CS-E4530

Computational Complexity Theory

Janne H. Korhonen
Aalto University
Spring 2018

Lecture 3:

Representations, Universality, and Undecidable Problems

Problem Representations

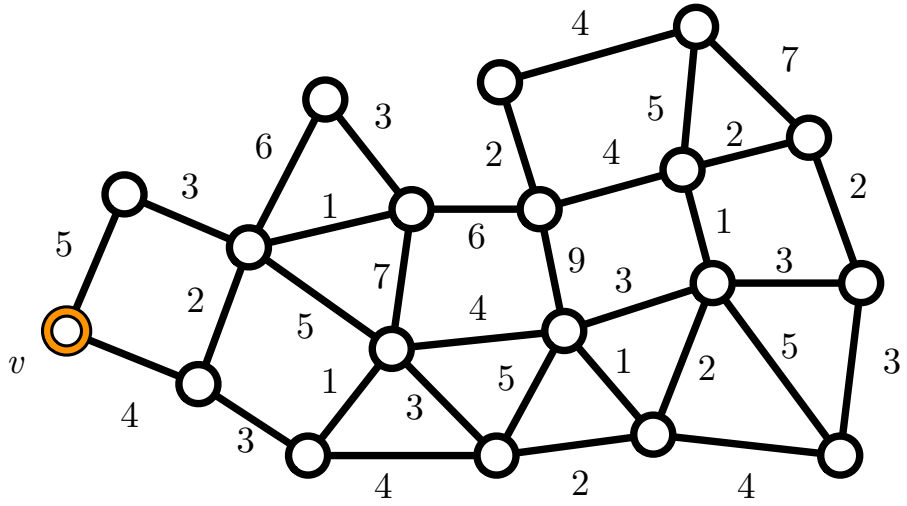
Decision Problems

- **Recall how we defined decision problems:**
  - Decision problem: $L \subseteq \{0,1\}^*$
- **We model all computational task as decision problems:**
  - How to handle **optimisation problems**?
  - How to handle non-binary string inputs, like graphs?

Decision Problems: Example

Travelling Salesman Problem (Decision Version)

- **Instance:** Graph $G = (V, E)$ with positive edge weights, integer $W \geq 0$, a vertex $v \in V$.
- **Question:** Is there there a tour starting from vertex v that visits all other vertices exactly once and then returns to v with weight at most W?



Representations

- **For general inputs:**
  - Encode all inputs as binary
  - Just like we actually do with computers
- **More formally:**
  - Define an encoding function that maps instance x into a binary string $\llcorner x \lrcorner$

Representations: Numbers

- **Number are represented in binary**
  - $\llcorner n \lrcorner$ is the binary representation of n
  - Leading zeroes can be ignored

$$\llcorner 1 \lrcorner = 1$$
$$\llcorner 2 \lrcorner = 10$$
$$\llcorner 3 \lrcorner = 11$$
$$\llcorner 10 \lrcorner = 1010$$
$$\llcorner 1203 \lrcorner = 10010110011$$

Representations: Non-binary strings

- **Encoding strings over non-binary alphabet Γ:**
  - Encode each symbol using $\lceil \log_2 |\Gamma| \rceil$ bits
  - Encode strings by appending the binary representations
- **Example:** $\Gamma = \{a, b, c, d\}$, $\lceil \log_2 |\Gamma| \rceil = 2$

$$\llcorner a \lrcorner = 00 \qquad \llcorner b \lrcorner = 01$$
$$\llcorner c \lrcorner = 10 \qquad \llcorner d \lrcorner = 11$$

$$\llcorner abababcd \lrcorner = 00\,01\,00\,01\,00\,01\,10\,11 = 0001000100011011$$

Representations: Pairs and tuples

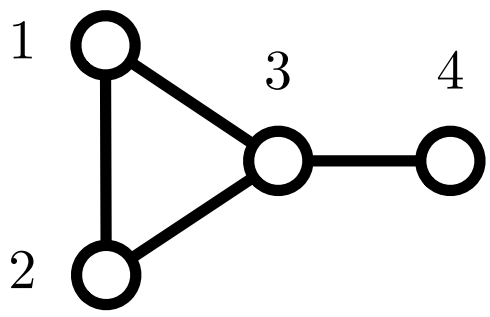
- **Encoding pairs of objects:**
  - Assume we already have a encoding function $\llcorner \cdot \lrcorner$ for objects x and y using alphabet Γ
  - Let # be a symbol not in Γ
  - **Pairs:** encode (x, y) as $\llcorner x \lrcorner \# \llcorner y \lrcorner$
  - **Tuples:** encode $(x_1, x_2, \ldots, x_k)$ as $\llcorner x_1 \lrcorner \# \llcorner x_2 \lrcorner \# \cdots \# \llcorner x_k \lrcorner$
  - Encode the resulting string in binary
- Apply recursively for nested pairs and tuples

Representations: Graphs

- **Convenient to assume:** vertex set is $V = \{1, 2, \ldots, n\}$
- **Two common encoding schemes for graphs:**
  - **Adjacency lists**
  - **Adjacency matrices**

