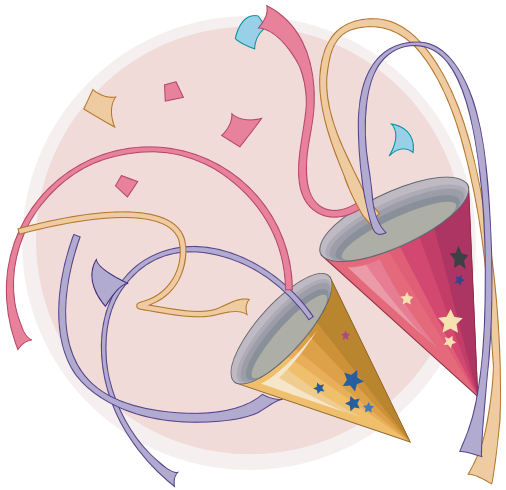
アルゴリズム論

平成22年11月04日

NP 完全問題［1］

11月7日は開学記念日です

平成９年開学
14年目の記念日です



11時00分～ 学生表彰等
13時30分～ 公開行事（13時00分 開場）

第52回 JAXAタウンミーティング in 高知工科大学

宇宙が教えてくれること

～ 人工衛星からみる宇宙開発 ～

◎ 特別上映 “HAYABUSA - BACK TO THE EARTH -”
［タウンミーティング終了後，約40分］

参加申込みが必要です

課題 c1：クラス NP

- （NP完全）充足可能性問題　satisfiability

  ［入力］乗法標準形 $B(x_1, x_2, \cdots, x_n) = F_1 \bullet F_2 \bullet \cdots \bullet F_p$

  ［性質］$B = \mathrm{True}$ となるように各変数 x_k へ
  値（True or False）を割り当てできる

- （NP完全）部分和問題　subset sum

  ［入力］正整数の集合 $A = \{x_1, x_2, \cdots, x_n\}$，定数 N

  ［性質］$\sum_{x \in B} x = N$ となる A の部分集合 B が存在する

課題 c1：クラス NP

- （NP完全）グラフの頂点被覆　vertex-cover

  ［入力］グラフ $G=(V, E)$, 正整数 K $(0<K<|V|)$

  ［性質］サイズが K 以下の頂点集合 $V' \subset V$ で
  すべての辺を被覆できる

  辺 $(u,v) \in E$ は, $u \in V'$ または $v \in V'$ のとき
  V' で被覆されるという

- （多分NP完全でない）素数判定問題　prime

  ［入力］正整数 n

  ［性質］n は素数である

課題 c1：クラス NP

- （NP完全）ナップサック問題　knapsack

  ［入力］価値が p_i で重さが c_i の n 個の荷物 $(1 \le i \le n)$,
  最大容量 C のナップサック, 総価値 P

  ［性質］重さの合計が C を超えない範囲で
  価値の合計が P 以上になるよう荷物を選べる

- （NP完全）箱詰め問題　bin packing

  ［入力］n 個の荷物の容積 $a_1, a_2, \cdots, a_n$,
  箱の大きさ B, 箱の数 K

  ［性質］すべての荷物を K 個の箱におさめられる

課題 c1：クラス NP

- （NP完全）ハミルトン閉路問題　Hamilton

  ［入力］グラフ $G=(V, E)$
  ［性質］G には, すべての頂点をちょうど１回だけ
  通って出発点に戻る閉路が存在する

- （NP完全？）ぷよぷよの連鎖数判定問題

  ［入力］４色以上のぷよぷよで, ぷよぷよが配置された
  ある盤面, 落ちてきたピース列
  ［性質］ある盤面と落ちてきたピース列から連鎖数を
  判定することができる

問題の帰着 (reducible)

- 二つの問題 Q_1 と Q_2 があり,
  Q_2 を解く多項式アルゴリズム A_2 がある
- A_2 を利用すれば, Q_1 を解く多項式時間の
  アルゴリズム A_1 を構成できるとき
  『Q_1 は Q_2 に帰着できる』（$Q_2 \triangleright Q_1$）という
- 非対称係数連立方程式問題は,
  対称係数連立方程式問題に帰着できる

帰着できることの意味

『Q_1 は Q_2 に帰着できる［$Q_2 \triangleright Q_1$］とき』

- $Q_2 \in \mathbf{P} \Rightarrow Q_1 \in \mathbf{P}$
- Q_2 は Q_1 より難しい［$Q_2 > Q_1$ という感じ］

