

1216 計算量の理論と離散数学

上原隆平、面 和成

I216 Computational Complexity
and
Discrete Mathematics

by

Prof. Ryuhei Uehara

and

Prof. Kazumasa Omote

計算量の理論

- ゴール1:
 - “計算可能な関数/問題/言語/集合”
- ゴール2:
 - 「問題の困難さ」を示す方法を学ぶ
 - 計算可能な問題であっても、手におえない場合がある！
 - 計算に必要な資源(時間・領域)が多すぎる時
 - 関連する専門用語;
 - クラスNP, $P \neq NP$ 予想, NP困難性, 還元

Computational Complexity

- Goal 1:
 - “*Computable Function/Problem/Language/Set*”
- Goal 2:
 - How can you show “*Difficulty of Problem*”
 - There are *intractable* problems even if they are computable!
 - because they require too many resources (time/space)!
 - Technical terms;
 - The class NP, P≠NP conjecture, NP-hardness, reduction

5. 計算量の理論

5.0. 概要

- 「計算可能」

➔ 計算にどのくらいの資源(コスト)が必要か？

- 計算量の理論

1. 計算量の上界の研究
2. 計算量の下界の研究
3. 計算の困難性の階層構造の研究

5. Computational Complexity

5.0. Overview

- “Computable?”

➔ How much cost is required for computation?

– Computational Complexity Theory

1. Studies on upper bound of computational cost
2. Studies on lower bound of computational cost
3. Structural studies on hardness of computation

5. 計算量の理論

5.0.1. 計算量の上界の研究

アルゴリズム論: 効率の良いアルゴリズムの設計

- アルゴリズム A が問題 X を **大きさ n のどんな入力** に対しても **高々 $T(n)$ 時間** で解けるとする.
- このとき, アルゴリズム A の時間計算量の **上界** は $T(n)$ である.

[最悪な場合の漸近的な時間計算量]

5. Computational Complexity

5.0.1. **Studies on upper bound of computational cost**

Algorithm Theory: design of efficient algorithms

- Suppose we have an algorithm A which solves a problem X in **at most** time $T(n)$ for **any input of size n** .
- Then, an **upper bound** on the time complexity of the algorithm A is $T(n)$.

[asymptotic worst case time complexity]

5. 計算量の理論

5.0.2. 計算量の下界の研究

問題 X に対するどんなアルゴリズムも最悪の場合 $T(n)$ 時間かかるとき, 問題 X の時間計算量の下界は $T(n)$ である.

- $P \neq NP$ 予想
- 暗号システムの頑健性

5.0.3. 計算の困難性の階層構造の研究

問題の困難性に関する概念「 xx 困難性」の階層構造の特徴付けを研究する

5. Computational Complexity

5.0.2. **Studies on lower bound of computational cost**

If any algorithm for a problem X takes time $T(n)$ in the worst case, a lower bound on the time complexity of the problem X is $T(n)$.

- $P \neq NP$ conjecture
- Robustness of crypto system

5.0.3. **Structural studies on hardness of computation**

Studies to characterize hardness in the level of “xx-hardness” hierarchical structure depending on the hardness

5. 計算量の理論

5.1. 計算時間の評価

5.1.1. チューリングマシンの時間計算量

定義: どんな入力に対しても停止する(決定性)チューリングマシンを M とする. このとき M の実行時間または時間計算量は以下を満たす関数 $f: \mathbb{N} \rightarrow \mathbb{N}$ である:.

$f(n)$ は長さ n のすべての入力 x に対する $M(x)$ のステップ数の最大値

このとき,

- M の実行時間は $f(n)$ 時間
- M は $f(n)$ 時間チューリングマシンという

アルゴリズムを評価/比較するためにはさらなる道具が必要.

[注意]

- $f(n)$ は長さ n の文字列すべてに対する最大値をとっている (最悪の場合の実行時間)
- 通常, $f(n)$ は単調増加関数だが...?

