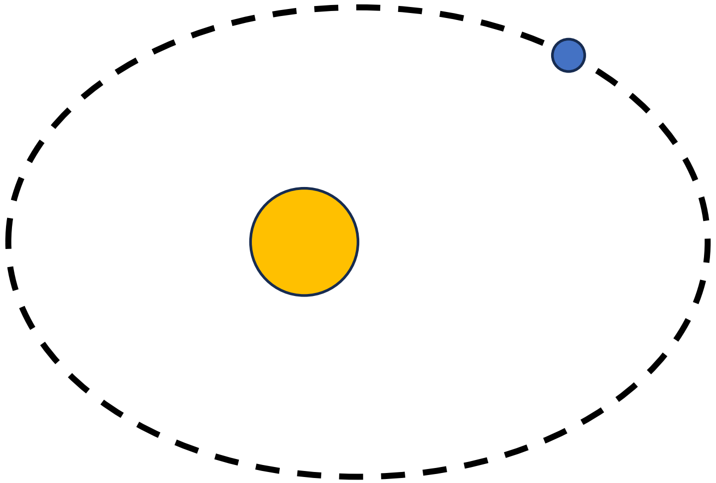
- The mathematical model we have studied so far: Decidable vs. undecidable
- Now we will refine our model to take into account the fact that in real life, we only have a limited amount of time (and other resources)
- “Complexity theory”

Analogy: Gravity



- In an introductory physics class, we might model gravity as a constant downward force of 9.8 N/kg
- In a more advanced physics class, we might use a more sophisticated model of gravity:

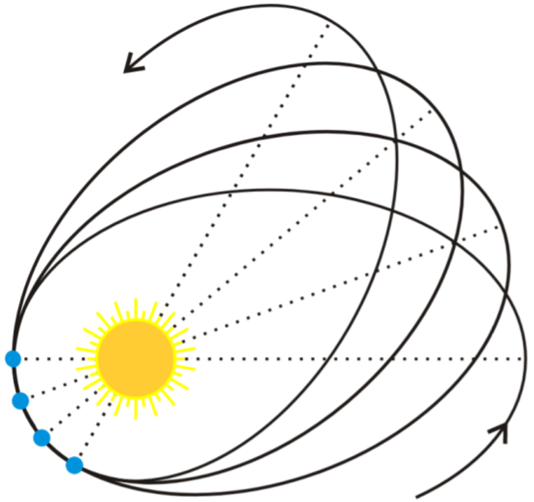
$$F = G \cdot \frac{m_1 \cdot m_2}{r^2}$$



Theory vs. practice

- Disclaimer: Our theoretical model will still not be perfectly accurate!
- Sometimes, we might categorize a problem as “tractable” even though it is not actually “solvable in practice”
- Other times, we might categorize a problem as “intractable” even though it is actually “solvable in practice”

Theory vs. practice



- Physics analogy: Newton's universal law of gravitation ($F = G \cdot \frac{m_1 \cdot m_2}{r^2}$) is a fantastic approximation in many cases, but it does not correctly predict Mercury's motion around the sun!
- "…all models are wrong, but some are useful." –George Box
- Even though our model of tractability will not be completely accurate, it will still give us real insights into the nature of computation

Time complexity

- Let M be a Turing machine with input alphabet Σ
- The time complexity of M is a function $T_M : \mathbb{N} \to \mathbb{N}$ defined as follows:

$$T_M(n) = \max_{w \in \Sigma^n}(\text{running time of } M \text{ on } w)$$

- We are focusing on the worst-case n-symbol input

Deciding a language in time T

- Let L be a language and let $T: \mathbb{N} \to \mathbb{N}$ be a function
- **Definition:** We say that L can be decided in time T if there exists a Turing machine M such that
  - M decides L, and
  - If we let $T_M: \mathbb{N} \to \mathbb{N}$ be the time complexity of M, then for every n, we have $T_M(n) \leq T(n)$

Scaling behavior

- We will mainly focus on the limiting behavior of $T(n)$ as $n \to \infty$
- How "quickly" does the time complexity $T(n)$ increase when we increase the input length n?

Asymptotic analysis

Asymptotic analysis

- Two possible time complexities:

$$T_1(n) = 3n^2 + 14$$

$$T_2(n) = 2n^2 + 64n + \lceil \sqrt{n} \rceil$$

- When n is large, the leading $C \cdot n^2$ term dominates

- We will ignore the low-order terms and the leading coefficient C

- We focus on the n^2 part (“quadratic time”)

Big-O notation

- $3n^2 + 14$ and $2n^2 + 64n + \lceil\sqrt{n}\rceil$ are both "$O(n^2)$"
- More generally, let $T, f : \mathbb{N} \to \mathbb{N}$ be any two functions
- We say that T is $O(f)$ if there exist $C, n_* \in \mathbb{N}$ such that for every $n > n_*$, we have $T(n) \leq C \cdot f(n)$
- Notation: $T \in O(f)$ or $T \leq O(f)$ or $T = O(f)$

Big-O notation examples

- $3n^2 + 14$ is $O(n^2)$
- $3n^2 + 14$ is $O(n^2 + n)$
- $3n^2 + 14$ is $O(n^3)$
- $3n^2 + 14$ is not $O(n^{1.9})$

Big-Ω and big-Θ

- Let $T, f: \mathbb{N} \to \mathbb{N}$ be any two functions
- We say that T is $\Omega(f)$ if there exist $c \in (0,1)$ and $n_* \in \mathbb{N}$ such that for every $n > n_*$, we have $T(n) \geq c \cdot f(n)$
- We say that T is $\Theta(f)$ if T is $O(f)$ and T is $\Omega(f)$

Big-Ω and big-Θ examples

- $0.1n^2 + 14$ is $\Omega(n^2)$ and $\Omega(n)$, but not $\Omega(n^3)$
- $0.1n^2 + 14$ is $\Theta(n^2)$ and $\Theta(n^2 + 2n^{1.4})$, but not $\Theta(n)$

Let $T(n) = 2^{3n+4}$. Which of the following statements is false?

A: $T(n)$ is $\Omega(2^n)$

B: $T(n)$ is $2^{\Theta(n)}$

C: $T(n)$ is $\Theta(2^{3n})$

D: $T(n)$ is $O(2^n)$

Respond at PollEv.com/whoza or text “whoza” to 22333