Lecture 1: Computation, Information and Intelligence

Models, Principles and Methodology

Angsheng Li

BeiHang University

Computational Theory
17, Sept., 2019

Outline

1. Overview - **From Logic to Intelligence**
2. Logic
3. Computation
4. Information, **Information processing?**
5. Learning, new learning, **human-like learning?**
6. Game, new game, **real world game?**
7. **What Is Intelligence?**
8. **Intelligence=Learning+Game**

什么是计算理论? Big Picture!

为什么要学计算理论? 怎样理解计算这个概念?

1. Computation - History
2. Computing devices
3. Computation vs Proof
4. **Computing vs Game**
5. **Computing vs Learning**
6. **Computation vs Information**
7. **Computing vs Evolution**
8. **Computation vs Intelligence**

Our Course

1. **The big picture of computation**
2. **The most fundamental achievements of computational theory**: The most important **Definitions, Models, Principles, Methods, and Applications**
   读创始人原著、学经典、不忘初心
3. **The role of computation in the new movement of science progress - towards AI and learning etc**
4. Assignments (40%, before 9th week)
5. Final examination (60%)

Contents

Three Aspects of Computation

1. Computable functions - Mathematics
2. Computing devices - Math and Physics
3. Algorithms - Art, Science and Technology

- **Algorithm is the Engine** of computer science
- **Algorithmic New Idea** is essential to all aspects of computer science

The Key to Computation

1. **Modelling**
   Modelling what?
2. **Optimization**
   What we should optimize?
3. **Algorithms**
   How fast can we compute?

Proof

Hilbert 1900 Conjecture: Any true mathematical statement is provable.

- **Axiomatic mathematics**
- Mathematical definition of "**proof**"
- **Gödel**, 1930, There are statements that are true, but they cannot be proved within the mathematical systems, if the system is strong enough.
- Proof theory - theory about proof
- Model theory - semantics of mathematical statements
- Set theory - the universe of math
- Recursion theory - axiomatic characterisation of **computable functions**

Operations

1. $+$
2. $-$
3. $\times$
4. $\div$

Operations of numbers are basic to all math. Operations are actually algorithms

Euclidian Algorithm

Given two integers a and b, find the greatest common divisor of a and b.
For $a > 0, b > 0$, there exist unique q, r such that

$$a = q \times b + r$$

with $0 \leq r < b$.
Consequently

$$(a, b) = (b, r)$$

This suggests a recursive procedure to compute (a, b).

Programming Theory

Logical approach of programming, 1950:

- Why logic is important?
- Formal methods
- Hoare logic
- Algebraic process
- Model checking
- . . .

Computational Theory

- **Turing**, 1936: Model of computer (Birth 1912, June); 24 years old
- Hartamanis, Stearns, 1965, Computational complexity
- Cook, 1971, NP completeness
- Karp, 1986, NP completeness in combinatorics
- Hopcroft, 1979, Depth first search, width first search
- Rabin, 1980, Randomized algorithms, primality test
- Yao, 1982, Trapdoor theory, Security protocol, 1984, Communication complexity

Shannon's Information

Shannon, 1948:
Given a distribution $p = (p_1, p_2, \cdots, p_n)$, the Shannon's entropy is

$$H(p) = -\sum_{i=1}^{n} p_i \cdot \log_2 p_i. \tag{1}$$

p_i is the probability that item i is chosen, $-\log_2 p_i$ is the "self-information" of item i.

- Shannon's information measure the uncertainty of a probabilistic distribution.
- This metric and the associated notions of noises form the foundation of information theory and the information theoretic study in all areas of the current science.
- Shannon's metric provides the foundation for the current generation information technology.

Principles of Communications

Shannon's theory perfectly solves the **fundamental questions of communications**.

- The lower bound of data compression - entropy
- The transmission rate of point to point communication: channel capacity
- Shannon's theory guarantees that lossless information transmission is possible

Artificial Intelligence

- Turing, 1950: Turing test, learning machine
- Logical approach: applied logic, theorem proving, from 1950's
- Neural network, 1950's
- Valiant, 1984: PAC learning
- Deep learning: Successful applications of neural networks

Alan Turing

The Changing Computing -I

1. 20 世纪
   计算机有用
   Computer and algorithms
2. 21 世纪
   计算机无处不用
   Modelling the world, and computing the world, this will be the story in the 21st century!