Representations: Adjacency Lists

- **Adjacency list representation:**
  - For each v, list the neighbours of v
  - List all the adjacency lists
  - Encode using the tuple encoding



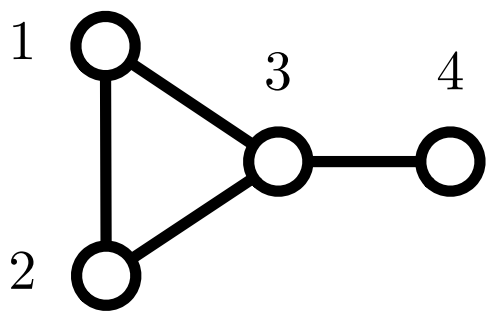
$$(1, (2, 3)),$$
$$(2, (1, 3)),$$
$$(3, (1, 2, 4)),$$
$$(4, (3))$$

Representations: Adjacency Matrices

- **The adjacency matrix representation of $G = (V, E)$:**
  - Matrix M_G such that

$$M_G(v, u) = \begin{cases} 1 & \text{if } v \neq u \text{ and } v \text{ and } u \text{ are adjacent, and} \\ 0 & \text{otherwise.} \end{cases}$$

- **Encode the matrix as a string:**
  - Example: $\llcorner G \lrcorner = 0110\#1010\#1101\#0010$



$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Adjacency Lists vs. Adjacency Matrices

- **Graph $G = (V, E)$ with n vertices and m edges**
  - **Adjacency list encoding:** $O(n + m \log n)$ bits
  - **Adjacency matrix encoding:** $O(n^2)$ bits
  - Possible to do encoding with $O(m \log n)$ bits

- **Representations can be extended to handle directed graphs and weighted graphs**

- **Equivalent in terms of polynomial-time algorithms**
  - Can convert from one to the others in polynomial time
  - However, can matter in other settings for **sparse graphs** (meaning $m = o(n^2)$)

Representations in Practice

- **We assume that representations are 'reasonable':**
  - Encoding is injective
  - Conversion between two reasonable representations can be done in polynomial time
  - We can decide in polynomial time if a given string $x \in \{0,1\}^*$ represents a valid object
- **We assume encoding happens in the background:**
  - We don't distinguish between the input and its encoding
  - For non-encoding strings, output 0

Decision Problems: Example

Travelling Salesman Problem (Decision Version)

- **Instance:** Graph $G = (V, E)$ with positive edge weights, a vertex $v \in V$, and an integer $W \geq 0$, .
- **Question:** Is there there a tour starting from vertex v that visits all other vertices exactly once and then returns to v with weight at most W?

- Input is an encoding of a tuple (G, v, W), where G is a weighted graph, v is an integer (i.e. a vertex), and W is an integer
- If the encoding is not valid, output 0
- Otherwise, output is 1 or 0 depending on the instance

Representing Turing Machines

Representations: Turing Machines

- **Turing machines are finite objects, and we can obviously represent them as binary strings**
- **Concretely:**
  - Map the state space and the alphabet to integers
  - Turing machine is a tuple $M = (\Gamma, Q, \delta)$
  - Γ can be interpreted as a tuple of integers
  - Q can be interpreted as a tuple of integers
  - Each entry in δ can be interpreted as a tuple, and δ itself can be interpreted as a tuple
- **Apply encoding for tuples**

Representations: Turing Machines

- **Convenient to tweak the semantics so that we have certain nice properties**
- **Each TM is represented by infinitely many strings**
  - Allow ‘empty symbols’ at the end of the representation
- **Each string represents some Turing machine**
  - Non-valid encodings are mapped to a single TM
  - E.g. a TM that always halts immediately
- **Notation:** M_α is the Turing machine **represented** by $\alpha \in \{0,1\}^*$

Turing Machines as Data

- **Simple, yet important observation**
  - Programs are data
  - We can define computational problems that refer to Turing machines
  - Essentially the same idea that underlines modern computer: a single computer can run all programs

The Universal Turing Machine

Universality

- **Another simple idea:**
  - Use a Turing machine to simulate another Turing machine
- **There is a universal Turing machine:**
  - Input: an encoding of a Turing machine M and a string x
  - Simulates M on input x and produces the same output
  - Moreover, we can make this simulation **efficient**
- **A single Turing machine captures all computation**

Universal Turing Machine

Theorem

There is a Turing machine $\mathcal{U}$ such that for every $\alpha, x \in \{0,1\}^*$,

- *if M_α halts on input x, then $\mathcal{U}\big((\alpha, x)\big) = M_\alpha(x)$, and*
- *if M_α does not halt on input x, then $\mathcal{U}$ does not halt on (α, x).*

Moreover, if M_α halts on input x in T steps, then $\mathcal{U}$ halts on input (α, x) in CT^2 steps, where C is a constant that only depends on M_α.

