第８回：１１月２５日（水）

8 第８回

最適化（後半）では離散最適化を取り扱う．

今回の講義では，離散最適化問題の一般的な定義とその例を与える．また，問題を解くアルゴリズムの計算時間や使用領域の量を評価するために**計算量**の概念を導入する．

8.1 離散最適化問題の一般的定義

最適化問題の一般形

離散最適化問題について触れる前に，最適化問題の一般形を改めて示しておく．

$$\begin{array}{ll} \text{最小化} & f(x) \\ \text{制約条件} & x \in F \end{array} \tag{1}$$

すなわち $x \in F$ なる**解** (solution) のうち，$f(x)$ の値を最小にするものを問うのが最適化問題である．

- f: **目的関数** (objective function).
- F: **実行可能領域** (feasible region). 制約条件を満たす解の集合である.
  - 解 $x \in F$ を**実行可能解** (feasible solution) という．
  - 解の定義域を D とするとき ($D \supseteq F$), $x \in D \setminus F$ を **実行不可能解** (infeasible solution) という．
- $f(x)$ は実数値あるいは整数値をとる関数 $f : F \to \mathbb{R}$ (あるいは $\mathbb{Z}$) である. ただし $\mathbb{R}$ は実数の集合, $\mathbb{Z}$ は整数の集合を表す.
- $f(x)$ を最小にする実行可能解を**最適解** (optimal solution) という．最適解の与える目的関数値を**最適値** (optimal value) という．

なお最小化問題を一般形とみなしたが，最大化問題は，その目的関数を $g(x)$ とすると，$f(x) = -g(x)$ とすることで取り扱うことができる．

- F が組合せ的な構造 (例: n 次元 $0,1$-ベクトルあるいは整数ベクトルの集合) を持つとき, 問題 (1) を**離散最適化問題** (discrete optimization problem) もしくは**組合せ最適化問題** (combinatorial optimization problem) という.

♠「講義資料と課題」2 ページ: 離散最適化問題

- 実際の問題を最適化問題として定式化するとき，目的関数は何らかの利潤あるいは費用を表すように設計するのが一般的である．利潤を表す場合は最大化問題，費用を表す場合は最小化問題とするのが自然であろう．

- 「問題の入力」とは，解きたい問題を具体的に表すためのデータ，もっと言えばパラメータの集まりと考えればよい．**問題例** (instance, problem instance, input instance) ともいう．たとえば線形計画問題の場合，目的関数および制約条件に現れる係数や定数の組がこれに該当する．問題例については今回の講義の後半で詳しく述べる．

- 与えられた問題例に対し，制約条件を満たす解，すなわち実行可能解の集合（実行可能領域）が定まる．

- 離散最適化問題の場合，個々の実行可能解は「組合せ的」な構造を有している．何をもって「組合せ的」とするかは難しいところだが，たとえば n 次元 0, 1-ベクトル・n 次元整数ベクトル・順列・集合などは「組合せ的」なものとして扱われることが多い．

- 離散最適化問題では，**実行可能解はある適当な空間上に離散的に分布しているものとみなせる**．たとえば実行可能解が n 次元整数ベクトルで表される場合，実行可能領域は n 次元実数空間における整数格子点の集合となる．**実行可能解が離散的に分布していることは連続最適化問題との大きな違いだが，離散最適化問題を難しく，また面白くしているのは正にこの点であろう．**

- 最適解とは，この実行可能解のうち，目的関数値を最小（あるいは最大）にするものをいう．

♠「講義資料と課題」3 ページ: 離散最適化問題を解くアルゴリズム

- 離散最適化問題を解くアルゴリズムとは，任意の問題例を入力として，その最適解を正しく出力するアルゴリズムをいう．

- 離散最適化問題では実行可能解の総数が有限であることが一般的である．そこで，すべての実行可能解とその目的関数値を調べあげ，最良の目的関数値 (**最適値**) を達成する実行可能解を最適解として出力する素朴なアルゴリズムが考えられる．

- ところがそのようなアルゴリズムは望ましくない．入力サイズ（すなわち問題例のサイズ）を n とすると，実行可能解の総数は最大で $n!$ 個や 2^n 個など，指数関数個に達することが一般的だからである．(たとえば実行可能解が n 個の要素の順列あるいは部分集合で表されるような問題を考えよ．)

- 詳しくは今回の講義の後半で述べるが，計算の基本操作の回数 (ステップ数) が n の多項式で抑えられるようなアルゴリズムが望ましい．そのようなアルゴリズムを**多項式時間アルゴリズム**という．