- 革命性变化
- 中国的机遇

The Changing Computing-II

What does a computer do?

1. The main stream of CS in the 20th century is to investigate:
   - 1.1 **Computable functions and computing devices**
   - 1.2 **Efficient algorithms**
2. The new mission of CS in the 21st century is to study:
   - 2.1 **Computing the real world**
   - 2.2 mathematical definition of **Information processing**?
   - 2.3 Mathematical definition of **Intelligence**?

Understanding of Logic

The challenges came from **philosophy** and **mathematics**

- The **laws** of human thought - philosophy
- Mathematical definition of **proof** - mathematics
  The limit of proofs: There are true statements that cannot be proven.
- The rules of reasoning, can be taken as **intelligence** - CS
- The math support of computer programming - CS
  The **soundness** and **completeness** approaches to the proofs of CS results - methodology
- The source of algorithmic problems such as SAT, CSP, circuits QSAT etc - CS
- The **design** of computer

How hard of logic?

1. The essence is Boolean functions
2. $0, 1$ **(阴阳)**values as the Chinese philosophy - **China Question!**

General view

- Understanding the various closely related, but different concepts, such as **definition**, **proofs**, **induction**, **recursion** and **computation**.
  What the is **essence** of the **computation** and **proof**?
- The challenges for the future

General approach of proof by induction

- Base step
  To establish the initial results as the base of the induction.
- Inductive step
  To prove that the result for a new term can be established from the established propositions

Inductive proofs

Inductive proofs naturally apply to:

- The order n of recursive sequences
- The time step of the execution of an algorithmic procedure
- The complexity of a structure such as:
  – the length of strings,
  – the number of operations used in an object
- Inductive proof is one of the main proving methods in CS. It works by induction on the time step of an algorithmic procedure or on the complexity of a structure.

The key for inductive proofs

- The key to an inductive proof is to fully understand the **structure** of the proofs
- The essence of inductive proof is to establish new things from the existing propositions, which occurs in various forms in Nature and Society
- Induction establishes the results one by one in **order**. So order is the key to induction.

Defining an object or a system

An object such as functions, sets, sequences or systems, is usually complex, for which we define it inductively.
An inductive definition of an object consists of the following steps:
Base step. Determine the initial elements of an object.
Inductive step Describe the rules to generate new elements of the objects from the existing components.
It is natural to prove by induction that the object defined by induction has certain properties.

In this case, the result is proved by induction on the **order** of the construction of the objects.

Recurrence

A **recurrence** is the function that determines the inductive step of a function, together with some initial values.
Examples:
The Fibonacci numbers are defined by

$$\begin{cases} f_0 = 0, f_1 = 1 \\ f_n = f_{n-1} + f_{n-2}. \end{cases} \tag{2}$$

For recurrence, we are interested in finding the **explicit expression** of the general term such as the f_n in the Fibonacci sequences.

The roles of recursion

- Induction is a **key** to proof
- Induction is a method for constructing systems
- Recurrence is a method for representing of a function
- Recursion is an idea for designing algorithms

Consequently, **recursion** is a key to proofs, expressions, constructions and algorithms

Definition of Algorithm

Turing machines capture the machine computability. Classic algorithms satisfy 5 **principles**, that is, the **definition of algorithm**: A set of instructions satisfying:

(i) Input

(ii) Effectiveness

(iii) Finiteness

(iv) Determinism

(v) Output

The algorithm for the greatest common divisor

Our algorithm for computing the **greatest common divisor** is a **recursive algorithm**: For natural number a, b,
Suppose that

$$a = qb + r, 0 \leq r < b. \tag{3}$$

Then

$$(a, b) = (b, r), \tag{4}$$

which gives the recursive procedure for computing the greatest common divisor of a and b.

The idea

What are the computable functions?

- Recursion is a mechanism of computation
- Recursion is the essence of computation

Encoding discrete objects by natural numbers

Any discrete or symbolic computation can be encoded as functions over natural numbers.
Therefore, we focus on the functions on natural numbers $\mathbb{N} = \{0, 1, 2, \cdots\}$.

Characteristic function

For every set $X \subset \mathbb{N}$, X is identical to its characteristic function χ_X, defined by

$$\chi_X(x) = \begin{cases} 1, & \text{if } x \in X, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Sets are the special case of functions.

