数学入門公開講座

平成１１年８月２日（月）から８月６日（金）まで

京都大学数理解析研究所

講師及び内容

1．多項式の解の近似がとりもつ数論と幾何の関係（6時間15分）

京都大学数理解析研究所・助教授　望月　新一

多項式の有理数解の研究は、歴史が長いだけに、様々なアプローチを産み出しているが、二十世紀の後半に開発され、現在では数々の輝かしい成果を挙げているアプローチとして、現代数論幾何がある。本講義の目標は、その現代数論幾何の世界を紹介することにある。現代数論幾何の基本は、標語的にいえば、多項式の解の近似にあるといってもよい。つまり、有理数というものは、整数論の対象としては構造が複雑すぎるため、数論的にはより単純な構造をした実数や複素数のような数で近似することによって多項式の有理数解を調べるのである。このような近似解のなす集合は、有理数解のなす集合と違い、「滑らかな物質」で出来た幾何的な対象をなしていて、その対象の幾何的性質が、有理数解の性質に大きく影響することが知られている。

2．計算幾何学入門（6時間15分）

京都大学数理解析研究所・助教授　田村　明久

平面上に与えられた有限個の点の集合に対して、これを含む最小の凸多角形を求める問題を（2次元）凸包問題とよびます。計算幾何学とは、このような幾何的な問題を解くアルゴリズム（解法）を研究する計算機科学の一分野です。

本講座では凸包問題のほかに勢力圏のモデルとして利用されるボロノイ図など、計算幾何学において基礎的な問題とそれらに対するアルゴリズムを紹介します。また、アルゴリズムの効率性の評価についてもふれます。

3．微積分をつうじて多様体が見える（6時間15分）

京都大学数理解析研究所・教授　宮岡　洋一

「多様体」は現代数学を理解する上で鍵となる概念です。

数学のなかでも最も古い伝統をもつ幾何学は、三次元空間という入れ物にはいっている図形という素朴な直感から出発したわけですが、百五十年ほど前のこと、リーマンは、必ずしも入れ物を必要とせず、いくらでも高い次元をもてる、多様体の概念に到達しました。この概念は解析学を複雑な図形のなかで自由に展開することを可能とし、その結果として宇宙全体の幾何構造といったものまで考察することまでできるようになったのです。

この講義では、多様体の豊かな世界への入門として、積分を通じて解析（微分形式）と幾何（コホモロジー）とがかかわりあう、その様子に焦点をしぼって解説したいと思います。

時　間　割

日 / 時　間	8月2日（月）	3日（火）	4日（水）	5日（木）	6日（金）
10：30～11：45	望月	望月	望月	望月	望月
11：45～13：00	休　憩				
13：00～14：15	田村	田村	田村	田村	田村
14：15～14：45	休　憩				
14：45～16：00	宮岡	宮岡	宮岡	宮岡	宮岡

平成11年度(第21回)数学入門公開講座テキスト(京都大学数理解析研究所, 平成11年8月2日～8月6日開催)

計算幾何学入門

京都大学数理解析研究所・助教授　　田　村　明　久

1999, AUGUST 2,3,4,5,6, 13:00～14:15

計算幾何学入門

田村 明久

1 はじめに

計算幾何学 (computational geometry) は幾何学と計算量理論（あるいはアルゴリズム理論）の交わりに位置し、幾何学的な問題に対する効率的なアルゴリズムの開発や問題の難しさを解析することが主テーマである計算機科学 (computer science) の一分野である。定規とコンパスを道具として指定の図形を描けるかを問う作図も幾何学とアルゴリズムに関連しその歴史は古いが、計算機を道具としさらにはアルゴリズムの効率性まで考慮する計算幾何学の歴史は新しく、R.L. Graham の２次元凸包問題に対するアルゴリズムや M.I. Shamos の "Computational Geometry" と題された博士論文など１９７０年代から始まり近年急速に発展した分野である。

まずは計算幾何学で扱う基本的な問題を幾つか紹介しよう。

[凸包問題] n 個の点が与えられたとき、これらすべてを含む最小の凸多面体（凸集合）を求めよ。計算幾何学で扱う問題の中でも最も基本的な問題の一つである。

[ボロノイ図構成問題] 平面上に n 個の点（母点）が与えられたとき、各母点 p にその他の母点より直線距離で近い点の集合を母点 p のボロノイ領域とよぶ。ボロノイ領域の境界線によって平面上に描かれる図形をボロノイ図とよぶ。ボロノイ領域は母点の勢力圏を意味し、各母点の勢力圏を求めることがボロノイ図の構成に対応する。

[位置決定問題] 平面の領域分割（地図、ボロノイ図など）に対して、質問点が含まれる領域を求める。

[可視化問題] 平面上に多数の多角形（障害物）が与えられたときに、指定された点から見える部分を求めよ。

[最短経路問題] 平面上に多数の多角形（障害物）が与えられたときに、指定された２点を結ぶ最短経路を求めよ。

[多角形の分割問題] 多角形の内部を指定された基本図形に分割する問題。多角形を三角形に分割するなど色々なバリエーションがある。

計算幾何学で扱う問題は平面（２次元空間）や３次元空間と言った低次元のものが多い。その理由には、低次元の幾何学的な構造を利用することで効率的なアルゴリズムが開発でき

るという理論的な側面と、実際に応用する場面でＶＬＳＩの設計、コンピュータグラフィックスなど低次元の場合が多いことによる。

本講座「計算幾何学入門」では、凸包問題とボロノイ図構成問題を中心に計算幾何学のエッセンスを、計算量理論とアルゴリズム理論の立場から紹介すること、すなわち、

- 幾何的情報をいかに採り入れるか
- アルゴリズムの効率性の評価法
- アルゴリズム効率化の技法
- 問題の難しさ

の紹介を試みる。

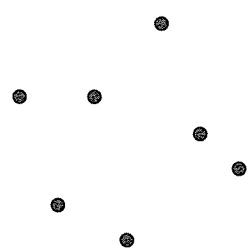
後で詳しく触れるが、２次元凸包問題「平面上に与えられたn個の点、より具体的にはn点のxy座標の列$(x_1, y_1), \cdots, (x_n, y_n)$が与えられたとき、これらを含む最小の凸多角形を求めよ」（例えば下図の左図から、右図の多角形を求める問題）に対する以下の２つのアルゴリズムを考える。話を簡単にするために、与えられたn点は同一直線上に３点以上のらないと仮定する。

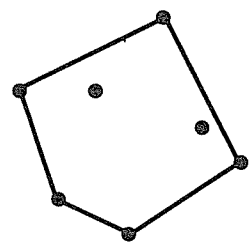
[発想１] 下図から分かるように凸多角形の辺は２点を含み、他の点はこの辺を延長した直線の片側に存在する。逆に、２点を通る直線に関して、他のすべての点が片側に存在するなら、この２点を結ぶ線分は求める凸多角形の辺となる。