大域最適解と局所最適解

問題の複雑さについては今回の講義の後半で詳しく述べるが，特に「難しい」とされる問題の場合，多項式時間アルゴリズムのような効率の良いアルゴリズムが知られていないことが多い．このような場合，長い時間をかけて厳密な意味での最適解を求めるのではなく，実用的な時間内で（目的関数値の意味で）良質な実行可能解を探索し，それを最適解の代替とする近似的なアプローチが考えられる．以下に述べる局所最適性は，そのようなアプローチにおける解の質の基準の一つである．

- 実行可能解 $x \in F$ に「わずかな」変形を加えて得られる実行可能解の集合を x **の近傍**という．
- 近傍を写像 $N : F \to 2^F$ で表すことにすると,[1]実行可能解 $x \in F$ がすべての $x' \in N(x)$ に対して $f(x) \leq f(x')$ を満たすとき, x を (近傍 N に関する) **局所最適解** (locally optimal solution) と呼ぶ.
- 最適解を局所最適解と区別して強調するときには**大域最適解** (globally optimal solution) と呼ぶ.

近傍の例を示しておく．たとえば 8.2 節で取り上げる巡回セールスマン問題では，実行可能解は n 個の都市 $1, \ldots, n$ の順列で表されるが,

- 一つの都市を順列の他の位置に移動することで得られる解の集合,[2]
- 二つの都市の順列における位置を互いに交換することで得られる解の集合,[3]

のようなものが近傍として考えられる．

[1]集合 S に対し, 2^S は S **の冪集合** (べき集合; power set), すなわち S の部分集合すべての族を表す．たとえば $S = \{a, b, c\}$ のとき，$2^S = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$ となる．

[2]**挿入近傍**と呼ばれる．

[3]**交換近傍**と呼ばれる．

8.2 離散最適化問題の例

この 8.2 節では，離散最適化問題の具体例として巡回セールスマン問題, 最大充足可能性問題, ナップサック問題, グラフ彩色問題の四つを紹介する．

巡回セールスマン問題 (traveling salesman problem, TSP; 行商人問題)

- TSP は, 与えられた n 個の街の集合 $V = \{1, \ldots, n\}$ および街 i と j の間の距離 d_{ij} $(i, j \in V)$ に対し, すべての街をちょうど 1 度ずつ訪問したあと元に戻る**巡回路** (tour) のうち, 距離最小のものを求める問題である.
- TSP の解は順列 $\sigma : V \to V$ で表される. $\sigma(k) = i$ は, k 番目に訪れる街が i であることを表す.

> **巡回セールスマン問題 (TSP)**
>
> **入力:** n 個の街の集合 $V = \{1, \ldots, n\}$ と，街 i と j の間の距離 d_{ij} $(i, j \in V)$.
>
> **出力:** 総距離 $f_{\mathrm{TSP}}(\sigma) = \sum_{k=1}^{n-1} d_{\sigma(k),\sigma(k+1)} + d_{\sigma(n),\sigma(1)}$ を最小にする巡回路 σ.

街の表現方法や距離行列 (d_{ij}) の性質に関していくつかの特殊ケースが知られる．

- **幾何的 TSP** (geometric TSP): 各街が空間上の点として座標によって表され, 街と街の間の距離が適当なノルムで与えられる TSP.
- **ユークリッド TSP** (Euclidean TSP): 距離としてユークリッドノルムが用いられた幾何的 TSP.
- **対称 TSP** (symmetric TSP): 距離行列 (d_{ij}) が対称な TSP (すなわちすべての $i, j \in V$ に対して $d_{ij} = d_{ji}$).
- **非対称 TSP** (asymetric TSP): 距離行列が非対称な TSP.

TSP は離散最適化問題の代表選手と言っていいだろう．プリント基板の穴開けロボットの移動計画問題，配送計画問題や巡回トーナメント問題など，実社会への応用が盛んに研究されている．

♠「講義資料と課題」4 ページ: TSP の問題例とその最適解

- (a) に示したのは，日本において人口の密集した 9847 地点の座標である．
- 各座標を街とみなし，街と街の間の距離をユークリッド距離としたときの最適巡回路が (b) である．

最大充足可能性問題 (maximum satisfiability, MAXSAT) と充足可能性問題 (satisfiability problem, SAT)

MAXSAT および SAT は，初めて見る人にはきわめて抽象的な問題に映るかもしれないが，論理回路・論理関数の諸問題と密接な関係があるのみならず，様々な組合せ問題を記述することができ，モデルとしての柔軟性は極めて高い．