Pairing function

There exist computable functions π, π^{-1} such that

- $\pi : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ is a one-to-one map such that for every $x \in \mathbb{N}$,

  $$\pi(x) = \langle y, z \rangle,$$

  for some y, z

- $$\pi^{-1}\pi(x) = x.$$

Consequence

We regard $\mathbb{N}$ and $\mathbb{N}^2$ as identical.

The Basic Functions

1. Constant functions

$$f(x) = c,$$

for some constant $c \in \mathbb{N}$.

2. The successor function

$$f(x) = x + 1.$$

3. Projection functions
For every $n \geq 1$, every i with $1 \leq i \leq n$,

$$U_i^n(x_1, x_2, \cdots, x_n) = x_i.$$

Intuition All the basic functions are computable.

Composition

Given $f(x_1, x_2, \cdots, x_n), g_1(y), \cdots, g_n(y)$,

$$f(g_1(y), g_2(y), \cdots, g_n(y))$$

is the composition of $f, g_1, \cdots, g_n$.
Intuition If $f, g_1, \cdots, g_n$ are computable, then so is the composition of the functions.

Recursion

It is an **operator** defined below.
Given $f(x)$, $g(x, y, z)$, define h by:

$$\begin{cases} h(x, 0) = f(x) \\ h(x, y + 1) = g(x, y, h(x, y)). \end{cases} \tag{6}$$

Intuition If both f and g are computable, then so is h.

Search operator

Given $f(x, y)$, define

$$g(x) = \text{the least } y \text{ such that } f(x, y) = 0. \tag{7}$$

We denote g by

$$g(x) = \mu y[f(x, y) = 0].$$

g is called *search operator* or *minimisation operator* or *μ-operator*.

Remarks:

(i) $g(x)$ may not be defined, due to the fact that there is no y such that $f(x, y) = 0$.

In this case, the search for y with $f(x, y) = 0$ is unbounded, without giving any output.

This leads to the *halting problem*.

(ii) If for every x, there is y such that $f(x, y) = 0$, and if f is computable, then so is g.

Definition of recursive function

Definition

It is an inductive definition.

Base Step: The basic functions are recursive functions.

Inductive Step: Any function defined by one of the composition operator, recursion operator and search operator from recursive functions is a recursive function.

Recursive function

For some functions f defined in the inductive step, f may not be total, in the sense that on some input x, $f(x)$ is undefined, meaning that the search enters a dead loop or unbounded searching. Those functions are called *recursively partial functions*. Usually, a function is called **recursive** if it is defined by the inductive definition above as a total function.

The algorithm of the inductive definition

The inductive definition naturally gives an algorithm that generates all recursive or recursively partial functions. This means that there is an algorithm $\mathcal{A}$ that lists all the functions of the inductive definition as follows:

$$\phi_0, \phi_1, \cdots, \tag{8}$$

where ϕ_i is the *i*th function defined by the inductive definition or algorithm $\mathcal{A}$, each of which is a partial recursive function.

Inductive definition implies an algorithm

There is no algorithm to generate exactly all the recursive functions

Suppose to the contrary that this is an algorithm $\mathcal{B}$ that generates all the recursive functions by

$$\psi_0, \psi_1, \cdots, \tag{9}$$

where ψ_i is the ith function defined by algorithm $\mathcal{B}$
Then define

$$\psi(x) = \psi_x(x) + 1.$$

Then:

- ψ is recursive
  (This can be proven.)
  Let $\psi = \psi_n$
- $\psi(n) = \psi_n(n)$ and $\psi(n) = \psi_n(n) + 1$. A contradiction.

Turing Machine

A Turing machine is four-tuple (H, Σ, Q, Δ) of finite sets Σ, Q and Δ, satisfying:

(1) H is the location of the *reading head* of M.

(2) Σ is the alphabet representing the inputs and outputs

(3) Q is the set of states, denoted $q_0, q_1, \cdots, q_l$, with q_0 and q_l being the initial and final states, respectively.

(4) Δ is the finite set of *instructions* of the form

$$(q_i, s; q_j, s', X), \tag{10}$$

where $X = L$ or R, or M.