アルゴリズム１

与えられたn点に対して、各２点対に対して（$\frac{n(n-1)}{2}$組）次のことを実行する

　２点を通る直線の片側に他のすべての点が存在するなら２点を結ぶ線分を出力する。





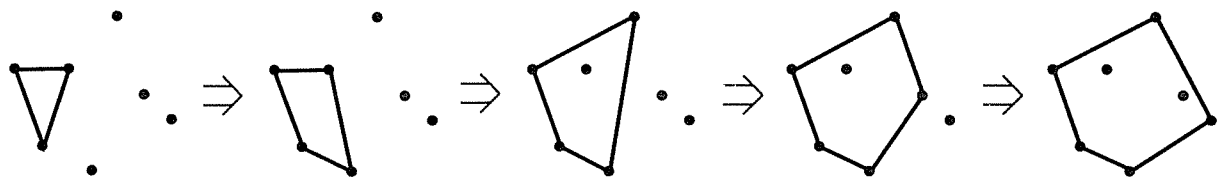
[発想２] ３点からなる三角形から順次点を追加していき凸多角形を変更していく（凸多角形の

内部の点は更新の際に考慮する必要がない）。さらに、追加する点が多角形の内部か外部の判断をしなくても良いように、n 点を x- 座標の順に並び換え、座標値の小さい順に加える。

アルゴリズム 2

Step 1. 与えられた点を x- 座標の順に並び換える。

Step 2. 左端の 3 点からなる三角形から x- 座標順に点を順次追加し凸多角形を変更する（下図参照）。



Step 2 の操作をどのように実行するかの説明は後にまわすが、はたしてどちらのアルゴリズムがより好ましいのであろうか。次の節ではアルゴリズムの評価法を簡単に紹介する。

2 アルゴリズムの評価法

アルゴリズムの効率を評価する方法には幾つかの指標がある。最も良く使われるのが、アルゴリズムの計算時間に対応する**時間計算量**と実行に必要なメモリー量に対応する**領域計算量**がある。本講座では、時間計算量を用いてアルゴリズムの効率性の評価を行なうことにする。

一つのアルゴリズムをプログラムし計算機で実行する場合にその計算時間は個々の計算機のスピードに依存してしまうため、計算時間をそのまま用いることはできない。一般的な方法としては、入力のサイズ（例えば凸包問題ならば入力点数）に関して、計算時間がどのような関数になるかを解析する。例えば、入力のサイズ n に関して計算時間が n^2 に比例するアルゴリズムでは、計算機のスピードに応じて比例定数が変化するが、n^2 に比例するということは計算機のスピードに依存しない。しかし、上記のような評価法では入力サイズのみで計算時間が決まる必要がある。凸包問題に対するアルゴリズムの場合、その計算時間は点の配置に依存することが容易に予想でき、このような仮定は一般的ではない。そこで、最も良く用いられる**最悪時間計算量**（以降、簡単のため単に時間計算量とよぶ）を紹介する。アルゴリズム A の入力サイズ n に関する最悪時間計算量 $t_A(n)$ は、同一サイズのすべての入力に対

する計算時間の最大値と n の関係を表す関数とする。また、最悪時間計算量を厳密に評価することが難しい場合も少なくない。そのために、次ような記号を用いその上界や下界を用いて評価することが多い。

変数 n を引数とする関数 $f(n)$ と $g(n)$ について次のような記号を導入する。

$$f(n) = O(g(n)) \iff [\text{ある定数 } n_1 \text{ と } c_1 \text{ が存在して}, n \geq n_1 \Rightarrow f(n) \leq c_1 g(n)]$$

$$f(n) = \Omega(g(n)) \iff [\text{ある定数 } n_2 \text{ と } c_2 \text{ が存在して}, n \geq n_2 \Rightarrow f(n) \geq c_2 g(n)]$$

$$f(n) = \Theta(g(n)) \iff [f(n) = O(g(n)) \text{ かつ } f(n) = \Omega(g(n))]$$

例えば、アルゴリズム A の時間計算量 $t_A(n)$ が、$t_A(n) = O(n^2)$ であるならば、（サイズが十分大きいならば）サイズ n のいかなる入力に対しても n^2 に比例する時間をかければこのアルゴリズムは終了することを意味し、時間計算量の上界を与える。$t_A(n) = \Omega(n \log n)$ ならば、入力サイズの列 $n = n_2, n_2+1, n_2+2, \cdots$ に対して、少なくとも $n \log n$ に比例する計算時間を要する入力の列が存在することを意味し、時間計算量を下から押える評価をしている。下からと上からの評価が一致した場合が、$\Theta(\ )$ 表記である。

前節のアルゴリズム 1 について考えよう。これは $\frac{n(n-1)}{2}$ 組の点対それぞれに対して、それらを通る直線とその他すべての点との位置関係を判定している。それぞれの点対に対して位置関係の判定は最悪の場合 $n-2$ 回必要となる。1 直線と 1 点の位置関係の判定は単位時間で実行することができるため、アルゴリズム 1 の時間計算量 $t_1(n)$ は $t_1(n) = O(n^3)$ となる。また、$\frac{n(n-1)}{2}$ 組の点対を考えるので $t_1(n) = \Omega(n^2)$ も明らかであろう。一方アルゴリズム 2 については、時間計算量 $t_2(n)$ は $t_2(n) = O(n \log n)$ となる（詳しくは後の節で述べる）。上記のことから、アルゴリズム 2 はアルゴリズム 1 よりも効率的であり、望ましいアルゴリズムである。

3 計算機のモデル

まずは準備として、アルゴリズムを記述する際に許される演算やデータ構造を定めておく。本講座用に、現実の計算機を念頭に次のような計算機モデルを考える。

[定数、変数、配列] 定数や変数が扱える。また定数や変数を指定した数だけ並べた配列も扱える。配列のイメージとしては、例えば大きさ 5 の a という配列は、

$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$
5	8	2	7	1

のように5個の箱が並んだもので、配列 a の3番目の箱の中身（すなわち値2）は $a[3]$ のように参照する。上の例は1次元配列とよばれるものであるが、箱が碁盤の目のように並んだ2次元配列（i 行 j 列の箱の中身は $a[i][j]$ で参照する）やさらに多次元の配列も扱える。定数、変数、配列の任意に指定した箱の中身の値の参照は単位時間で行なえる。

[四則演算] 定数、変数の四則演算を単位時間で行なえる。

[値の比較] 定数、変数の値の比較を単位時間で行なえる。

[値の代入] 変数に定数や計算結果を代入できる。代入は単位時間で行なえる。

[リスト構造] $5,8,2,7,1$ という数列の8と2の間に3という数を追加したいとする。配列を使ってこの数列を表現していた場合には、$2,7,1$ を1つづつ後ろの箱に移し、3番目の箱に3を代入すればよい。しかし、この実行には2以降の数列の長さに比例した時間を要する。このような要素の追加を単位時間で行なうデータ構造としてリストとよばれるものがある。イメージとしては、

$$\boxed{5} \to \boxed{8} \to \boxed{2} \to \boxed{7} \to \boxed{1}$$

のように箱を矢印でつなげたもので、3の追加には、新たな箱を用意し中身を3とし8の箱の矢印を3の箱に3の箱の矢印を2の箱に向ければよい。以上のように矢印の付け換えだけなので、数列の指定した要素1つの削除、数列の指定した要素間に新たな数列を追加、数列の連結なども単位時間で行なえる。しかし数列をリストで表現した場合には、3番目の要素を参照するためには1番目の箱から矢印をたどり3番目まで見る必要があり、配列のように単位時間で参照できないことに注意されたい。実際にはリストは配列を用いて実現できるが、細かい説明は省略する。