- n 個の変数 $y_1, \ldots, y_n$ とその否定 $\bar{y}_1, \ldots, \bar{y}_n$ を総称して**リテラル** (literal) という．
- $v = (v_1, \ldots, v_n) \in \{0,1\}^n$ を変数 $y = (y_1, \ldots, y_n)$ への 0,1 の二値割当とする．すなわち，
  - $v_j = 1$ ならば $y_j = 1$ および $\bar{y}_j = 0$;
  - $v_j = 0$ ならば $y_j = 0$ および $\bar{y}_j = 1$ が成立．
- リテラルからいくつかを選んで論理和をとったものを**節** (clause) という (たとえば $C = y_1 \vee \bar{y}_3 \vee y_4$)．
- 節 C は 0,1 の二値割当 v に対し次のように値をとる．
  - C に含まれるリテラルの 1 個以上が値 1 をとるならば 1;
  - そうでなければ (すなわち C に含まれるすべてのリテラルが値 0 をとる) 0.
- 節 C が 0,1 の二値割当 v に対してとる値を $C(v)$ で表す．$C(v) = 1$ のとき，v **は節** C **を充足する**という．

最大充足可能性問題 (MAXSAT)

入力: n 変数より成る m 個の節 $C_1, \ldots, C_m$ と各節の重み $w_1, \ldots, w_m$.

出力: $f_{\mathrm{MAXSAT}}(v) = \sum_{i=1}^{m} w_i C_i(v)$ を最大にする 0,1 の二値割当 $v \in \{0,1\}^n$.

例 ($n = 3$, $m = 4$)

$$\begin{array}{lcll} C_1 & = & y_1 \vee y_2 & (w_1 = 2), \\ C_2 & = & \bar{y}_1 \vee y_3 & (w_2 = 3), \\ C_3 & = & y_1 \vee \bar{y}_2 \vee y_3 & (w_3 = 1), \\ C_4 & = & \bar{y}_3 & (w_4 = 2). \end{array} \tag{2}$$

この問題例に対し, たとえば $v = (0,1,0)$ とすれば, $C_1(v) = C_2(v) = C_4(v) = 1$, $C_3(v) = 0$ となるので, $f_{\mathrm{MAXSAT}}(v) = w_1 + w_2 + w_4 = 7$ である．

充足可能性問題 (SAT) SAT は $\sum_{i=1}^{m} C_i(v) = m$ となる (すなわち $C_1(v) = \cdots = C_m(v) = 1$, すべての節が充足される) 割当 v の存在を問う決定問題である.

充足可能性問題 (SAT)

入力: n 変数より成る m 個の節 $C_1, \ldots, C_m$.

出力: $\sum_{i=1}^{m} C_i(v) = m$ をみたす 0,1 の二値割当 $v \in \{0,1\}^n$ が存在すればイエス, そうでなければノー. ただしイエスの場合はその証拠も出力せよ.

先に挙げた MAXSAT の問題例 (2) を SAT の問題例とみなしたとき (重み $w_1, \ldots w_4$ は考慮しない), その解はノーである. なぜなら,

- 節 $C_4 = \bar{y}_3$ を充足するには $y_3 = 0$ でなければならず,
- そのため節 $C_2 = \bar{y}_1 \vee y_3$ を充足するには $y_1 = 0$ でなければならず,
- そのため節 $C_1 = y_1 \vee y_2$ を充足するには $y_2 = 1$ でなければならないが,
- $(y_1, y_2, y_3) = (0, 1, 0)$ は節 $C_3 = y_1 \vee \bar{y}_2 \vee y_3$ を充足しないからである.

ナップサック問題 (knapsack problem, KNAPSACK)

離散最適化問題には「詰込み型」と呼ばれる類の問題がある. すなわち, 容量や面積などに関する上限の下で, できるだけ多くの要素を詰め込むことを問う問題である. たとえば平面に長方形を敷き詰める, コンテナに荷物を詰めるといった類の問題はその例である.

ナップサック問題 (KNAPSACK) はそのような詰込み型問題の最も基本的な問題として知られる.

ナップサック問題 (KNAPSACK)

入力: n 個の要素集合 $V = \{1, \ldots, n\}$, 各要素 j に対するサイズ a_j と利得 c_j, およびナップサックのサイズ b.

出力: 条件 $\sum_{j \in V} a_j z_j \leq b$ の下で $f_{\mathrm{KNAPSACK}}(z) = \sum_{j \in V} c_j z_j$ を最大にする n 次元整数ベクトル $z = (z_1, \ldots, z_n)$.

z_j を $0,1$-変数に限定する (すなわち各要素は高々一回しか選べない) 場合は, **$0,1$-ナップサック問題**と呼ばれる.