Turing Machine - continued

(5) Instruction $(q_i, s; q_j, s', X)$ means if
the current state is q_i, and the symbol scanned by the reading head is s, then
– change the state to q_j,
– change the symbol s to s' in the cell scanned by the reading head,
– move the reading head one cell to the left if $X = L$, to the right if $X = R$, and keep unchanged if $X = M$.

Turing computation

Given a Turing machine M, the computation of M proceeds as follows:

(i) Input
It is a string of the form $s_1, s_2, \cdots s_n$ for symbols s_i's in Σ.

(ii) The initially, the state is q_0, and the reading head points at the first symbol s_1 of the input

(iii) During the procedure of the computation, all the instructions are executed in a working type.

(iv) The machine halts when is reaches its last state q_l.

(v) The output is the configuration written on the working type (or a separate output type) when it halts.

Nondeterministic Turing Machine

An instruction is a set of the following form:

$$(q, s; I)$$

where I is a set of triples (q', s', X').

Functions computable by Turing Machines

Theorem

For any function $f : \mathbb{N} \to \mathbb{N}$, if it is computed by a Turing machine, then f is a function generated by the inductive definition of recursive functions.

- **Recursive functions = Turing computable functions**

Is recursive equal to computable?

- **Recursive functions are computable**.
- Fundamental Theorem:
  Recursion $=$ Computation

Universal Turing Machine

There exists an algorithm to encode a Turing machine M to a natural number e, which generates all the Turing machines as follows:

$$M_1, M_2, \cdots, \tag{11}$$

where M_i is the Turing machine with code i, or the i-th Turing machine.

For each i, let ϕ_i be the function that is computed by M_i.

Then define U:

1. On input x, y,
2. Decode M_x,
3. Compute and output $M_x(y)$, i.e., $U(x, y) = M_x(y)$.

U is a Turing machine that simulates all M_i for all the i's., which is hence called the **universal Turing machine** - which becomes the model of the computers in the real world

Halting Problem

Let U be a universal Turing machine, define

$$K = \{x \mid U(x, x) \downarrow = 0\}.$$

Theorem

There is no Turing machine that computes K.

If χ_K is computed by M_n, then $M_n = K$. However,
Case 1. If $n \notin K$, then either $M_n(n) = K(n) = 0$, so that $n \in K$.
A contradiction.
Case 2. If $n \in K$, then $M_n(n) = K(n) = 1$, so that $n \notin K$.
In either case, there is a contradiction.

This is a typical **diagonal method**, which is a general method in wide range of disciplines.

Relative Computation

We may introduce an extra type for representing the information, referred to **oracle**, provided to a Turing machine from outside source.
In this case, we may develop the Turing machine M_x^A, which may query the information in A, called an oracle A, stored in the oracle type.
This leads to the direction of Computability Theory.
Oracle could be a new resource for algorithms

Conclusions

- Turing machine provides the mathematical model for computers, and for computer science. This is the reason why we have computer science.
- Turing machines focus only on one of the aspects of computation, that is, the **effectiveness**, or the **machine mechanism**.
- Juris Hartmanis defined the *time complexity* of a Turing computation to be the number of times that the instructions are executed, and the *space complexity* to be the number of cells used in the working type of a Turing machine. This leads to Complexity Theory.
  This feature makes Turing machine essentially different from the characterisation of recursive functions, opening the new world of human civilization.

Recursion vs Computation

- Recursion is the **closing definition** of computation
- Computation is an **open world phenomenon**

Church-Turing Thesis

The current state of the art suggests

- Every function that is computed by any device can be computed by a Turing machine.
- Every function that is computed by any device in time polynomial in n, can be computed by a Turing machine in time $n^{O(1)}$, where n is the size of the input.

New Algorithms

(i) Input
input could be large, computation may not even read the inputs

(ii) Effectiveness

(iii) Finiteness
This must be the same

(iv) Determinism

(v) Output
Output may not be unique, like the answers from internet queries

Open Question:

1. **How to realise intelligence?**
2. **Randomness is useful in real world algorithms**

Shannon's entropy

Another great achievement in the 20th century!

Definition

Given a probability $p = (p_1, p_2, \cdots, p_n)$, the **Shannon entropy of p** is defined by

$$H(p) = -\sum_{i=1}^{n} p_i \log_2 p_i. \tag{12}$$

$-\log_2 p_i$: 理解这个量?