4　1次元の問題

まずは基本的な1次元の問題を幾つか紹介する。これらは、多次元の問題を解くためにも重要な役割をはたす。

4.1　整列問題

[整列問題] 長さ n の数列が与えられたとき、値が昇順（小さいものが前）に並び換える。

整列問題を解くアルゴリズムをソーティングアルゴリズムとよぶ。これには多くのものが

あるが、ここではマージソートとよばれるものを紹介する。発想としては、既に整列済みの2つの数列を合併 (merge) させ、1つの整列済みの数列を構成する。

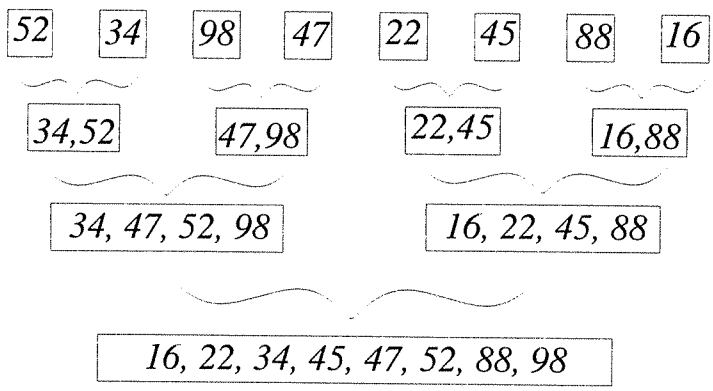
マージソート

Step 1. n 個の数をほぼ均等になるように2つに分ける。

Step 2. 分けた2つの数列をマージソートで再帰的に整列する。

Step 3. 整列済みの2つの数列を前から順番に見ていき、先頭の2つの中で小さいものを取り出す。どちらの数列も空になるまで繰り返す。

例でマージの部分の動きをみてみよう。長さ8の数列 52, 34, 98, 47, 22, 45, 88, 16 に対しては



のように最終的に整列された数列を求める。

さてマージソートの時間計算量 $T(n)$ を求めてみる。Step 1 は数列を配列を用いて表現すれば単位時間で行なえる。Step 2 は再帰的な実行なので $2\times T(n/2)$ 時間あれば十分である。Step 3 では、値の比較が $n-1$ 回行なわれるのでこの部分だけでは n に比例した時間を要する。よって

$$T(n) \leq cn + 2T(n/2) \qquad (c \text{ は適当な定数})$$

という関係式が成り立ち、これを解くと $T(n) = O(n \log n)$ が導ける。$\log n$ が現れることは上の例からも理解できる。なぜなら、n 個の要素を2等分づつしていき、要素数が1になるには約 $\log_2 n$ 回の2等分の操作が必要である。これは、上例の図の高さに対応している。上のレベルから下のレベルに移るには（例えば1つ1つばらばらなものから長さ2の数列を4つ作るには）高々 n 回の比較をすれば十分である。この操作を $\log n$ 回繰り返せば整列は終了するので、総計で $O(n \log n)$ 時間あれば十分である。

アルゴリズム高速化技法： 分割統治法 (divide-and-conquer)
マージソートのように、問題をより小さな問題に分割しそれらを解き、得た答を統合することでアルゴリズムの高速化をはかる技法。問題を等分するところがみそで、マージソートの Step 1 で 1 と $(n-1)$ に分割しては時間計算量は $O(n^2)$ となってしまう。

4.2 値探索問題

[値探索問題] 長さ n の数列 $(a[i] : i = 1, \cdots, n)$ が与えられたとき、指定の数 q が数列の中に存在するか判定する。

単純な方法として、数列の先頭から順番に q との比較を行なうものがあるが、最悪の場合には $O(n)$ 時間を要する。ここでは、数列が昇順に整列されているという状況を考えよう。探索を始める前には、q が存在するならば 1 番目から n 番目までの範囲にある（この範囲を $[1, n]$ と記述する）。数列の $k = \lfloor n/2 \rfloor$ （$n/2$ の切捨）番目の数 $a[k]$ と q を比較し、同じなら yes と答える。もし $a[k] < q$ ならば、整列されているので $[1, k]$ の範囲には q は存在しない。q が存在しうる範囲を $[k+1, n]$ に更新する。もし $a[k] > q$ ならば、範囲を $[1, k-1]$ に変更する。このような操作を繰り返し q をみつけるか、あるいは範囲 $[i, i]$ となり $a[i] \neq q$ ならば no と答える。このような探索法を 2 分探索法とよぶ。

2 分探索法

Step 1. $[i, j] := [1, n]$ と初期化する。

Step 2. $i \leq j$ である限り次の操作を繰り返す

$k = \lfloor (i+j)/2 \rfloor$;

もし $a[k] = q$ ならば yes と答えて終了する ;

もし $a[k] < q$ ならば $i := k+1$ と更新する ;

もし $a[k] > q$ ならば $j := k-1$ と更新する ;

Step 3. （$i > j$ となったので） no と答えて終了する。

2 分探索法では 1 回の比較で範囲が半分の長さになるので、$O(\log n)$ 回の比較で終了する。また、多くの場合は $\log n$ 回は比較を必要とするので時間計算量は $\Theta(\log n)$ となる。

4.3 k 番値探索問題

[k 番値探索問題] 長さ n の数列が与えられたとき、k 番目に小さい数を求めよ。

$k = 1$ ならば最小の数を探すわけで、$n-1$ 回の比較で求まることが容易に分かる。一般の k ではどうであろうか。安直な方法としては、最小の数を探しては取り除くことを k 回繰り返せば求まるが、$k = \lfloor n/2 \rfloor$ のとき、この方法では n^2 に比例する比較を必要としてしまう。整列問題を解いてから、k 番目を求める方が効率的ではあるが、それでも時間計算量は $O(n \log n)$ である。ここでは、Blum, Floyd, Pratt, Rivest, Tarjan[3] の $O(n)$ 時間アルゴリズムを紹介する。話を簡単にするために数列中の数値はすべて異なるとする。