5. Computational Complexity

5.1. Measuring Computation Time

5.1.1. Time complexity of a Turing machine

Definition:

Let M be a (deterministic) Turing machine that halts on all inputs.

The running time or time complexity of M is the function $f:\mathbb{N}\rightarrow\mathbb{N}$ s.t.

$f(n)$ is the maximum number of steps of $M(x)$ for all x of length n

In the case,

- we say that M runs in time $f(n)$
- M is an $f(n)$ time Turing machine

We need further *tools* to estimate/compare algorithms

[Note]

- $f(n)$ takes the maximum for all strings of length n (worst case complexity)
- usually, $f(n)$ may be monotone increasing function, but...?

5. 計算量の理論

5.1. 計算時間の評価

5.1.2. O記法

定義: 自然数上の関数 f と g に対し,
 $\exists c, n_0 > 0, \forall n \geq n_0 [f(n) \leq c g(n)]$
が成立するなら, $f(n)$ は オーダー $g(n)$ の関数といい, $f(n) = O(g(n))$ と書く.

注意: 定数 c と n_0 は n とは 独立 に決められる必要がある.

例1: 自然数上の任意の関数 f, g, h に対して以下が成立:

1. $\forall n [f(n) \leq g(n)] \rightarrow f(n) = O(g(n))$
2. $[f(n) = O(g) \text{ and } g(n) = O(h(n))] \rightarrow f(n) = O(h(n))$

例2: 以下を示せ:

1. $5n^3 + 4n^2 + n = O(n^3)$
2. $5n^3 + 4n^2 + n = O(n^4)$
3. $5n^3 + 4n^2 + n \neq O(n^2)$

[コメント] $f(n) \in O(g(n))$ と書く人もいる

5. Computational Complexity

5.1. Measuring Computation Time

5.1.2. Big-O notation

Definition: For functions f and g on natural numbers, if
 $\exists c, n_0 > 0, \forall n \geq n_0 [f(n) \leq c g(n)]$
then we say $f(n)$ is in the order of $g(n)$ and denote it by $f(n) = O(g(n))$.

Remark: the constants c and n_0 must be determined independently of n .

Ex. 1: The followings hold for any functions f, g and h on natural numbers:

1. $\forall n [f(n) \leq g(n)] \rightarrow f(n) = O(g(n))$
2. $[f(n) = O(g) \text{ and } g(n) = O(h(n))] \rightarrow f(n) = O(h(n))$

Ex. 2: Prove the following:

1. $5n^3 + 4n^2 + n = O(n^3)$
2. $5n^3 + 4n^2 + n = O(n^4)$
3. $5n^3 + 4n^2 + n \neq O(n^2)$

[Comment] Some people write as $f(n) \in O(g(n))$

5. 計算量の理論

5.1. 計算時間の評価

5.1.3. 問題の時間計算量

定義: Φ を計算問題, $f(n)$ を自然数上の関数とする.
 Φ を計算するプログラム A が $f(n)$ 時間で動作して以下を満たすとする.
 $\exists c, n_0 > 0, \forall n \geq n_0 [f(n) \leq c g(n)]$
このとき Φ は $O(f(n))$ 時間で計算可能, または Φ の時間計算量は $O(f(n))$ である.

注意: 定数 c と n_0 は n とは独立に決められる必要がある.

直観的には,

「問題 Φ が $f(n)$ 時間で計算可能」というとき,

- ・ A の時間計算量は $f(n)$ より小さいかもしれない
- ・ A よりも速く Φ を計算するアルゴリズムがあるかもしれない

問題の複雑さに関する上界を与えているにすぎない

解析を改善できるかもしれない

より速いアルゴリズムが見つかるかもしれない

5. Computational Complexity

5.1. Measuring Computation Time

5.1.3. Time complexity of a problem

Definition: Let Φ be a computing problem and $f(n)$ be a function over natural numbers. If we have a program A to compute Φ that runs in time $f(n)$ such that
$$\exists c, n_0 > 0, \forall n \geq n_0 [f(n) \leq c g(n)]$$
then we say that Φ is **computable in $O(f(n))$ time**, or **time complexity of Φ is $O(f(n))$** .