Intuition of Shannon's entropy

Intuition

- $H(p)$ is the measure of the **uncertainty (information)** contained in the probability distribution p.
- The probability distribution p is **unstructured**.
- $H(p)$ is a number characterising the global uncertainty of p.
- $H(p)$ fails to support clearing of a data
- $H(p)$ is a one-dimensional metric

Question: How to define the metric of information (uncertainty) embedded in a physical system or graph that supports the analysis of the graph?

Entropy vs Information

- **Entropy** is the amount of uncertainty
- **Information** is the amount of uncertainty that is decoded.

Given p (probability distribution, or a random variable with the distribution, a function), $H(p)$ is the amount of uncertainty embedded in p.
Suppose that we pick an index i according to probability distribution p, we gain some information, that is, the amount of uncertainty is decoded. In this case, it is again $H(p)$. So for p

entropy $=$ **information**

Question: The statement above is misleading in many ways, especially in understanding of the concept of **Information**. For general object, other than probability distribution p, the statement above must not be true.

Prefix Free Codes

Given an alphabet Σ, we may encode all symbols in Σ by $0,1$-strings such that for any $s_1, s_2 \in \Sigma$, if $s_1 \neq s_2$, then $\sigma(s_1) \not\subseteq \sigma(s_2)$, where $\sigma(s)$ is the codeword of s.
Then

$$\{\sigma(s) \mid s \in \Sigma\}$$

is the prefix free encoding of Σ.
The prefix free encoding of Σ allows to determine the word $s = s_1 s_2 \cdots s_k$ from the codeword $\sigma(s)$ of s. That is, if σ is a $0,1$-string that encodes s, then the first symbol s_1 of s is the unique symbol $y \in \Sigma$ such that the codeword of y is an initial segment of σ. Deleting the initial segment of s_1 from σ, we obtain a code word σ_1. Then the second symbol s_2 of σ is the symbol $z \in \Sigma$ whose codeword is the initial segment of σ_1. Repeating the procedure, we decode s from the codeword σ.

Prefix Free Codes by Binary Tree

Given a rooted binary tree, we assign a label 0 or 1 to each of the edges such that for a node α of the tree, the immediate successors of α are $\alpha_0 = \alpha^\frown\langle 0\rangle$ and $\alpha_1 = \alpha^\frown\langle 1\rangle$ with α_0 to the left of α_1.
Let T be such a tree. Then every leaf of T is denoted by the labels on the path from the root to the leaf, which is a $0,1$-string.
Let C be the set of all the codewords of the leaves of T. Then C is a prefix free encoding system, since for every $\alpha, \beta \in C$, $\alpha \not\subseteq \beta$, i.e., α is not an initial segment of β.

Prefix Free Encoding

As above, we may construct a prefix free codes of a set Σ to be the set of all the leaves of an ordered rooted binary tree. However, in real application, the elements in Σ may have different frequencies of occurrence. To save the length of codewords, we may assign the symbols in Σ that have higher frequencies to have the codes of shorter lengths, and the symbols in Σ having low frequencies to have the codes of longer lengths.

Understanding of the prefix free codes

- The length of codewords may vary
- We need a way to recognise the end of the current word.
- If no codeword is an initial segment of another, the current word ends as soon as the bits after the end of the last word form a codeword
- Under prefix free coding, the binary codewords corresponding to the leaves of a binary tree, the left/right i.e., $0/1$, edges.
- Suppose that item i has probability p_i and has code of length l_i. Then the expected length of a message is

$$\sum p_i l_i.$$

Huffman's Prefix-Free Coding

Input: Weights or distribution:

$$p_1, p_2, \cdots, p_n.$$

Output: Prefix-Free Code
Idea: If p_i is small, then i is coded by longer $0/1$-string.
Initial Case: $n = 2$
The two codes are:

$$0, 1.$$

Shannon codes

Let $p = (p_1, p_2, \cdots, p_n)$ be a probability distribution of set $N = \{1, 2, \cdots, n\}$.
For each i, let $l_i = \lceil -\log_2 p_i \rceil$. We can construct a prefix-free codes α_i for i such that for each i, the length of α_i is $l(\alpha_i) = l_i$.
(**Prove this!**)

Kraft Inequality

Theorem

For any prefix code over an alphabet of size 2, the codeword lengths $l_1, l_2, \cdots, l_n$ satisfy

$$\sum_{i=1}^{n} 2^{-l_i} \leq 1. \tag{13}$$

Conversely, given a set of codeword lengths that satisfy this inequality, there exists a prefix code with these word lengths.