BFPRT アルゴリズム

Step 1. n 個の数を 5 個づつのグループに分割し、各グループを昇順に整列する。

Step 2. $n/5$ 個のグループの中央値の中央値 M をこの方法を再帰的に用いて求める。

Step 3. 各グループに対して M より小さい数と大きい数に分割する。

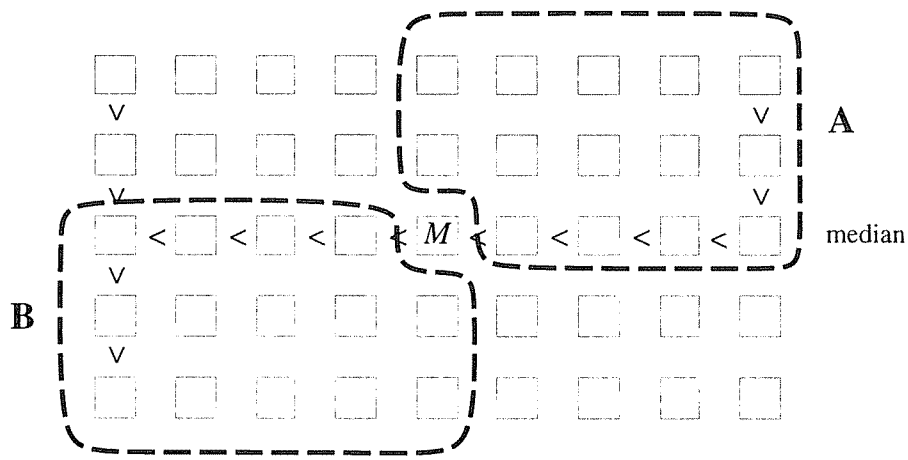
Step 4. M より小さい数が r 個存在したとする。

$k = r+1$ ならば、M が求める答えとなり、終了。

$k < r+1$ ならば、M より小さい r 個の数の中で k 番目の数を再帰的に求める。

$k > r+1$ ならば、M より大きい $n-r-1$ 個の数の中で $k-r-1$ 番目の数を再帰的に求める。

Step 2 が終了したときの様子は次の図のようになっている。



簡単のために比較の回数 $T(n)$ の上界を求めてみよう。Step 1 では、5 個の数の整列を行

なうが高々8回の比較で行なえる。すなわち、Step 1では$8 \times (n/5)$の比較で十分である。Step 2では再帰的に中央値を求めているので$T(n/5)$回の比較ですむ。Step 3では、グループは既に整列されているので高々2回の比較で行なえ、総合で$2 \times (n/5)$回ですむ。Step 4での再帰では、上の図のAまたはBの部分が除かれ、少なくとも$(\frac{1}{2} \times \frac{3}{5})n$個の数が不必要となり、再帰の部分は$T(7n/10)$回の比較ですむ。よって

$$\begin{aligned} T(n) &\leq 8 \times (n/5) + T(n/5) + 2 \times (n/5) + T(7n/10) \\ &= 2 \times n + T(n/5) + T(7n/10) \end{aligned}$$

をとなるが、これを解くと

$$T(n) \leq 20 \times n$$

を得る。ここでは、比較の回数の上限を求めているが、$O(n)$でデータの管理等すべてのことを実現することができる。また、原論文では、グループを15個の数に分けたが、ここでは5個としたことを付記しておく。

アルゴリズム高速化技法： **枝刈探索法** (prune-and-search)

上記アルゴリズムのように、問題を解決する際に不必要な部分を取り除くことで効率化をはかる技法。サイズnの問題に対して$P(n)$時間で一定の比率r以上の不要な部分が取り除けるならば、計算時間$T(n)$は、

$$T(n) \leq P(n) + T((1-r)n)$$

となる。もし$P(n) = O(n)$ならば、

$$T(n) \leq cn + c(1-r)n + c(1-r)^2 n + \cdots \leq (c/r)n$$

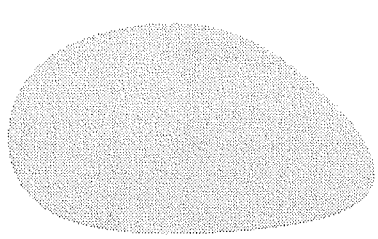
となり$T(n) = O(n)$となる。もし$P(n) = O(n \log n)$ならば、

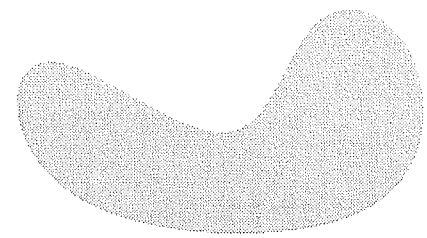
$$T(n) \leq cn \log n + c(1-r)n \log(1-r)n + c(1-r)^2 n \log(1-r)^2 n + \cdots \leq (c/r)n \log n$$

で$T(n) = O(n \log n)$となる。

5 凸包問題

まずは、幾つかの基本的な定義を与えよう。d 次元空間 R^d 内の集合 S を考える。S の任意の 2 点を結ぶ線分が S に含まれるとき、S は**凸集合** (convex set) であるという。例えば、（R^2 の場合）下の左側の集合は凸集合であるが、右側の集合は凸集合ではない。





R^d 内に与えられた集合に対して、それを含む最小の凸集合を与えられた集合の**凸包** (convex hull) とよぶ。凸包問題では、与えられるのが n 点からなる集合である。有限個の点集合の凸包は特殊なもので、特に**凸多面体** (convex polytope) とよばれている。2 次元の場合には凸多角形、3 次元の場合には立方体や 3 角錐などのいわゆる '多面体' が凸多面体の例となる。凸包問題は、与えられた有限個の点から得られる凸多面体を求めるわけであるが、具体的には何を求めれば良いのであろうか。その指針を次の定理が与える。

定理 5.1 凸多面体 $P \subseteq R^d$ を次のように表現する有限個の不等式が存在する

$$P = \{x = (x_1, \cdots, x_d) \in R^d \mid a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{id}x_d \leq b_i \ (i = 1, \cdots, m)\}.$$

これらの不等式は、2 次元の場合には凸多角形の辺にそれぞれ対応し、3 次元の場合にはいわゆる '面' に対応する。例えば、

$$\{(0,0,0),(1,0,0),(0,1,0),(0,0,1)\}$$

の凸包である 3 角錐は

$$\begin{aligned} x_1 + x_2 + x_3 &\leq 1 \\ -x_1 &\leq 0 \\ -x_2 &\leq 0 \\ -x_3 &\leq 0 \end{aligned}$$

という 4 つの不等式で表現できる。

以上のことから凸包問題は以下のように記述できる。

[凸包問題] R^d 内に与えられた n 個の点の凸包の不等式表現を求めよ。

以下で扱う2次元凸包問題では出力が凸多角形であることから、凸多角形の頂点の順序列（反時計回りまたは時計回り）を求めるとしてもよい。

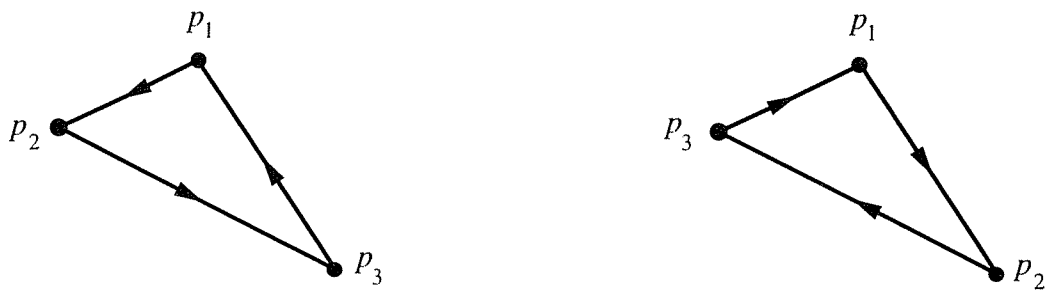
[2次元凸包問題] 2次元空間に与えられた n 個の点に対して、その凸包の頂点の順序列を求めよ。

ここでは、凸多角形の頂点は反時計回りの順に並べるとし、このような順序列はリストを用いて管理をするという前提で話を進める。

5.1 2次元凸包問題

以下では簡単のため、与えられた点集合の3点が同一直線上にのることはないと仮定する。（この仮定がなくとも例外排除の処理を加えればアルゴリズムの時間計算量にはまったく影響しない。）

2次元凸包問題の前に、同一直線上にのらない3点の列 p_1, p_2, p_3 が反時計回りか、時計回りかを判定する方法について簡単に考察しておこう。ここでは、3点の座標 $p_i = (x_i, y_i)$ $(i = 1, 2, 3)$ を用いた判定式を紹介する。これは2次元凸包問題に対するアルゴリズム中で良く用いられる基本的な演算である。