例 $(n = 4)$

$$\begin{array}{llll} c_1 = 4, & c_2 = 6, & c_3 = 13, & c_4 = 15, \\ a_1 = 2, & a_2 = 4, & a_3 = 6, & a_4 = 7, \\ b = 11. & & & \end{array} \tag{3}$$

- 解 $z = (0, 0, 1, 1)$ は実行不可能 (サイズの総和は $13 > b = 11$).
- 解 $z = (2, 0, 0, 1)$ は一般のナップサック問題に対して実行可能だが (サイズの総和は $11 \leq b$, 利得は $f_{\mathrm{KNAPSACK}}(z) = 23$), $0, 1$-ナップサック問題に対しては実行不可能.
- 解 $z = (0, 1, 0, 1)$ は $0, 1$-ナップサック問題に対して実行可能 (サイズの総和は $11 \leq b$, 利得は $f_{\mathrm{KNAPSACK}}(z) = 21$).

グラフ彩色問題 (graph coloring problem, GCP)

グラフ彩色問題 (GCP) は，離散数学における四色問題 (19 世紀に遡る) などと密接な関係がある一方，広汎なスケジューリング問題を記述することができ，オペレーションズ・リサーチの基本問題としても知られる．

- GCP は, 与えられたグラフ $G = (V, E)$ (V は節点集合, $E \subseteq V \times V$ は枝集合) に対し, 枝の両端点には同じ色を塗らないという条件の下ですべての節点に彩色するとき, 色数最小の彩色を求める問題である.
- 彩色に用いる色の集合を $\{1, \ldots, \chi\}$ とすると (χ は決定変数であって定数ではない), 彩色は写像 $\pi : V \to \{1, \ldots, \chi\}$ で表現できる.
- $\pi(i) = k$ は節点 i を色 k で彩色することを意味する.

グラフ彩色問題 (GCP)

入力: 無向グラフ $G = (V, E)$.

出力: 条件 $\pi(i) \neq \pi(j)$ $(\forall \{i, j\} \in E)$ の下で $f_{\mathrm{GCP}}(\pi) = \chi$ を最小にする彩色 $\pi : V \to \{1, \ldots, \chi\}$.

♠「講義資料と課題」5 ページ：グラフ彩色問題における彩色の例　色数 $\chi = 3$ の場合における実行可能解と実行不可能解の例を示した．右の (b) では両端点に色 1 が割り当てられた枝が存在するため，実行不可能である．

♠「講義資料と課題」6 ページ: 部屋の利用スケジュール問題へのグラフ彩色問題の応用　レンタルルームを行っている業者を想定してもらいたい.

- 五件の利用申込 $v_1, \ldots, v_5$ があったとする．各申込の希望するタイムスロットが，上側のチャートに示されている.

  = たとえば申込 v_1 はタイムスロット t_5, t_6, t_7, t_8 を希望している.

  = 申込 v_3, v_4 のように非連続なタイムスロットを希望する場合もある.

- 各申込に対してただ一つの部屋を割り当てたい．ただし希望タイムスロットが重複する申込に同じ部屋を割り当てることはできない．清掃等の手間を省くため，できるだけ少ない数の部屋で運用したい．どのように部屋を割り当てればよいか．なお貸し出すことのできる部屋は十分多くあり，すべての部屋は一様なものと仮定する.

この問題は,

- 申込 $v_1, \ldots, v_5$ を節点,

- 同じ部屋を割り当てることのできない申込 (節点) 同士を枝で結んだグラフ

におけるグラフ彩色問題とみなすことができる (色は部屋を表す)．実行可能解の例を下側のグラフに示す.

8.3 整数計画問題 (integer programming problem, IP 問題; 整数最適化問題)

m, n をそれぞれ自然数とする．係数 a_{ij}, b_i および c_j $(i = 1, \ldots, m,\ j = 1, \ldots, n)$ と集合 $J \subseteq V = \{1, \ldots, n\}$ が与えられたとき, 以下の形に書ける問題を整数計画問題（整数線形計画問題）と呼ぶ.

整数計画問題

変数	$z_j \geq 0,\ j = 1, \ldots, n,$
	$z_j:$ 整数, $j \in J,$
制約条件	$\sum_{j=1}^{n} a_{ij} z_j \geq b_i,\ i = 1, \ldots, m,$
目的関数	$f_{\mathrm{IP}}(z) = \sum_{j=1}^{n} c_j z_j \ \rightarrow$ 最小 (最大)

以下のように，集合 J によって特殊な呼び方をする場合もある．

- **全 IP 問題**: すべての変数が整数変数の場合 (すなわち $J = V$).
- **混合 IP 問題** (mixed integer programming problem, MIP 問題): 整数変数と実数変数が混在している場合 (すなわち $J \neq \emptyset$ かつ $J \subsetneq V$).
  - 整数変数が $0, 1$-変数に限定される場合は **全 $0, 1$-計画問題**, **混合 $0, 1$-計画問題**ともいう.
  - 簡単のため $J = V$ であっても混合 IP と呼ぶ場合もある．この最適化（後半）ではこの立場を取る．
- **線形計画問題** (linear programming problem, LP 問題): すべての変数が実数変数の場合 (すなわち $J = \emptyset$).