Proof - I

Let l be the maximum l_i, $i = 1, 2, \cdots, n$. Then

$$2^{l-l_i} \leq 2^l$$

$$\sum_{i=1}^{n} 2^{l-l_i} \leq \sum_{i=1}^{n} 2^l$$

This gives rise to

$$\sum_{i=1}^{n} 2^{-l_i} \leq 1. \tag{14}$$

Proof - II

Suppose without loss of the generality that $l_1 \geq l_2 \geq \cdots \geq l_n$.
Let $\alpha_1 = 0 \cdots 0$ with length l_1, and $x_1 = 0.\alpha_1$ as binary representation of number, and let $I_1 = [x_1, x_1 + \frac{1}{2^{l_1}})$ be an interval.
Choose x_2 be the least number x such that
- $x \geq x_1 + \frac{1}{2^{l_1}}$
- $x = 0.\alpha_2$
- $|\alpha_2| = l_2 \leq l_1$.

Suppose that $\alpha_1, \cdots, \alpha_i$ have been defined. Then define α_{i+1} by the same way as above.
Then all the $\alpha_1, \alpha_2, \cdots, \alpha_n$ are the codewords of lengths $l_1, l_2, \cdots, l_n$, respectively.

Recursion, Huffman's Codes

Let $n > 2$.

(1) Let i, j be such that the least two weights are p_i and p_j.

(2) Let $q = p_i + p_j$.

(3) Find the prefix-free codes for

$$p_1, \cdots, p_{i-1}, p_{i+1}, \cdots, p_{j-1}, p_{j+1}, \cdots, p_n, q.$$

(4) Split the item with weight q into two, labelled 0 and 1 with weights p_i and p_j, respectively.

Huffman Codes are Optimal

Theorem

Given a probability distribution $\{p_i\}$ on n items, Huffman's algorithm produces the prefix-free codes with minimum expected length.

We prove by induction on n

Basis step: $n = 2$.

1 bit is necessary. Therefore, it is the shortest possible length of codes.

Inductive step: $n > 2$.

Suppose by induction that the result holds for any distribution with $n - 1$ items.

Note that every code assigns items to the leaves of a ordered, rooted binary tree.

Inductive Step

Given a fixed tree with n leaves, we minimise the expected length by greedily assigning the messages with probability

$$p_1 \geq p_2 \geq \cdots \geq p_n$$

to leaves in increasing order of depth.
Thus every optimal code has least likely messages assigned to leaves of greatest depth.
Since every leaf at the greatest depth has another leaf as its sibling and permuting the items at the same depth does not change the expected length.
We assume that the least likely messages appear as siblings at the greatest depth.

Inductive Step - continued

Let T be an optimal tree for $p_1, \cdots, p_n$ with the least likely items p_n and p_{n-1} located at sibling leaves at the greatest depth.
Let T' be the tree obtained from T by deleting these leaves, and let $q_i = p_i$ for $i \leq n-2$, and $q_{n-1} = p_{n-1} + p_n$. The tree T' gives a code for $\{q_i\}$. Let k be the height of the leaf assigned for q_{n-1}.
The expected length for T is the expected length for T' plus q_{n-1}, since if k is the depth of the leaf assigned q_{n-1}, we lose kq_{n-1} and gain $(k+1)(p_{n-1} + p_n)$ in moving from T' to T.
Therefore, the expected length for $\{p_i\}$ is at least the expected length for $\{q_i\}$ plus $p_{n-1} + p_n$.
Key:

$$l^*(p) \geq l(T'; q) + p_{n-1} + p_n \geq l^*(q) + p_{n-1} + p_n,$$

where $l^*(p)$ is the least expected length.

Inductive Step - continued

By inductive hypothesis, the Huffman algorithm finds an optimal tree T' for $\{q_i\}$. For $\{p_i\}$, the Huffman algorithm first replaces $\{p_{n-1}, p_n\}$ by q_{n-1}. The algorithm gives a tree with length to be the least expected legth for $\{q_i\}$ plus $p_{n-1} + p_n$. Therefore, the least expected length for $\{p_i\}$ must be be at most the least expected length for $\{q_i\}$ plus $p_{n-1} + p_n$, which has been achieved by the Huffman algorithm.