上図の左が反時計回りで、右が時計回りである。反時計回りか時計回りかの判定は、p_1 の位置から p_2 を見たときに p_3 が左側にある場合が反時計回りで、右側にある場合が時計回りである。ベクトル $a = (x_2 - x_1, y_2 - y_1)$ に対し、$b = (-y_2 + y_1, x_2 - x_1)$ はこれと直行し原点から a 方向を見て b は左側にある。よって、方向ベクトル $(x_3 - x_1, y_3 - y_1)$ とベクトル b の内積が正なら p_1, p_2, p_3 は反時計回り、負なら時計回りとなる。すなわち、

$$(x_3 - x_1)(y_1 - y_2) + (y_3 - y_1)(x_2 - x_1) > 0 \iff p_1, p_2, p_3 : \text{反時計回り}$$

$$(x_3 - x_1)(y_1 - y_2) + (y_3 - y_1)(x_2 - x_1) < 0 \iff p_1, p_2, p_3 : \text{時計回り}$$

となる。四則演算と大小比較のみで判定できるので、一定時間でこの判定は終了する。

以下では2次元凸包問題に対する2つのアルゴリズムを紹介する。1つは分割統治法を用いたアルゴリズムで、もう1つは逐次添加法を用いたもの（1節のアルゴリズム2）である。

分割統治法を用いたアルゴリズム

[アイデア] n 点を個数の差が1以下になるように2つに分割する。それぞれの凸包を求め、これらを統合して元の点集合の凸包を構成する。n 点に対して凸包を求める時間計算量を $T(n)$ とすると統合する部分を $O(n)$ 時間でできるならば、

$$T(n) \leq cn + 2T(n/2)$$

という関係式から $T(n) = O(n \log n)$ を得る。よって2つの凸包の統合が $O(n)$ 時間でできるかが問題となる。ここでは簡単のために、n 点は x- 座標の昇順に整列されているとし、分割も x- 座標の小さいものと大きいものに2分割する。整列に必要な時間は $O(n \log n)$ であるから、この分の計算時間を加えても全体の時間計算量 $O(n \log n)$ には影響しない。（整列をしなくても同様の時間計算量は得られるが、リストより複雑なデータ構造を利用する必要があることを付記しておく。）以上をまとめると次のように記述できる。

分割統治法を用いたアルゴリズム

Step 1. 与えられた点を x- 座標の順に並び換える。

Step 2. x- 座標の順に点集合を2分割しそれぞれの点集合の凸多角形を再帰的に求める。

Step 3. 2つの凸多角形を統合し1つの凸多角形を構築する。

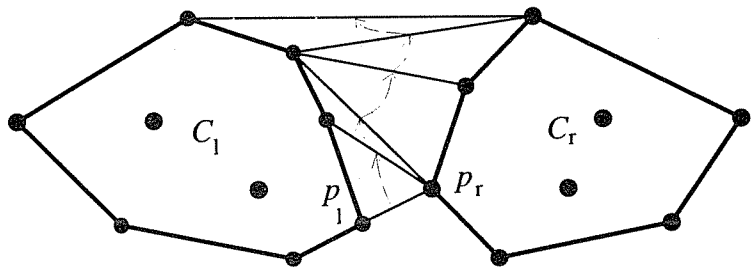
Step 2 では再帰的に凸多角形を求めるが整列の操作は最初に1度だけ行なえばよく、以降の再帰のときには必要ない。Step 3 の部分を詳しく説明しよう。2つの点集合は x- 座標の大小で分割されているので、2つの凸多角形は交わらない。これらを統合するには、2本の共通外接線を求めればよい。左側の凸多角形を C_l、右側の凸多角形を C_r とし、C_l の最右端の頂点を p_l、C_r の最左端の頂点を p_r とする。上部共通外接線は以下の方法で求めることができる。

上部共通外接線の求め方（下図参照）

$v_l := p_l$, $v_r := p_r$ とし (a),(b) を直線 $\overline{v_l v_r}$ が上部共通外接線になるまで繰り返す

(a) 直線 $\overline{v_l v_r}$ が C_l の上部外接線になるまで v_l を反時計回りに移動する

(b) 直線 $\overline{v_l v_r}$ が C_r の上部外接線になるまで v_r を時計回りに移動する



直線 $\overline{v_l v_r}$ が C_l の上部外接線であるかは、v_l の C_l での反時計回りで次の頂点 v' を用いて、点列 v_r, v_l, v' が反時計回りとなるかで判定できる。C_r についても同様に外接線の判定は行なえる。下部共通外接線についても頂点を逆回りにたどることで求めることができ、共通外接線が 2 本求まった時点で頂点の順序列を更新する。以上の操作は 2 つの凸多角形の頂点の個数に比例する時間があれば十分であるから $O(n)$ 時間で終了する。

逐次添加法を用いたアルゴリズム

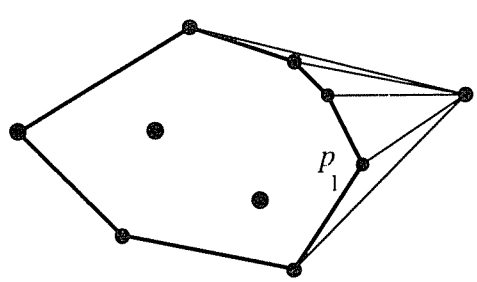
凸多角形に次々に点を追加しながら凸多角形を更新する方法で以下のように記述できる。

逐次添加法を用いたアルゴリズム

Step 1. 与えられた点を x- 座標の順に並び換える。

Step 2. 左端 3 点からなる三角形から x- 座標順に点を順次追加し凸多角形を変更する。

追加される点は常に凸多角形の外側に存在する。凸多角形の更新は追加点を通る古い凸多角形の接線を 2 本求めればよい。先の方法を下図のように利用する。



Step 1 の計算時間は前の議論より $O(n \log n)$ である。以降、Step 2 について考える。k 番目の点を加えるときに古い凸多角形の頂点数に比例する時間があれば十分で、計算時間は明らかに $O(k)$ である。単純に考えれば Step 2 全体の計算時間は $O(\sum_{k=3}^{n} k) = O(n^2)$ となるが、正確に解析すると Step 2 の計算時間は次のように $O(n)$ となる。k 番目の点を加えるときに古い凸多角形の頂点で調べられた頂点の個数を m_k とする。このうち $m_k - 2$ 個（接点の 2 点を除いたもの）が除かれ、新しい凸多角形の内部に入り、以後の操作では無視される。すなわち、$\sum_{k=3}^{n}(m_k - 2) \leq n$ が成り立ち、Step 2 の計算時間は $\sum_{k=3}^{n} m_k$ に比例するので Step 2 の計算時間は $O(n)$ となる。注目すべきは、整列に $O(n \log n)$ を要しているが、その後の処理は $O(n)$ 時間しか必要としない。逐次添加法は単純な方法でこのように効率的な結果が得られることの方が稀である。