∵ 状況の可能性は次の４とおり

『$Q_1 \in \mathbf{P} \wedge Q_2 \in \mathbf{P}$』

『$Q_1 \notin \mathbf{P} \wedge Q_2 \in \mathbf{P}$』 ⇐ この状況が起こりえない というのが $Q_2 \triangleright Q_1$

『$Q_1 \in \mathbf{P} \wedge Q_2 \notin \mathbf{P}$』

『$Q_1 \notin \mathbf{P} \wedge Q_2 \notin \mathbf{P}$』

クラス NP とは？

判定問題 Q に関する次のようなアルゴリズム A が存在するとき，Q はクラス **NP** に属するという

① Q の判定が〝yes〟となる個別問題 I に対して，その証明書 C_I が存在して，I と C_I を A に入力すると〝yes〟と判定する

② Q の判定が〝no〟となる個別問題 I に対しては，①のような証明書はとれなくてよい

③ アルゴリズム A は多項式時間で演算する

- 個別問題 instance （具体例）
- アルゴリズム A は判定問題 Q を解くのではなく，証明書の正しさを判定するだけ

NP 完全（NP-complete）の定義

- クラス **NP** に属す判定問題 Q に対し，
  NP に属すすべての判定問題が Q に帰着できるとき，Q は **NP完全** であるという

  ① $Q \in \mathbf{NP}$

  ② $Q \triangleright Q'$ for any $Q' \in \mathbf{NP}$

- **NP完全** な問題は，クラス **NP** の中でも最も難しい問題ということになる

NP 完全は NP 完全を生む

If $Q \in \mathbf{NP}\text{-complete}$, $\tilde{Q} \in \mathbf{NP}$ and $\tilde{Q} \triangleright Q$

then $\tilde{Q} \in \mathbf{NP}\text{-complete}$.

証明：$\tilde{Q} \in \mathbf{NP}$-complete となるには

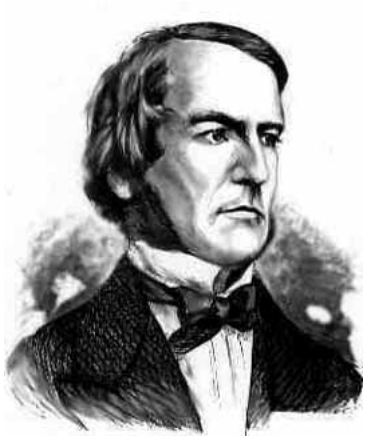
① $\tilde{Q} \in \mathbf{NP}$　② $\tilde{Q} \triangleright Q'$ $(\forall Q' \in \mathbf{NP})$

- ① は条件から成立する。
- $Q \in \mathbf{NP}$-complete なので，③ $Q \triangleright Q'$ $(\forall Q' \in \mathbf{NP})$

  よって，条件 $\tilde{Q} \triangleright Q$ と ③ から ② が成立する。

  ［関係 $\triangleright$ は推移律を満たす］

『ブール代数』



- ジョージ・ブール *George Boole*
  1815.11.02～1864.12.08
- 1855年 Mary Everest と結婚

- ブール代数は，デジタル回路の設計に必須
- 二つの２項演算（OR とAND）と一つの１項演算（NOT）が定義された代数系
- ブール代数を知らないと“情報”で生きていけない

ブール代数

- 台集合 B の上の代数系　　$< B, \vee, \bullet, \bar{} >$
- 二つの２項演算：$\vee$ および $\bullet$
- 一つの１項演算：$\bar{}$
- 特別な性質をもつ B の要素 0 と 1
  （それぞれ，零元，単位元とよぶ）
- つぎに示す公理を満たす

［公理：前提として導入される基本的な仮定］

ブール代数の公理

$a, b, c \in B$

交換律
$$\begin{cases} a \vee b = b \vee a \\ a \bullet b = b \bullet a \end{cases}$$

分配律
$$\begin{cases} a \bullet (b \vee c) = (a \bullet b) \vee (a \bullet c) \\ a \vee (b \bullet c) = (a \vee b) \bullet (a \vee c) \end{cases}$$

同一律
$$\begin{cases} a \vee 0 = a & [\text{零元の性質}] \\ a \bullet 1 = a & [\text{単位元の性質}] \end{cases}$$

補元律
$$\begin{cases} a \vee \overline{a} = 1 \\ a \bullet \overline{a} = 0 \end{cases} \quad [\overline{a} \text{ を } a \text{ の補元という}]$$