Shannon's entropy is the lower bound

Theorem

For every code with binary digits, the expected length is at least the entropy of $\{p_i\}$, which is

$$-\sum_{i=1}^{n} p_i \log_2 p_i.$$

Proof - 1

Proof.

$$\begin{aligned} L - H(p) &= \sum_{i=1}^{n} p_i l_i + \sum_{i=1}^{n} p_i \log_2 p_i \\ &= \sum_{i=1}^{n} p_i \log_2 \frac{p_i}{2^{-l_i}} \end{aligned}$$

□

Proof - 2

Proof.

Set $q_i = 2^{-l_i}$. Then $\sum_{i=1}^{n} q_i = \sum_{i=1}^{n} 2^{-l_i} \leq 1$ (you are asked to prove. **By induction on *n*!**), and

$$\begin{aligned}
-(L - H(p)) &= -\sum_{i=1}^{n} p_i \log_2 \frac{p_i}{q_i} \\
&= \sum_{i=1}^{n} p_i \log_2 \frac{q_i}{p_i} \\
&\leq \log_2(\sum_{i=1}^{n} p_i \frac{q_i}{p_i}), -\log \text{ is convex,} \\
&\leq 0, \text{since } \sum_{i=1}^{n} q_i \leq 1.
\end{aligned}$$

Convexity and Cavacity

This shows that $L \geq H(p)$, equality holds if and only if $p_i = q_i$ for all i.

Definition

A function f is called **convex** in an interval (a, b), if for every $x_1, x_2 \in (a, b)$ and any $\alpha, \beta \in [0, 1]$, if $\alpha + \beta = 1$, then

$$f(\alpha \cdot x_1 + \beta \cdot x_2) \leq \alpha \cdot f(x_1) + \beta \cdot f(x_2). \tag{15}$$

We say that f is **concave**, if $-f$ is convex.

Huffman codes are greedily constructed, leading to global optimal solution. This is a general principle, since **Convex Optimization: Local optimum implies global optimum.**

What Is Information Processing (IP)?

The challenges are:

1. What is the mathematical definition of **information processing**?

Definition

(Information processing)**Given a system S, the information processing for S is to distinguish the laws from the noises in S.**

2. What is the **mathematical theory** that supports the current information processing?

The mathematical theory about the limit of distinguishing the laws from noises in a system.

Big Data Phenomenon

1. There are relationships among individuals of data
   - How to build the system of big data (unstructured data)?
2. (Assumption) The laws of big data exist in the relationships of data
3. Big data is an observed system in which laws are embedded in a structure of noises
4. The mission of data analysis is hence:
   To distinguish the laws from noises
   This is
   To decode the laws from the observed system of data
   Grand challenge:
   What is the relationship between information and computation?

Understanding of Information

1. Shannon entropy: $H(p)$ is the quantification of uncertainty contained in a probability distribution or random variable (essentially, a function)
2. **Information** is defined as the amount of uncertainty that is eliminated.
   Choosing an item i according to probability distribution p, we obtained an information of amount exactly $H(p)$.
3. This suggests that
   - 3.1 Entropy is a **static metric** associated with an object (probability distribution above)
   - 3.2 Information is a **dynamic metric** determined by both an **object** and an **action**, random selection here

The Challenges

1. What is the entropy that is contained in a complex system such as graphs?
2. What is the quantification of information obtained from a complex system such as graphs?
3. How to **generate** the maximum amount of information?

Information vs Computation

1. Is information useful in computation?
2. What is the role of computation in information?
   Computing is decoding information, that is, eliminating uncertainty.
3. What is the relationship between information and intelligence?
   Information is the basis of intelligence.

Information vs Intelligence

Grand challenge: Information vs Intelligence?

Inductive Learning

1. Classification/regression
2. Decision tree
3. Computational learning theory
4. Linear regression/logistic regression
5. Neural networks
6. Support vector machines

Statistical Learning - principle-based

1. Bayesian learning
2. Maximum a posteriori (MAP)
3. Expectation maximum
4. Maximum-likelihood
5. Model selection
6. Nonparametric models
7. Learning Bayes net structure with hidden variables - open
8. Bayesian networks

Reinforcement Learning

Turing (1948, 1950) proposed the approach.