5.2 高次元の凸包問題

最後に、高次元の凸包アルゴリズムについて簡単に述べる。2 次元の場合には与えられた n 点に対して出力のサイズ（不等式数や点の順序列の長さ）は高々 n である。3 次元の場合には、多面体の頂点数 v、辺数 e、面数 f の間に Euler の公式

$$v - e + f = 2$$

が成り立つ。さらに、各辺はちょうど 2 つの面に含まれそれぞれの面は 3 辺以上含むことより

$$2e \leq 3f$$

が成り立ち、これらより

$$f \leq 2v - 4$$

を得る。すなわち、出力サイズが入力サイズの高々 2 倍なので 2 次元と同様に $O(n \log n)$ 時間アルゴリズムが存在してもおかしくはない。事実、分割統治法を用いた $O(n \log n)$ 時間アルゴリズムが提案されている [6]。

4 次元以上では、出力サイズ（不等式の数）が $n^{\lfloor d/2 \rfloor}$ に比例した数となることがある。アルゴリズムでは $O(n \log n + n^{\lfloor d/2 \rfloor})$ 時間アルゴリズムの存在が知られている（[9] 参照）。

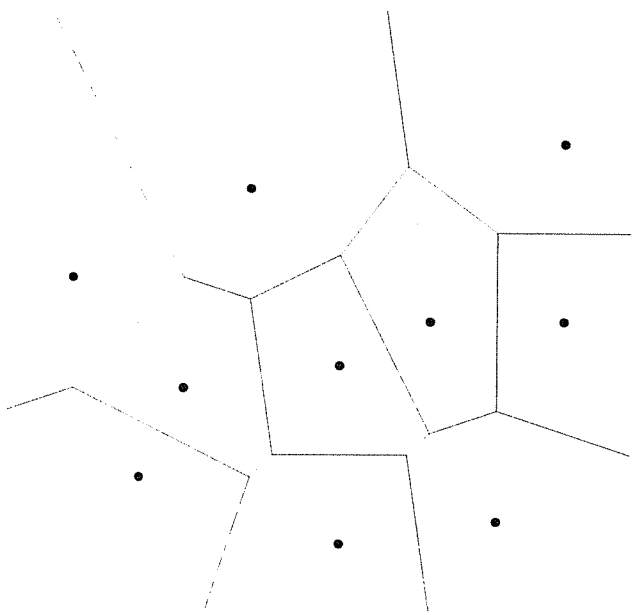
6　ボロノイ図構成問題

ボロノイ図構成問題は多次元の場合にも考えられるが、ここでは平面上に与えられた点集合に対するボロノイ図について考える。議論を簡単にするために、

非退化仮定　与えられた点集合のどの4点も同一円周上にのらない

と仮定する。（この仮定は繁雑さを除去するためのもので構成アルゴリズムの正当性や計算量には影響しない。）

$P = \{p_1, p_2, \cdots, p_n\}$ を平面上に与えられた点集合とする。ボロノイ図ではこれらの点を**母点**とよぶ。各母点 p_i に対する**ボロノイ領域**とは、他の母点よりも p_i が直線距離で近い点からなる領域のことをいう。隣接するボロノイ領域の境界線を**ボロノイ辺**、3本以上のボロノイ辺の交点を**ボロノイ点**とそれぞれよぶ。P に対する**ボロノイ図** (Voronoi diagram) とは、（$V(P)$ と書くことにする）、ボロノイ線によって平面上に描かれる図形のことをいい、例えば下図のようなものである。



ボロノイ辺上の点は2つの母点から等距離にあるので、2つの母点の垂直2等分線の一部分である。ボロノイ点は3本以上のボロノイ辺の交点なので3つ以上の母点と等距離である。さらに、非退化仮定のもとではボロノイ点はちょうど3つの母点と等距離となり、ちょうど3本のボロノイ辺の交点である。言い換えると、ボロノイ点は直線距離が最短となる3つの母点のつくる三角形の外接円の中心（外心）となる。また、これら3母点以外の母点はこの外接円の外側に存在している。以上のようなボロノイ図の性質を以下で簡単にまとめておく。

性質 6.1 非退化仮定のもとで、各ボロノイ点はちょうど3つのボロノイ辺の交点である。

ボロノイ点 v に対して、v を中心とし、最短距離の母点を円周上に含む円を $C(v)$ と書く。

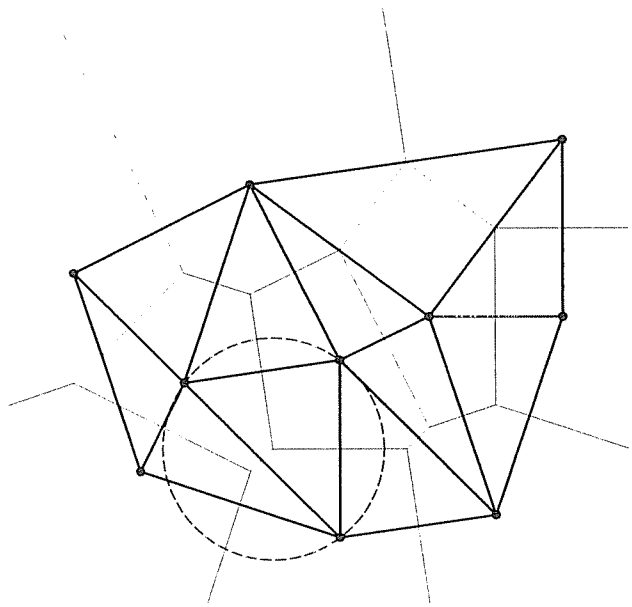
性質 6.2 任意のボロノイ点 v について、円 $C(v)$ の内部は P の点を含まない。非退化仮定のもとでは、$C(v)$ の円周上にはちょうど3つの母点がのる。

ボロノイ領域がボロノイ辺で接する母点同士を辺で結ぶことでできる図形 $D(P)$ を考える。

性質 6.3 $D(P)$ は P の凸包である凸多角形の辺を含み、この凸多角形の分割になっている。半直線となるボロノイ辺は、この凸多角形の辺に対応している。

性質 6.4 非退化仮定のもとでは、$D(P)$ は P の凸包を母点を頂点とする三角形に分割する。

この三角形分割を**ドローネ三角形分割** (Delaunay traingulation) という。下図の太線からなる図は先程のボロノイ図に対するドローネ三角形分割である。



性質 6.5 ドローネ三角形分割の各三角形の外接円の内部は母点を含まない。逆に、3つの母点を通る円の内部が他の母点を含まないならば、この3母点からなる三角形はドローネ三角形分割に含まれる。(上図の破線の円参照。)

これはドローネ三角形分割(ボロノイ図)を構成するときに用いる基本的な性質となる。

性質 6.6 非退化仮定を満たす n 点の集合 P のドローネ三角形分割 $D(P)$ を考える。P の凸包が h 角形のとき、$D(P)$ の三角形の個数は $2n-h-2$、辺の本数は $3n-h-3$ である。