最短路問題を例にとり整数計画問題に書き直してみよう．

最短路問題

入力: 有向グラフ $G = (V, E)$, 非負の枝重み $w : E \rightarrow \mathbb{Z}_+$, 2 点 $s, t \in V$.

実行可能解: 点 s から点 t への有向パス P.

目的: 枝重み和 $\sum_{e \in E(P)} w(e)$ の最小化.[4]

[4] $E(P)$ は P に含まれる枝の集合を表す.

最短路問題の整数 IP への定式化の例

整数変数

$$x_{u,v} \in \{0,1\}, \qquad \forall (u,v) \in E,$$

制約式

$$\sum_{(u,v)\in E} x_{u,v} - \sum_{(v,u)\in E} x_{v,u} = \begin{cases} -1 & v = s, \\ 1 & v = t, \\ 0 & v \in V \setminus \{s,t\}. \end{cases} \tag{4}$$

目的関数

$$\sum_{e=(u,v)\in E} w(e) x_{u,v} \to \text{ 最小.}$$

変数は $|E|$ 個，制約式は $|V|$ 本である．この整数 IP の最適解 x に対応するグラフ上の解 $E_x = \{(u,v) \in E \mid x_{u,v} = 1\}$ には，s から t の有向パス P 以外にも枝重み和が 0 となる有向閉路 C がいくつか含まれることもあるが，最短路問題を正しく解くことには変わりはない．

離散最適化問題を混合 IP 問題へ書き下すことに関して注意点を述べておこう．

1. 離散最適化問題は通常は文章の形で与えられる．例えば最短路問題では，グラフや路などの用語を定義した上で,「枝に重みを持つ有向グラフにおいて与えられた 2 点間の路のうち最短のものを求めよ.」というように．文章の形で与えられた離散最適化問題を混合 IP 問題の形へ書き直す作業は人が考えて行うもので，どんな離散最適化問題も（頑張れば）混合 IP 問題に書き下せ，書き下し方は一通りとは限らない．今回の演習課題はこの書き直しに関するものである．

2. 最短路問題の問題例は枝の本数 m に比例する入力データで表現でき，m を最短路問題の問題の規模と見ることができる．しかし，最短路問題を混合 IP 問題の形で書き直した場合には，混合 IP 問題の規模（変数の個数，制約式の本数）は，m 以上となり，書き下し方によっては m の指数関数になる

3. 離散最適化問題 P を混合 IP 問題 I_P に書き直して，混合 IP 問題用のソルバーで解く場合，一般には（あるいは 1 回で解く場合には），混合 IP 問題 I_P の規模は小さいほうが望ましい．例えば，離散最適化問題 P の問題規模 n に対して，混合 IP 問題 I_P の規模が n の指数関数になることは避けたい．ただし，混合 IP 問題用のソルバーで解く場合に，混合 IP 問題 I_P のすべての変数，制約式を最初から使わない方法があり，この場合は，I_P の規模が n の指数関数になることをいとわない．その方法の概略は以下の通り，一部の変数や制約式のみからなる I_P の縮小問題 I'_P を解き，これが P の最適解でなければ，その情報をもと

に，残りの変数や制約式から適切なものを選び，I'_P に追加して解き直す．これを P の最適解が得られるまで反復する．運が良ければ（？），指数関数個の変数，制約式のうち n の多項式個程度を使っただけで P の最適解が得られる．

4. 離散最適化問題の数学的な性質を調べるために混合 IP 問題に書き直すこともある．これについては本講義の第 14 回で勉強する．

8.4 計算量

計算量

アルゴリズムを計算機上で実行するとき, 計算手間を表す**時間量** (time complexity) と必要なメモリの量を表す**領域量** (space complexity) を把握することが重要である. 両者を合わせて**計算量** (complexity; 計算複雑度) という.

- **時間量**: 計算の各基本操作を単位とみなしたとき, それらの実行回数.
- **領域量**: 計算の各時点で保持しておかねばならないデータの個数の最大値.