Remark: the constants c and n_0 must be determined independently of n .

Intuitively,

“problem Φ is computable within time $f(n)$ ” means

- time complexity of A may be less than $f(n)$.
- there may be a faster program to compute Φ than A does.

It only gives **an upper bound** of the complexity of the problem.

Our analysis may be improved

Better algorithm may be found

5.*. 素数判定問題の歴史

PRIME

入力: 自然数 n (2進数で表現)

質問: n は素数か?

PRIME $\equiv \{n : n \text{ は素数}\} = \{2, 3, 5, 7, 11, 13, 17, \dots\}$

スターリングの公式:

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

```
program Naive(input n);
```

```
begin
```

```
  for each  $i := 1 < i < n$  do
```

```
    if  $n \bmod i = 0$  then reject end-if
```

```
  end-for;
```

```
  accept
```

```
end.
```

2~n-1までのすべての数で実際に割ってみる

$\log n \cdot \log i$ time

$O(l^6)$ 時間アルゴリズムが
2002年に開発された!!

$$\text{実行時間} \leq \sum_{1 < i < n} (c \log n \log i + d)$$

$$= c \log n \log n! + dn = O(n(\log n)^2)$$

自然数 n を表現した文字列の長さを l とすると, l はおおよそ $\log n$ である.

したがって実行時間は $O(l^2)$ である.

よって PRIME の時間計算量は $O(l^2)$ である.

5.*. A history of the PRIME problem

PRIME
 Input : a natural number n (binary representation)
 Question: Is n prime?
 PRIME $\equiv \{n : n \text{ is prime}\} = \{2,3,5,7,11,13,17,\dots\}$

Stirling's Formula :

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

```

program Naive(input n);
begin
  for each i:= 1 < i < n do
    if n mod i = 0 then reject end-if
  end-for;
  accept
end.
    
```

try to divide by numbers between 2 ~ n-1

log n • log i time

$O(l^6)$ time algorithm was developed in 2002!!

$$\begin{aligned}
 \text{running time} &\leq \sum_{1 < i < n} (c \log n \log i + d) \\
 &= c \log n \log n! + dn = O(n(\log n)^2)
 \end{aligned}$$

When the length of n is l , l is approximately $\log n$. So, the running time is $O(l^2 2^l)$. Thus, time complexity of PRIME is $O(l^2 2^l)$.

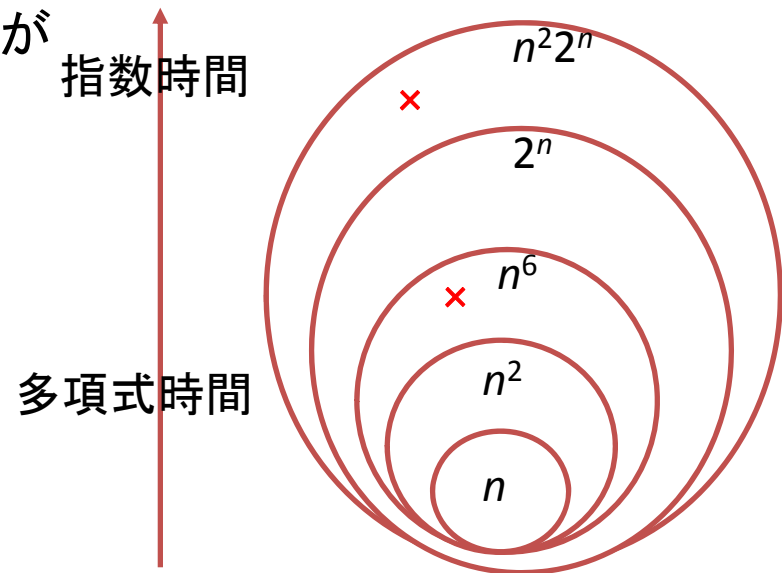
5. 計算量の理論

5.1. 計算時間の評価

5.1.3. 問題の時間計算量

定義: 自然数上の関数 $t(n)$ に対して, 時間計算量 $O(t(n))$ の集合(認識問題)全体の集合を **$O(t(n))$ 時間計算量クラス** とよび, **$\text{TIME}(t(n))$** とかく. こうした関数 $t(n)$ は制限時間と呼ぶ.