乗法標準形

- ブール変数　値は 1(真) or 0(偽)
- リテラル　ブール変数 or その補元
- 基本和　一つのリテラル or
  同じ変数を含まないリテラルの和
- 乗法標準形　一つの基本和 or
  他を含まない基本和の積

$(x \vee y)(\overline{x} \vee \overline{y} \vee z)(y \vee z)$ ← 乗法標準形

$(x \vee \overline{y})(x \vee \overline{y} \vee z)(\overline{x} \vee y \vee z) = (x \vee \overline{y})(\overline{x} \vee y \vee z)$　吸収律

和積形式のブール式

$f(x,y,z)=(x\vee y)(\overline{x}\vee\overline{y}\vee z)(y\vee z)$　← 和の積

$f(x,y,z)=1$ となるように

各変数に値を割り当てることができるか？

$f(0,0,0)=(0\vee 0)(\overline{0}\vee\overline{0}\vee 0)(0\vee 0)=0\bullet 1\bullet 0=0$

$f(0,0,1)=(0\vee 0)(\overline{0}\vee\overline{0}\vee 1)(0\vee 1)=0\bullet 1\bullet 1=0$

$f(0,1,0)=(0\vee 1)(\overline{0}\vee\overline{1}\vee 0)(1\vee 0)=1\bullet 1\bullet 1=1$

変数の割り当て方は全部で 2^n（n は変数の個数）

和積形式：CNF conjunctive normal form　ともいう

積和形式：DNF disjunctive normal form　ともいう

語順の違い

和積形式：和の積，product of sum

源義経：源（家）の義経，Yoshitsune (of) Minamoto

root mean square: ２乗の平均の平方根

$x(t) = A_m \cos(\omega t + \theta)$ の実効値

$$A_e = \langle x \rangle_{\mathrm{rms}} = \sqrt{\frac{1}{T}\int_{\tau=0}^{T} x(\tau)^2 d\tau} = \frac{1}{\sqrt{2}} A_m$$

ちなみに…

和積形式 $=0$ とできるか？

例：$f(x,y,z)=(x\vee y)(\overline{x}\vee\overline{y}\vee z)(y\vee z)=0$

$$f(0,0,?)=(0\vee 0)(?\vee ?\vee ?)(?\vee ?)=0$$

$$f(1,1,0)=(?\vee ?)(\overline{1}\vee\overline{1}\vee 0)(?\vee ?)=0$$

積和形式 $=1$ も簡単

例：$f(x,y,z)=xy\vee\overline{x}\,\overline{y}\,z\vee yz=1$

$$f(1,1,?)=1\bullet 1\vee ?\bullet ?\bullet ?\vee ?\bullet ?=1$$

積和形式 = 0 とできるか

例：$f(x,y,z) = x\,y\,\overline{z} \vee \overline{x}\,z \vee \overline{x}\,\overline{z} \vee x\,y\,z \vee x\,\overline{y} = 0$

$$f(0,0,0) = 0 \bullet 0 \bullet \overline{0} \vee \overline{0} \bullet 0 \vee \overline{0} \bullet \overline{0} \vee 0 \bullet 0 \bullet 0 \vee 0 \bullet \overline{0} = 1$$

$$f(0,0,1) = 0 \bullet 0 \bullet \overline{1} \vee \overline{0} \bullet 1 \vee \overline{0} \bullet \overline{1} \vee 0 \bullet 0 \bullet 1 \vee 0 \bullet \overline{0} = 1$$

$$f(0,1,1) = 0 \bullet 1 \bullet \overline{1} \vee \overline{0} \bullet 1 \vee \overline{0} \bullet \overline{1} \vee 0 \bullet 1 \bullet 1 \vee 0 \bullet \overline{1} = 1$$

$$f(1,1,1) = 1 \bullet 1 \bullet \overline{1} \vee \overline{1} \bullet 1 \vee \overline{1} \bullet \overline{1} \vee 1 \bullet 1 \bullet 1 \vee 1 \bullet \overline{1} = 1$$

実は，$f(x,y,z) = 0$ とすることはできない

$$\begin{aligned} f(x,y,z) &= x\,y\,\overline{z} \vee \overline{x}\,z \vee \overline{x}\,\overline{z} \vee x\,y\,z \vee x\,\overline{y} \\ &= (x\,y\,\overline{z} \vee x\,y\,z) \vee x\,\overline{y} \vee (\overline{x}\,z \vee \overline{x}\,\overline{z}) \\ &= x\,y(\overline{z} \vee z) \vee x\,\overline{y} \vee \overline{x}(z \vee \overline{z}) = x\,y \vee x\,\overline{y} \vee \overline{x} = x \vee \overline{x} = 1 \end{aligned}$$