したがって計算量を議論するとき，想定する計算モデルにおいて何が計算の基本操作なのか，また何がデータの単位（語）なのかを明確にしておかなければならない．この最適化（後半）では，現在の一般的なコンピュータに近い RAM (Random Access Machine) モデルのように，有理数の四則演算・有理数の大小判定・変数への値の代入など，ごく「簡単な」計算を基本操作とみなす．また一語は定数長のビット列によって表されるものとする．

問題と問題例

- 通常, 一つの**問題** (problem) は無限個の**問題例** (problem instance) から成る.
- 問題例は, 問題の記述に含まれるパラメータの値をすべて具体的に与えることで定義される (例: TSP における街の数 m および各街の対に対する距離 $d_{ij}, i = 1, \ldots, m, j = 1, \ldots, m$).[5]

問題例の規模

- アルゴリズムの計算量は, 問題例の**規模** (size; 入力サイズ) n のどのような関数であるかによって評価される.
- 問題例の規模は, それを定義するために必要な入力データの長さ (語数) とするのが普通である.
  - TSP の例 (1): 距離行列を入力とする場合. 街の数 m, 一つの距離 d_{ij} はそれぞれ一語に格納できると仮定すれば, $n = 1 + m^2$.
  - TSP の例 (2): 街を k 次元上の座標として与える場合. 同様の仮定の下, $n = 1 + km$.
- アルゴリズムによっては語数のみならず, 街間の距離 d_{ij} のような数値の大きさを考慮に入れて計算量を評価する場合もある.

[5] 8.2 節では TSP における街の数を n で表したが，この 8.4 節では記号の衝突を避けるために m で表し，n は問題例の規模を表すために用いる．

- 上記 TSP の例 (1) の場合, d_{ij} を整数と仮定すると, 入力に要するデータの長さは $\lceil \log_2 d_{ij} \rceil$. したがって距離行列全体では $n = \sum_{i,j} \lceil \log_2 d_{ij} \rceil \leq m^2 \lceil \log_2 d_{\max} \rceil$, ただし $d_{\max}$ は d_{ij} の最大値.

オーダー記法

計算量を問題例の規模 n の関数として議論する際, 細部の影響を除外するため, 定数倍の違いを無視した**オーダー記法** (order notation) が用いられる.

計算量の上界値　十分大きいすべての実数を定義域に含む実数値関数 T, φ が与えられたとする. ある正定数 c と n_0 が存在して, すべての $n \geq n_0$ に対して $T(n) \leq c\varphi(n)$ が成立するとき,

$$T(n) = O(\varphi(n))$$

と書く.

- たとえば, n^2, $100n^2$, $1000n + 2n^2$ などはすべて $O(n^2)$.
- $O(1)$: **定数オーダー**と呼ばれる. n に依存しない定数で抑えられることを表す.
- $O(n)$: **線形オーダー**と呼ばれる.

2 個の $n \times n$ の正方行列の積を定義通りに実行するアルゴリズム A の計算量 $T_A(n)$ の上界は $T_A(n) = O(n^3)$ と見積もることができる.（このアルゴリズムの計算量の上界が $T_A(n) = O(n^{100})$ であると言っても解析が甘いだけで間違っていない.）

計算量の下界値　十分大きいすべての実数を定義域に含む実数値関数 T, φ が与えられたとする. ある正定数 c と n_0 が存在して, すべての $n \geq n_0$ に対して $T(n) \geq c\varphi(n)$ が成立するとき,

$$T(n) = \Omega(\varphi(n))$$

と書く. 上界値 $O(f(n))$ と下界値 $\Omega(f(n))$ が一致する場合には $\Theta(f(n))$ とも記す.

正方行列の積を定義通りに実行するアルゴリズム A では計算量の下界は $T_A(n) = \Omega(n^3)$ である．これは上界とも一致し，$T_A(n) = \Theta(n^3)$ である．n 個の要素を整列させるマージソートアルゴリズム M の計算量 $T_M(n)$ についても $T_A(n) = \Theta(n \log n)$ である．このような例は実は稀で，多くのアルゴリズムでは同じ規模 n でも問題例によって必要となる計算ステップ数が大きく変わる.

最悪計算量と平均計算量　同じ規模 n を持つ問題例は多数 (場合によっては無限個) 存在するため, それら全体を通じた評価が必要である.

最悪計算量 (worst-case complexity): 規模 n のすべての問題例の中で最大の計算量を要するものに基づいて定める.

平均計算量 (average-case complexity): 規模 n の問題例のそれぞれの生起確率を考慮して計算量の平均を用いる.

最大流アルゴリズムを利用すると n 点を持つグラフの 2 点 s,t の間で本数最大の枝素な路を見つけるアルゴリズム B が得られ，最悪計算量として $T_B(n) = O(n^3)$ と見積もることができる.

多項式オーダー

- 計算量がある定数 k を用いて $O(n^k)$ と書けるとき，これを**多項式オーダー** (polynomial order) の計算量という.
- 多項式オーダーの最悪時間量を持つアルゴリズムを, **多項式時間アルゴリズム** (polynomial time algorithm) という.