例1 PRIME は $\text{TIME}(n^2 2^n)$ の要素であったが
今は $\text{TIME}(n^6)$ の要素.



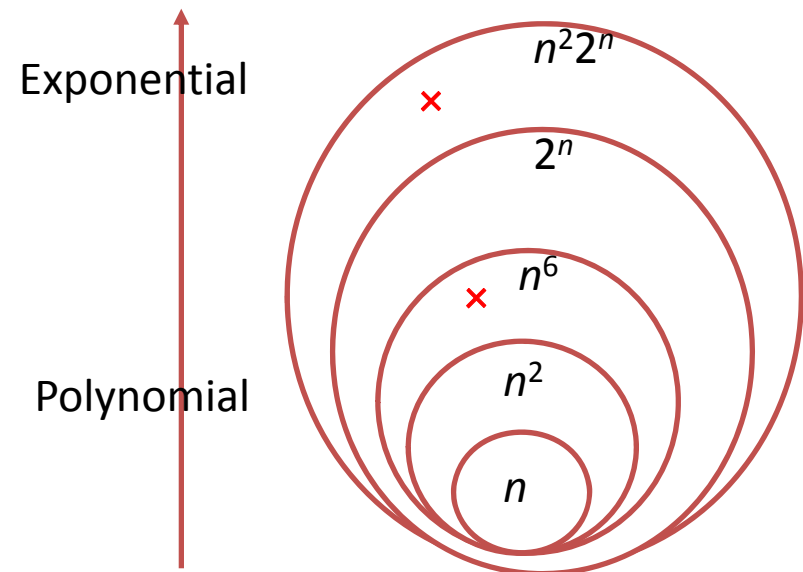
5. Computational Complexity

5.1. Time Complexity Classes

5.1.3. Time complexity of a problem

Definition: For a function $t(n)$ over natural numbers, the set of all sets (i.e. recognition problems) with time complexities $O(t(n))$ is called **$O(t(n))$ -time complexity class**, and it is denoted by **$\text{TIME}(t(n))$** . Such a function $t(n)$ is called a time limit.

Ex. 1 PRIME was in $\text{TIME}(n^2 2^n)$,
but now it is in $\text{TIME}(n^6)$.



5. 計算量の理論

5.2. 代表的な時間計算量クラス

5.2.2. 代表的な問題とその計算量

5.2.2.1. 命題論理式の評価 (PROP-EVAL)

入力: $\langle F, \langle a_1, a_2, \dots, a_n \rangle \rangle$

・ F は拡張命題論理式

・ (a_1, a_2, \dots, a_n) は F への真偽値割当て

質問: $F(a_1, a_2, \dots, a_n) = 1$?

	$x \rightarrow y$	$x \leftrightarrow y$
(x,y)	$(\neg x \vee y)$	$((x \rightarrow y) \wedge (y \rightarrow x))$
(0,0)	1	1
(0,1)	1	0
(1,0)	0	0
(1,1)	1	1

5. Computational Complexity

5.2. Representative Time Complexity Classes

5.2.2. Representative problems and their complexity

5.2.2.1. Problem of evaluating propositional expression (PROP-EVAL)

Input: $\langle F, \langle a_1, a_2, \dots, a_n \rangle \rangle$

- F is an extended propositional expression
- (a_1, a_2, \dots, a_n) is a truth assignment to F

Question: $F(a_1, a_2, \dots, a_n) = 1$?