Universal Turing Machine: Proof Idea

- **Turing machine $\mathcal{U}$ has inputs:**
  - α represents a k-tape TM M_α
  - x is the input for M_α

- **Basic construction for $\mathcal{U}$:**
  - **Simulated input tape:** simulates the input tape of M_α
  - **Machine tape:** stores the representation of M_α
  - **State tape:** stores the current state of M_α
  - **Simulation tape:** simulates **all** worktapes of M_α
  - Output tape of $\mathcal{U}$ simulates the output tape of M_α

Universal Turing Machine: Proof Idea

- **Simulation of the working tapes:**
  - Using the same tricks as last lecture
  - In interleaved positions, store full contents of all working tapes of M_α in binary
  - Use special marking characters to indicate which positions hold the heads of M_α

Universal Turing Machine: Proof Idea

- **Setup:**
  - Copy the representation of M_α and x to the corresponding tapes
  - Set the current state of M_α to starting state

- **Simulation step:**
  - Scan the simulation tape and store the symbols under head to the state tape
  - Scan the representation of M_α to find a transition corresponding to the current configuration of M_α, write down the written symbols and head movements
  - Pass over simulation tape, apply changes

Universal Turing Machine: Proof Idea

- **Time complexity:**
  - Assume M_α runs for T steps on input x
  - Any tape of M_α can have at most T symbols on it
  - Each simulation step takes at most CT steps for some constant C
  - At most T simulation step
  - Total CT^2, C subsumes constant factors from setup

Universal Turing Machine (Strong Version)

Theorem

There is a TM $\mathcal{U}$ such that for every $\alpha, x \in \{0,1\}^*$,

- *if M_α halts on input x, then $\mathcal{U}\big((\alpha, x)\big) = M_\alpha(x)$, and*
- *if M_α does not halt on input x, then $\mathcal{U}$ does not halt on (α, x).*

Moreover, if M_α halts on input x in T steps, then $\mathcal{U}$ halts on input (α, x) in $CT \log T$ steps, where C is a constant that only depends on M_α.

- **Proof:** complicated.

Undecidability

Simple Counting Argument

- **For any language L, is there a Turing machine that accepts or decides L?**
  - $\alpha \mapsto M_\alpha$ is a surjective function
  - Implies that there is a countable number of Turing machines
  - There is an uncountable number of languages
  - Thus, there are not enough Turing machines for even **accepting** every language

Theorem

There exists a language L that is not accepted by any Turing machine.

The Diagonal Language

Definition

The **diagonal function** $D\colon \{0,1\}^* \to \{0,1\}$ is defined as

$$D(\alpha) = \begin{cases} 0 & \text{if } M_\alpha(\alpha) = 1, \text{ and} \\ 1 & \text{otherwise.} \end{cases}$$

- **The corresponding language is the diagonal language**

Undecidability

Theorem

The diagonal language is undecidable.

- **Proof:**
  - Assume D is decidable
  - Then there exists a TM M such that $M(\alpha) = D(\alpha)$ for all $\alpha \in \{0,1\}^*$
  - In particular, $M(\llcorner M \lrcorner) = D(\llcorner M \lrcorner)$
  - This is a **contradiction**: by definition of D,
    - $M(\llcorner M \lrcorner) = 0$ implies $D(\llcorner M \lrcorner) = 1$,
    - $M(\llcorner M \lrcorner) = 1$ implies $D(\llcorner M \lrcorner) = 0$

The Halting Problem

Definition

The **halting function** HALT is defined as

$$\text{HALT}((\alpha, x)) = \begin{cases} 1 & \text{if } M_\alpha \text{ halts on input } x \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

- **The corresponding language is the halting problem**

The Halting Problem

Theorem

The halting problem is undecidable.

- **This proof is our first example of reduction:**
  - We will show that if we could decide the halting problem, we could also decide the diagonal language
  - The halting problem is **more difficult** than the diagonal problem

Proof: Halting Problem Is Undecidable

- **Assume there is a Turing machine M that decide the halting problem**
- **The we can decide the diagonal language as follows:**
  - On input $\alpha \in \{0,1\}^*$, simulate $M(\alpha, \alpha)$
  - If $M(\alpha, \alpha) = 0$, M_α does not stop on input α:
    - Output 1
  - If $M(\alpha, \alpha) = 1$, M_α stops on input α:
    - We can use universal simulation to compute $M_\alpha(\alpha)$
    - Output 0 if $M_\alpha(\alpha) = 1$, and output 1 otherwise

Implications of Undecidability

- **Halting problem is relevant in practice**
  - Implication: we cannot use programs to check that programs function correctly
  - Specifically, cannot check for **infinite loops**
- **More generally: Rice's theorem**
  - **Semantic properties** of Turing machines are undecidable
  - Does TM M compute something specific on input x?
  - Does TM M compute something specific on all inputs?
- **For example:** Does the Turing machine M halt on all inputs?

Lecture 3: Summary

- Encoding objects as binary strings
- Encoding Turing machines as binary strings
- The universal Turing machine
- Existence of undecidable problems
- Halting problem is undecidable