擬多項式オーダー　TSP における街間の距離 d_{ij} のような数値が計算量に関わる場合を考える.

- 入力される数値の最大値または総和を U とする.
- U を入力するのに必要なデータの長さは $O(\log U)$. $\Rightarrow$ $O(\log U)$ は多項式オーダーだが, $O(U)$ は多項式オーダーではない.
- 計算量がある定数 k, ℓ を用いて $O(n^k U^\ell)$ と書けるとき, これを**擬多項式オーダー** (pseudo-polynomial order) の計算量という.

オーダーに振り回されるな！ オーダーを振り回すな！

アルゴリズムのオーダー計算量はアルゴリズムの性能を大まかに見ているだけのときがある. 次のことに注意しよう.

1. アルゴリズムが適用される場面における例題の生起確率に基づいて平均計算量を解析できれば望ましいが，生起確率が分からないことや正確な解析が困難である．多くの場合，理論的な解析を行いやすい最悪計算量でアルゴリズムの性能の評価が行われている.
2. アルゴリズムの最悪計算量（上界値）のオーダー $O(f(n))$ は必ずしも最悪の挙動と一致しているとは限らない（実際に $f(n)$ の計算ステップを要する問題例が存在するとは限らない）．最悪計算量が最悪の挙動と一致するためには，その理論解析が精密でなければならないため，複雑なアルゴリズムの最悪計算量は精度が甘いことが多い．例えば，2 点 s,t の間で本数最大の枝素な路を見つけるアルゴリズム B の最悪計算量 $T_B(n) = O(n^3)$ は解析の仕方によっては $O(n^{2.5})$ に評価し直せるかもしれない（アルゴリズム自体を改良するのではなく）.

3. アルゴリズム B の計算量 $T_B(n)$ の最悪計算量（上界値）が $O(n^3)$ であるとき，言葉遣いとして,「アルゴリズム B の計算量は n^3 である」と言うのは，最悪計算量（上界値）が $O(n^3)$ を簡潔に言ったものとして捉えることもできるが,「アルゴリズム B は計算量が n^3 かかる」と言うのは間違っている．2. で説明したように，$\Omega(n^3)$ が示されない限り，単に解析が甘いだけかもしれないからである．

4. いま，n 点を持つグラフの 2 点 s,t の間で本数最大の枝素な路を見つけるまったく別のアルゴリズム C があり，最悪計算量として $T_C(n) = O(n^4)$ であったとしよう．このとき,「アルゴリズム B のほうがアルゴリズム C より速い」と言うのも間違っている．もうお分かりであろうが，このように言えるのは，$T_B(n) = \Theta(n^3)$, $T_C(n) = \Theta(n^4)$ くらいのことまで導出されているときに限る．これらの最悪計算量の上界は単に「現時点の解析」でのもので，同じアルゴリズムでも今後小さい解析値に変わるかもしれない．それと，かりに $T_B(n) = \Theta(n^3)$, $T_C(n) = \Theta(n^4)$ であっても，実装して例題に対して作動させた場合には，例題の生起確率によってはアルゴリズム C のほうがすぐれた性能を示すことも十分にあり得る．

決定問題のクラス $\mathcal{P}$ と $\mathcal{NP}$

SAT のように，イエスもしくはノーのいずれかで答えることを問う問題を**決定問題** (decision problem) という．

離散最適化問題に対し，それに対応する決定問題を次のように定めることができる．離散最適化問題（最小化問題とする）の実行可能領域を $\mathcal{F}$，目的関数を $f : \mathcal{F} \to \mathbb{R}$ とすると，定数 $\theta \in \mathbb{R}$ に対し，

「目的関数の値 $f(x)$ が θ 以下となるような実行可能解 $x \in \mathcal{F}$ は存在するか? イエスかノーで答えよ．ただしイエスの場合はその証拠となる実行可能解も示せ.」

という決定問題である．

一般に**離散最適化問題は，それに対応する決定問題と（多項式オーダーの範囲内で）同程度に難しい**と考えることができる．

- 離散最適化問題が解ければ，その最適値より，決定問題の解はすべての θ に対してただちにわかる．

- 決定問題が解ければ，そのサブルーチンを用いて離散最適化問題は次のように解くことができる．

  = 離散最適化問題では最適値が自明な上界と下界を持つのが一般的である．たとえばすべての枝重みが整数であるような任意の TSP 問題例において，$md_{\max}$ は最適値の上界 (m は節点の数，$d_{\max}$ は枝重みの最大値)．0 は最適値の下界となる．

= よって決定問題を解くサブルーチンを用いて，反復回数が $O(\log_2(md_{\max}))$ であるような二分探索を実行すれば，最適解を得ることができる．この反復回数は多項式オーダーである．

以上を踏まえ，決定問題の複雑さに関して話を進める．計算量理論の分野では，決定問題の主なクラスとして $\mathcal{P}$ と $\mathcal{NP}$ が知られている．

- $\mathcal{P}$ (Polynomial-time solvable): 多項式時間で解ける決定問題のクラス.
- $\mathcal{NP}$ (Nondeterministic Polynomial-time solvable): 解が与えられたとき，それがイエスの証拠となるか否かを多項式時間で判定できるような決定問題のクラス.