	$x \rightarrow y$	$x \leftrightarrow y$
(x,y)	$(\neg x \vee y)$	$((x \rightarrow y) \wedge (y \rightarrow x))$
(0,0)	1	1
(0,1)	1	0
(1,0)	0	0
(1,1)	1	1

5. 計算量の理論

5.2. 代表的な時間計算量クラス

5.2.2. 代表的な問題とその計算量

5.2.2.1. 命題論理式の評価 (PROP-EVAL)

入力: $\langle F, \langle a_1, a_2, \dots, a_n \rangle \rangle$

・ F は拡張命題論理式

・ (a_1, a_2, \dots, a_n) は F への真偽値割当て

質問: $F(a_1, a_2, \dots, a_n) = 1$?

PROP-EVAL \in P

F のコードから計算木を作る.

構築に必要な時間は $O(|F|^3)$.

ひとたび計算木ができると,

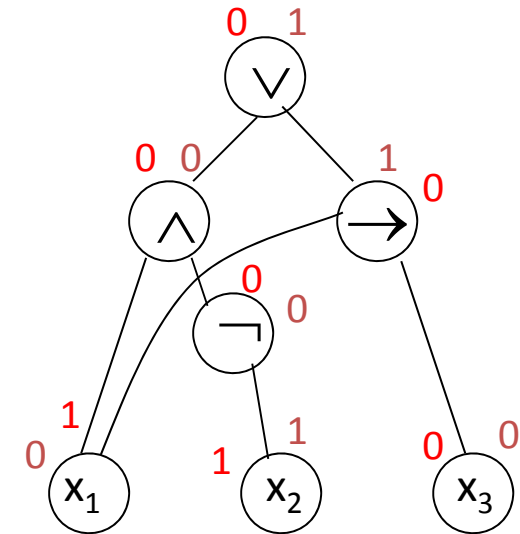
ボトムアップに計算すると

$F(a_1, a_2, \dots, a_n)$ の値は簡単に求まる.

Ex.: $F(x_1, x_2, x_3) = [x_1 \wedge \neg x_2] \vee [x_1 \rightarrow x_3]$

$$F(0,1,0) = 1$$

$$F(1,1,0) = 0$$



計算木

5. Computational Complexity

5.2. Representative Time Complexity Classes

5.2.2. Representative problems and their complexity

5.2.2.1. Problem of evaluating propositional expression (PROP-EVAL)

Input: $\langle F, \langle a_1, a_2, \dots, a_n \rangle \rangle$

- F is an extended prop. expression
- (a_1, a_2, \dots, a_n) is a truth assignment to F

Question: $F(a_1, a_2, \dots, a_n) = 1$?

PROP-EVAL \in P

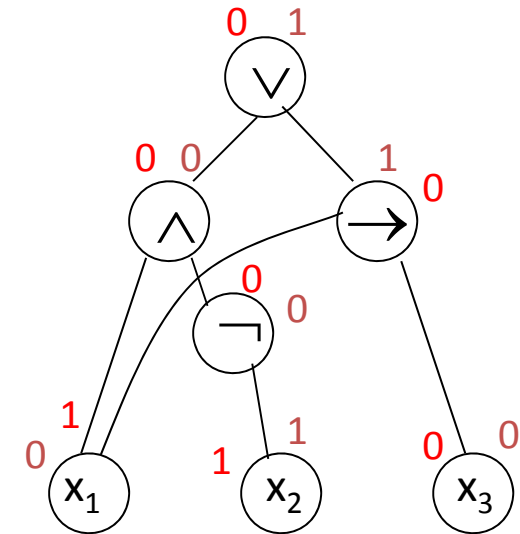
Construct a *computation tree* from a code of F .

It is built in time $O(|F|^3)$.

Once computation tree is built,
we can easily obtain the value

$F(a_1, a_2, \dots, a_n)$ in a **bottom-up fashion**.

Ex.: $F(x_1, x_2, x_3) = [x_1 \wedge \neg x_2] \vee [x_1 \rightarrow x_3]$



$F(0,1,0)=1$

$F(1,1,0)=0$

computation tree

5. 計算量の理論

5.2. 代表的な時間計算量クラス

5.2.2. 代表的な問題とその計算量

5.2.2.2. 充足可能性 (SAT)

入力: $\langle F \rangle$ F は 2-和積標準形

質問: $F(a_1, a_2, \dots, a_n) = 1$ とする真偽値割当てが存在するか?