証明）：辺数を e、三角形の個数を t とすると次の *Euler* の公式

$$n-e+(t+1)=2$$

が成立し（外側の h 角形も 1 つの面と数えるから）、さらに各辺は三角形か外側の h 角形のうちのちょうど 2 つに含まれるので

$$2e=3t+h$$

が成立する。これらを解くと主張が示せる。 //

ドローネ三角形分割の定義より、ボロノイ図 $V(P)$ のボロノイ辺たちと $D(P)$ の辺たちには 1 対 1 対応がある。また非退化仮定のもとでは、ボロノイ点たちと三角形たちの間に 1 対 1 対応がある。性質 6.6より、母点数に関してボロノイ辺の本数もボロノイ点の個数も定数倍で押さえられる。このことから、ボロノイ図からドローネ三角形分割を構成することもドローネ三角形分割からボロノイ図を構成することも $O(n)$ 時間で行える。例えば、ドローネ三角形分割からボロノイ図を求めるためには、すべての三角形を求めることを効率的に行う必要がある。このためには、$D(P)$ の辺を単に集合として持っているだけでは駄目で、各母点に接続する辺を反時計周りの順に（リストを用いて）記憶しておくと三角形を $O(n)$ 時間ですべて見つけることができる。各三角形の外心は一定時間で求められることは明かであろう。

以下では、非退化仮定を満たす点集合 P に対してドローネ三角形分割 $D(P)$ を構成する分割統治法を用いたアルゴリズムを紹介する。アルゴリズムの大枠は以下のようになる。

分割統治法を用いたアルゴリズム

Step 1. 与えられた点数が 5 以下なら直接ドローネ三角形分割を構成する。6 以上なら点集合を x-座標の小さいもの P_1 と大きいもの P_2 に 2 等分し（点数の違いは高々 1）、以下を実行する。

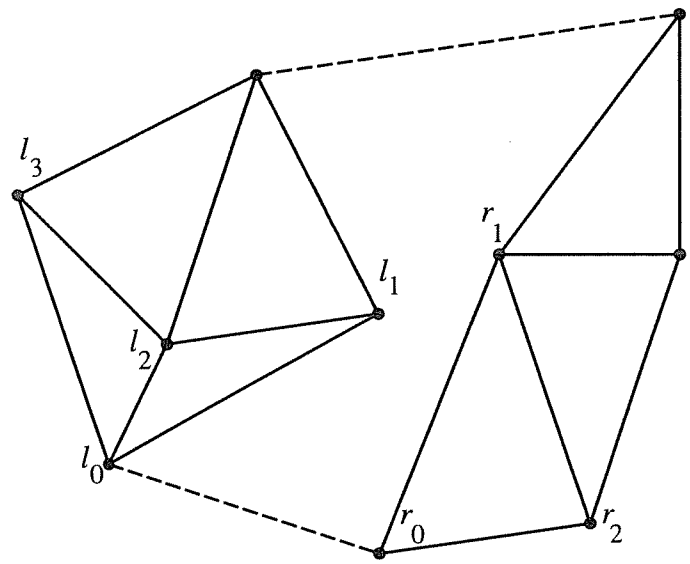
Step 2. P_1 と P_2 のドローネ三角形分割を再帰的に求める。

Step 3. 2 つのドローネ三角形分割を統合し 1 つのドローネ三角形分割を構築する。

Step 1 で点数が 6 以上なら、x-座標の中央値を求め、座標により分割すれば、$O(n)$ 時間で行える。また、y-座標も考慮することで P_1 と P_2 の凸包が交わらないようにすることも可

能なので以下ではこれを仮定する。点数が5以下ならば、適当な方法で一定時間でドローネ三角形分割を構成することができる。凸包を求める分割統治法の時間計算量の解析と同様に、Step 3が$O(n)$時間で行えれば、全体の時間計算量は$O(n\log n)$となる。以下では、2つのドローネ三角形分割の統合の仕方を説明する。

性質6.3より、$D(P)$は、P_1の凸包とP_2の凸包の2つの共通外接線に対応する辺を含む。前節の凸包を求める分割統治法を応用することで、これら2本の辺は$O(n)$時間で求めることができる（下図参照）。



下部共通外接線のP_1の接点をl_0、P_2の接点をr_0とし、$D(P_1)$でl_0に隣接する頂点を反時計周りに$l_1, l_2, \cdots$と名前を付け、$D(P_2)$でr_0に隣接する頂点を時計周りに$r_1, r_2, \cdots$と名前を付ける。$D(P)$には辺(l_0, r_0)を含む三角形が存在するはずである。その三角形のもう一つの頂点については、以下のような性質が成り立つ。

性質 6.7 (l_0, r_0)を含む$D(P)$の三角形の残りの頂点は$\{l_1, l_2, \cdots, r_1, r_2, \cdots\}$に含まれる。

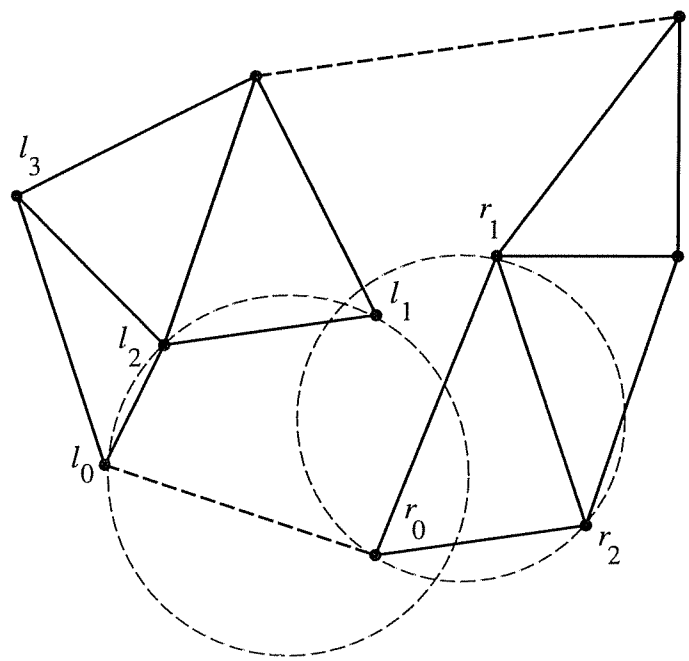
証明の概要）：$D(P_1)$がドローネ三角形分割より、l_0と$l \in P_1 \setminus \{l_0, l_1, l_2, \cdots\}$を結ぶ線分を直径とする円は$\{l_1, l_2, \cdots\}$の点を含むことが示せる。性質 6.5より、$l_0, r_0, l$を頂点とする三角形はドローネ三角形分割に含まれない。$D(P_2)$側についても同様のことがいえる。　／／

では、どのように(l_0, r_0)を含む三角形のもう一つの点を効率的に求めれば良いだろうか。例えば、性質 6.5より、三角形$\triangle l_0 l_1 l_2$の外接円がP_2の点を含んでいたら辺$l_0 l_1$は全体のドローネ三角形分割には現れない。ただし、この方法では1つの三角形に対して反対側の頂点集合の点との比較を行う必要があり非効率である。そこで次の性質を用いる。

性質 6.8 三角形 $\triangle l_0 r_0 l_i$ が $D(P)$ に含まれるなら、三角形 $\triangle l_0 l_k l_{k+1}$ $(k = 1, \cdots, i-1)$ の外接円は r_0 を含み、$\triangle l_0 l_i l_{i+1}$ は r_0 を含まない。