1. For eliminating hand coding, one of the most active areas, applications in robotics
2. Agent designs: model-based, model plus utility; action-utility function; policy design
3. Direct utility estimation
4. Adaptive dynamic programming (ADP)
5. Temporal-difference (TD)
6. Action-utility, or Q-function, by ADP or a TD
7. Learning actions between current values and potentials
8. Approximate functions
9. Policy-search

Knowlede

1. What is the definition of knowledge?
   The well-defined structure of data from which both syntactic and semantic functions of data can be extracted.
2. What is the definition of law?
   The law is the rules that generate knowledge.
3. What is the goal of learning?
   - to gain knowledge
   - to apply the knowledge
   - to create knowledge?

Mathematical study of game

1. von Neumann, Morgenstern, Theory of Games and Economic Behavior, Princeton University Press, 1944
2. J. F. Nash, Jr., Equilibrium Points in *N*-Person Games, Proc. Nat. Acad. Sci. USA, 36 (1950), 48-49
3. J. Nash, Non-cooperative Games, Annals of Mathematics, Vol. 54, No. 2, pp 286- 295, 1951.

Theorem

(Nash) Every finite game has an equilibrium point.

Learning + Creating

Intelligence = Learning + Creating

Intuitively, reasoning, computing and learning are all important to the generation of intelligence. However, they are insufficient for intelligence. Intelligence must include **creation** as its key factor.

Where does creation come from?

In the human evolution, **creation** probably started from the motivation and the design of **war**!

Real World Game

SUN T., **The Art of War**

1. Human evolution is the history of games and wars
2. Maximizing the payoff in games is the motivation of creation
3. **The Design and Analysis of Structural Game** - new science
   – High dimensional structural evolutionary game in the environment of dynamical network
   Urgently call for this new theory supports the game in the 21st century, for which classic game theory fails to support

Definition of Learning

Existing learning: To learn a function or a probability distribution (a function too) from data.

Human Learning?

However, human learning is a complex system including:

- An agent
- Some guest agents
- A complex environment which is much more complex than just a probability distribution
- A learning agent learns to determine:
  – the relationships between data
  – the certainty and uncertainty of her own, her guest agents and her environment

Understanding of Intelligence

1. **Computing** is basic to intelligence
2. **Information processing** measures and decodes of the information regarding the outside world and environments
3. **Learning** increases the knowledge and the use of knowledge
4. **Game** provides the motivation and design of intelligence
5. Is there a mathematical definition of intelligence?

High Dimensional Structural Game

- Evolutionary game
- Network game
- Entangled game
- High dimensional game

A Definition of Intelligence

Intelligence=Learning+Game

- Computing and reasoning are the bases of both learning and game
- Learning - 遗传
- Game – 变异、自然选择
- 遗传、变异、自然选择

Assignment - 1

Let f be an increasing function satisfying

$$f(n) = a \cdot f(\frac{n}{b}) + cn^d. \tag{16}$$

Show that

$$f(n) = \begin{cases} O(n^d), & \text{if } a < b^d, \\ O(n^d \log n), & \text{if } a = b^d, \\ O(n^{\log_b a}), & \text{if } a > b^d. \end{cases} \tag{17}$$

Assignment - 2

Given n points

$$(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n),$$

in a plane, find the closest pair of points, where the distance is the Euclidean distance.

Hint-The Algorithm

1. Sort the points by x_i's
2. Sort the points by y_j's
3. Find a line l, orthogonal to the x-axis that divides the points into two equal sizes, the left and the right part, respectively.
   – let d_L, d_R be the solutions for the left and the right parts, respectively.
   – let $d = \min\{d_L, d_R\}$.
4. For each point on the lower boundary of a rectangle of $2d \times d$ with l as the middle line, which contains at most 8 points of the instance. Find the least distance of the point from at most the 7 other points.

The Time Complexity

The recurrence of the algorithm is:

$$f(n) \leq 2 \cdot f(\frac{n}{2}) + 7n.$$

By solving the recurrence,

$$f(n) = O(n \log n).$$

Thank You!