今回の講義で取り上げた問題をはじめ，一般的な離散最適化問題に対応する決定問題は $\mathcal{NP}$ に属する．たとえば TSP の問題例と解が与えられたとき，その実行可能性と目的関数値は，明らかに多項式時間で計算できる．

- $\mathcal{NP}$ が表す “Nondeterministic Polynomial-time solvable” は，非決定性チューリングマシン (nondeterministic Turing machine) が多項式時間で解けることを意味する．詳細は省略するが，クラス $\mathcal{P}$ および $\mathcal{NP}$ の考え方は，1930 年代に A.Turing が提唱したチューリングマシン（現在の一般的なコンピュータの理論モデルと考えればよい）を用いて，1970 年代初頭に S.A.Cook（および L.Levin）が提案した．
- 明らかに，$\mathcal{P} \subseteq \mathcal{NP}$ である．しかし $\mathcal{P} = \mathcal{NP}$ なのか $\mathcal{P} \subsetneq \mathcal{NP}$ なのかは未解決である．この問題は「$\mathcal{P}$ vs $\mathcal{NP}$ 問題」として知られ，理論計算機科学最大の未解決問題である．また米国のクレイ数学研究所が 2000 年に発表した七つのミレミアム問題のうちの一つで，その解決には百万米ドルの懸賞金がかけられている．ただし大多数の研究者は後者，すなわち $\mathcal{P} \subsetneq \mathcal{NP}$ と予想している．

♠「講義資料と課題」7 ページ：クラス $\mathcal{NP}$-完全

- $\mathcal{NP}$ のサブクラスの中で最も代表的なものは **$\mathcal{NP}$-完全** ($\mathcal{NP}$-complete) と呼ばれるクラスである．
- $\mathcal{NP}$-完全は $\mathcal{NP}$ の中で最も難しい（計算困難度が高い）問題のクラスである.「最も難しい」の意味は，もしそのうちどれか一つでも多項式時間で解くことができれば，他のすべての $\mathcal{NP}$ 問題も多項式時間で解くことができる (すなわち $\mathcal{P} = \mathcal{NP}$)，ということである．
- しかし上で述べた通り，大多数の研究者は $\mathcal{P} \subsetneq \mathcal{NP}$ と予想しており，したがって $\mathcal{NP}$-完全は $\mathcal{NP} \setminus \mathcal{P}$ に含まれると予想されている．
- $\mathcal{NP}$ 問題の難しさは，本質的にすべての解の候補（一般的に問題例サイズの指数個）を調べなければならないところにある．多項式時間で解ける問題（すなわち $\mathcal{P}$ に属する問題）は，それをしなくても解ける例外的な存在と言っていいだろう．

- だからと言って $\mathcal{NP}$ は “Not Polynomial-time solvable” でもなければ，解くのに指数時間を要する問題のクラスでもない．決して誤解しないこと．
- 残念ながら，今回の講義で取り上げた TSP, MAXSAT, GCP, KNAPSACK, IP をはじめ多くの離散最適化問題（に対応する決定問題）が $\mathcal{NP}$-完全に属する．
- なお対応する決定問題が $\mathcal{NP}$-完全に属する離散最適化問題は，一般に $\mathcal{NP}$-困難と呼ばれる問題クラスに属する．
- またグラフ同型性問題など，$\mathcal{P}$ にも $\mathcal{NP}$-完全にも属さない $\mathcal{NP}$ 問題も存在する．

【課題８－１】次の問題を整数計画問題として定式化せよ.（ヒント：最短路問題の整数計画問題への定式化を参考にせよ）

入力: 有向グラフ $G = (V, E)$, 非負の枝重み $w : E \to \mathbb{Z}_+$, 始点 $s \in V$，目的地の点 $t_1, t_2 \in V \backslash \{s\}$.

実行可能解: 点 s を根とする G の有向木 F で目的地点 t_1, t_2 を含むもの．F は G の全域木である必要はない．目的地点は T の葉である必要もない.

目的: 枝重み和 $\sum_{e \in E(F)} w(e)$ の最小化.

第 8 回の講義は以上である.