和積標準形 (CNF)

$$F = (\bullet \vee \bullet \vee \dots \vee \bullet) \wedge (\bullet \vee \dots \vee \bullet) \wedge \dots \wedge (\dots)$$

- リテラルの \vee の \wedge で表現される式

k SAT: 各項が k 個のリテラルを含む

3SAT, 4SAT も同様に定義できる.

ちょうど/たかだか

SAT は任意の CNF を許す

ExSAT は拡張命題論理式 ($\vee, \wedge, \rightarrow, \leftrightarrow$) を許す

5. Computational Complexity

5.2. Representative Time Complexity Classes

5.2.2. Representative problems and their complexity

5.2.2.2. Satisfiability (SAT)

Input: $\langle F \rangle$ F is 2-conjunctive normal form

Question: Is there any assignment such that $F(a_1, a_2, \dots, a_n) = 1$?

Conjunctive Normal Form (CNF)

$$F = (\bullet \vee \bullet \vee \dots \vee \bullet) \wedge (\bullet \vee \dots \vee \bullet) \wedge \dots \wedge (\dots)$$

- described by \wedge of \vee of literals.

k SAT: Each closure contains k literals

exactly/at most

We can define 3SAT, 4SAT similarly.

SAT consists of any CNF.

ExSAT consists of any extended propositional expression.

5. 計算量の理論

5.2. 代表的な時間計算量クラス

5.2.2. 代表的な問題とその計算量

5.2.2.3. グラフの到達可能性問題 (ST-CON)

入力: $\langle G, s, t \rangle$: 無向グラフ G , $1 \leq s, t \leq n (= |G|)$

質問: G は s から t への経路を持つか?

5.2.2.4. オイラー閉路問題 (DEULER)

入力: $\langle G \rangle$: 有向グラフ G

質問: G はオイラー閉路を持つか?

5.2.2.5. ハミルトン閉路問題 (DHAM)

入力: $\langle G \rangle$: 有向グラフ G

質問: G はハミルトン閉路を持つか?

実際には,
有向/無向
閉路/パス
という違いは問題にならない

- 閉路とは両端点を共有する経路.
- オイラー閉路とはすべての辺をちょうど一回通る閉路.
- ハミルトン閉路とはすべての頂点をちょうど一回通る閉路.

5. Computational Complexity

5.2. Representative Time Complexity Classes

5.2.2. Representative problems and their complexity

5.2.2.3. Graph reachability problem (ST-CON)

Input: $\langle G, s, t \rangle$: an undirected graph G , $1 \leq s, t \leq n (= |G|)$

Question: Does G have a path from s to t ?

5.2.2.4. Euler cycle problem (DEULER)

Input: $\langle G \rangle$: a directed graph G

Question: Does G have an Euler cycle?

5.2.2.5 Hamiltonian cycle problem (DHAM)

Input: $\langle G \rangle$: a directed graph G

Question: Does G have a Hamiltonian cycle?

Actually,
directed/undirected
cycle/path
do not matter

- **Cycle** is a path that shares two endpoints.
- **Euler cycle** is a cycle that visits all **edges** once.
- **Hamiltonian cycle** is a cycle that visits all **vertices** once.

5. 計算量の理論

5.2. 代表的な時間計算量クラス

5.2.2. 代表的な問題とその計算量

以下は既知:

- 以下の問題は P の要素:
 - ✓ PROP-EVAL, 2SAT, ST-CON, DEULER
- 以下の問題は E の要素だが...
 - ✓ 3SAT, DHAM

P と E の間(?)にあるクラス NP

5. Computational Complexity

5.2. Representative Time Complexity Classes

5.2.2. Representative problems and their complexity

It is known that:

- The following problems are in P :
 - ✓ PROP-EVAL, 2SAT, ST-CON, DEULER
- The following problems are in E , but...
 - ✓ 3SAT, DHAM

The class NP between P and E ?