充足可能性問題（SAT）

- 和積形式のブール式の値が 1 となる（充足する）ように，各変数に 1 または 0 を割り当てることができるか。

  SAT = satisfyability problem

- 問題 SAT は判定問題
- 問題 SAT は **NP** に属する
  証明書は，ブール式が 1 となるような各変数へ値の割り当て方

NP 完全（NP-complete）の定義

- クラス **NP** に属す判定問題 Q に対し，
  NP に属すすべての判定問題が Q に帰着できるとき，Q は **NP完全** であるという

  ① $Q \in \mathbf{NP}$

  ② $Q \triangleright Q'$ for any $Q' \in \mathbf{NP}$

- **NP完全** な問題は，クラス **NP** の中でも最も難しい問題ということになる

Cook の定理（1971）

【Cook の定理】 SAT は NP完全である

証明の概略

- **NP** に属す任意の判定問題 Q を考える
- $Q \in$ **NP** であるから, Q の個別問題 I とその証明書 C_I の正当性を判別する多項式アルゴリズムは Turing 機械 TMQ で実現できる
- Q の判定が “yes” となる I と C_I を TMQ が受理するとき, かつそのときに限り充足可能になるブール式をつくることが可能である。
  さらに, そのブール式を充足させる変数の値から, C_I を記述できる。

従って, $SAT \triangleright Q$ $(\forall Q \in$ **NP**$)$

3SAT も NP-完全

- 3SAT :
  　個別問題である和積形式のブール式の各和項が，高々３個のリテラルの和からできているという条件がついた充足可能性問題

SAT ∈ NP-complete なので,

3SAT ▹ SAT であることを示せばよい

SAT は 3SAT に帰着できる

$f = F_1 \bullet F_2 \bullet \cdots \bullet F_i \bullet \cdots \bullet F_p$ (各 F_i は和項)

$g = G_1 \bullet G_2 \bullet \cdots \bullet G_j \bullet \cdots \bullet G_q$ (各 G_j は３個以下のリテラル)

f が充足可能 $\Leftrightarrow$ g が充足可能

例えば $F = x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5 \vee x_6$ のとき

$G = (x_1 \vee x_2 \vee z_1)(x_3 \vee \overline{z}_1 \vee z_2)(x_4 \vee \overline{z}_2 \vee z_3)(x_5 \vee x_6 \vee \overline{z}_3)$

SAT のための f が与えられたら
このルールで g を作り
3SAT アルゴリズムで解けばよい

よって 3SAT ▹ SAT である

SAT は 3SAT に帰着できる

$F = x_1 \vee x_2 \vee \cdots \vee x_k \quad (k \geq 4)$

$G = (x_1 \vee x_2 \vee z_1)(x_3 \vee \overline{z}_1 \vee z_2)\cdots(x_r \vee \overline{z}_{r-2} \vee z_{r-1})\cdots(x_{k-1} \vee x_k \vee \overline{z}_{k-3})$

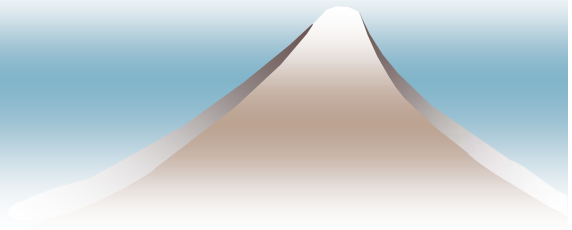
F が充足可能なとき

- $x_r = \mathrm{T}$ となる r がある
- $z_1 = \cdots = z_{r-2} = \mathrm{T}$, $z_{r-1} = \cdots = z_{k-2} = \mathrm{F}$ とすると $G = \mathrm{T}$

$$G = \cdots(x_{r-1} \vee \overline{z}_{r-3} \vee z_{r-2})(x_r \vee \overline{z}_{r-2} \vee z_{r-1})(x_{r+1} \vee \overline{z}_{r-1} \vee z_r)\cdots$$

G が充足可能なとき

- $x_1 = \cdots = x_k = \mathrm{F}$ と仮定する
- $G = z_1(\overline{z}_1 \vee z_2)(\overline{z}_2 \vee z_3)\cdots(\overline{z}_{k-4} \vee z_{k-3})\overline{z}_{k-3}$
  $= z_1 z_2(\overline{z}_2 \vee z_3)\cdots(\overline{z}_{k-4} \vee z_{k-3})\overline{z}_{k-3} = z_1 z_2 \cdots z_{k-3}\overline{z}_{k-3} = \mathrm{F}$
- G が充足可能であることに矛盾する



和積形式ブール式をリストで

- ブール式の変数は $x_1, x_2, \cdots, x_n$ とし，添字を変数名とする
- x_1 は“1”，$\overline{x}_2$ は“-2”と表す
- $(x_1 \vee \overline{x}_2 \vee x_4)(x_3 \vee \overline{x}_4 \vee x_5)(\overline{x}_2 \vee \overline{x}_4 \vee \overline{x}_5)(x_1 \vee \overline{x}_5)$ は
  [[1, -2, 4], [3, -4, 5], [-2, -4, -5], [1, -5]]
- $(x_{11} \vee \overline{x}_{21} \vee x_{41})(x_{13} \vee \overline{x}_{24} \vee x_{21})(\overline{x}_{23} \vee \overline{x}_{21} \vee \overline{x}_{31})(x_{22} \vee \overline{x}_{25})\cdots$

  変数リスト：[11, 13, 21, 22, 23, 24, 25, 31, 41]

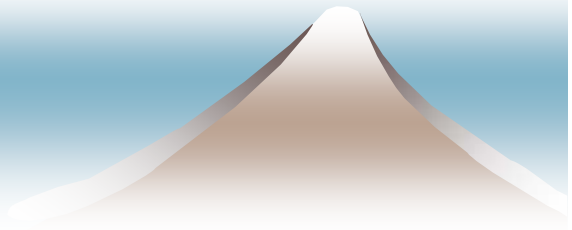
  $\Rightarrow (x_1 \vee \overline{x}_3 \vee x_9)(x_2 \vee \overline{x}_6 \vee x_3)(\overline{x}_5 \vee \overline{x}_3 \vee \overline{x}_8)(x_4 \vee \overline{x}_7)\cdots$

変数名を再構成

```
def restructure(A):
    """ A で使われている変数名を１からに再構成する """
    vrblList = []
    for term in A:          # term はリテラルの和
        for vrbl in term:
            if vrbl < 0:  vrbl = -vrbl
            if vrbl not in vrblList:  vrblList.append(vrbl)
    vrblList.sort()
    for term in A:
        for k in range(len(term)):
            if term[k] < 0:  sign, vrbl = (-1, -term[k])
            else:            sign, vrbl = ( 1,  term[k])
            term[k] = (vrblList.index(vrbl) + 1) * sign
    return vrblList
```

充足可能性問題

```
def satisfiablity(A):
    """ CNFリスト A の充足可能性を調べる """
    vrblList = restructure(A)
    termNum = len(A)
    vrblNum = len(vrblList)
    values = [0 for k in range(vrblNum)]
    while 1:
        if countSats(A, values) == termNum:
            print values
            return True
        if nextVal(values) == False:
            return False
    return False
```



課題 sat1：充足可能性問題

- 関数 countSats と nextVal を作成して，充足可能性問題を解くプログラム satisfiablity を完成せよ

```
def countSats(A, values):
    """ A に値 values を適用したとき充足する和項数を返す """
     ......
    return count

def nextVal(values):
    """ 値のリスト values を次の値にする """
     ......
    return False   # 次がなければ False を返す
```

課題 sat2 & sat3

- ［課題 sat2］２充足可能性問題 2SAT はクラス **P** に属すことを示せ［難しい］

- ［課題 sat3］和積形式のブール式が充足可能であるかどうかを答える多項式時間アルゴリムがある。これを利用して，変数の個数 n の多項式時間で変数の充足割り当て(の一つ)を見つけるにはどうしたらよいか。

countSats の高速化

- 各和項の変数名(1～n と -1～-n)をすべて +n すると 0, 1, ..., n-1, [n], n+1, n+2, ..., 2n となる。和項の中に 0 はないので, n となることはない。
- これに対応して, 変数値のリスト values の長さも 2n+1 にする必要がある。
- n = 4 とすると,

  values の初期値 = [1, 1, 1, 1, X, 0, 0, 0, 0]
  → [1, 1, 1, 0, X, 1, 0, 0, 0]
  → [1, 1, 0, 1, X, 0, 1, 0, 0]

  [0, 0, 0, 0, X, 1, 1, 1, 1] が最後