証明）：$i \geq 2$ とする。仮定と性質 6.5より $\triangle l_0 r_0 l_i$ の外接円は l_{i-1} を含まない。さらに、辺 (l_0, l_i) を延長した直線に対して l_{i-1} と r_0 は同じ側に存在するので、$\triangle l_0 l_{i-1} l_i$ の外接円に r_0 は含まれる。$i \geq 3$ に対しては以下の操作を繰り返すことで題意を証明できる。$D(P_1)$ がドローネ三角形分割であることより、$\triangle l_0 l_{i-1} l_i$ の外接円には l_{i-2} は含まれないが、r_0 は含まれている。辺 (l_0, l_{i-1}) を延長した直線に対して、l_{i-2} と r_0 は同じ側に存在するので、$\triangle l_0 l_{i-2} l_{i-1}$ の外接円は r_0 を含む。辺 (l_0, l_i) は $D(P)$ に含まれるので、$\triangle l_0 l_i l_{i+1}$ は r_0 を含まない。 ／／

同様の操作を $D(P_2)$ にも施すと、前例の 2 つのドローネ三角形分割に対しては下図のように辺 (l_0, l_1) が除去され、l_2 か r_1 が $l_0 r_0$ と三角形を作る候補となる。図では、$\triangle r_0 r_1 r_2$ の外接円は l_1 を含むが、上記の判定では辺 (r_0, r_1) はこの段階では排除されない。$\triangle l_0 r_0 l_2$ の外接円が r_1 を含まないことより 2 つの候補から l_2 が選ばれる。



上記の操作では、$l_0 r_0$ が外接線であるという情報は使っていない。用いている条件は、l_0 周りに $r_0, l_1, l_2, \cdots$ が反時計回りに並び、r_0 周りに $l_0, r_1, r_2 \cdots$ が時計回りに並んでいることと、$D(P_1), D(P_2)$ の三角形が性質 6.5を満たすことだけである。すなわち、辺 (l_2, r_0) に対しても同様の操作が適用でき、(l_1, r_0) の追加、(r_0, r_1) 削除と (l_1, r_2) 追加、(l_1, r_1) 追加などと操作が進む。

上記の操作において、1辺を加えるために必要な1円と1点の位置関係の判定の回数は（除かれる辺の本数＋2）＋（加えられる1辺に対して2）となる。性質 6.6より、除かれる辺の総数も加えられる辺の総数も n の定数倍であり、この統合の操作は $O(n)$ 時間で終了する。すなわち、全体としてこの分割統治法を用いたアルゴリズムの時間計算量は $O(n \log n)$ である。

7 問題の難しさ

2つの問題 $\mathcal{P}_1$ と $\mathcal{P}_2$ に対して、

1. $\mathcal{P}_1$ の入力 I_1 を $\mathcal{P}_2$ の入力 I_2 に変換する。
2. I_2 に対して $\mathcal{P}_2$ を解く。
3. 解いた答を I_1 に対する $\mathcal{P}_1$ の解に変換する。

という関係が存在するとき、問題 $\mathcal{P}_1$ は問題 $\mathcal{P}_2$ に**変換可能** (transformable) であるという。特に、入力サイズ n に対して、1 と 3 のステップが $O(\tau(n))$ 時間で行えるとき、$\tau(n)$ 変換可能であるといい、$\mathcal{P}_1 \propto_{\tau(n)} \mathcal{P}_2$ と書くことにする。定義から変換可能性は推移律を満たしている。

例えば、整列問題の入力 $I_1 = \{x_1, \cdots, x_n\}$ が与えられたとき、$I_2 = \{(x_1, x_1^2), \cdots, (x_n, x_n^2)\}$ という平面上の点集合を構成する。これらの点は図のように2次関数 $y = x^2$ 上にあるので、I_2 に対する2次元凸包問題を解くと、I_2 のすべての点は凸多角形の頂点となり、これらは反時計周りに並んで出力される。x-座標の最小の点を探し、反時計周りに凸多角形の頂点をたどると x-座標は単調に増加する。すなわち整列問題が解けることになり、整列問題は2次元凸包問題に（線形時間で）変換可能である。

2次元凸包問題の入力 P に対して、そのままの点集合をドローネ三角形分割問題の入力とすると、性質 6.3より P のドローネ三角形分割から P の凸包を $O(n)$ 時間で構成できる。すなわち、2次元凸包問題はドローネ三角形分割問題に（線形時間で）変換可能である。

問題の変換可能性は一般的には対称的ではない。前節で述べたように、ボロノイ図からドローネ三角形分割の構成とその逆はともに $O(n)$ 時間で構成可能である。この例のように

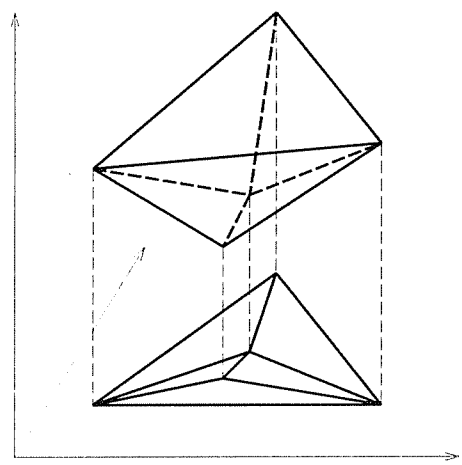
$$\mathcal{P}_1 \propto \mathcal{P}_2 \text{ かつ } \mathcal{P}_2 \propto \mathcal{P}_1$$

であるとき、$\mathcal{P}_1$ と $\mathcal{P}_2$ は等価 (equivalent) であるという。ボロノイ図構成問題とドローネ三角形分割問題は等価である。

問題の変換に関する例としてもう一つ紹介しておこう。ドローネ三角形分割問題の入力 $I_1 = \{(x_1, y_1), \cdots, (x_n, y_n)\}$ に対して $I_2 = \{(x_1, y_1, z_1), \cdots, (x_n, y_n, z_n)\}$、ただし $z_i = x_i^2 + y_i^2$ を考える。ドローネ三角形分割で $(x_i, y_i), (x_j, y_j), (x_k, y_k)$ を頂点とする三角形の外接円の式を $(x-a)^2 + (y-b)^2 = r^2$ とすると、

$$\begin{aligned} -2ax_h - 2by_h + x_h^2 + y_h^2 &= -2ax_h - 2by_h + z_h &&= r^2 - a^2 - b^2 \quad (h = i, j, k) \\ -2ax_h - 2by_h + x_h^2 + y_h^2 &= -2ax_h - 2by_h + z_h &&\geq r^2 - a^2 - b^2 \quad (h \neq i, j, k) \end{aligned}$$

すなわち、1 次不等式 $-2ax - 2by + z \geq r^2 - a^2 - b^2$ は、I_2 の凸包である 3 次元凸多面体を定義する不等式で 3 点 $(x_i, y_i, z_i), (x_j, y_j, z_j), (x_k, y_k, z_k)$ がのるものである。この面は凸多面体の下側にある。逆に、この 3 次元凸多面体の下側の面はドローネ三角形分割の三角形に対応していて、下図のように 3 次元凸多面体の下側の面を xy 平面に射影したものがドローネ三角形分割と一致する。詳しいことは省くが、ドローネ三角形分割問題は 3 次元凸包問題に（線形時間で）変換可能である。



以上をまとめると

$$\text{整列問題} \ \propto_n \ \text{2 次元凸包問題} \ \propto_n \ \begin{matrix}\text{ドローネ三角形分割問題} \\ \text{ボロノイ図構成問題}\end{matrix} \ \propto_n \ \text{3 次元凸包問題}$$

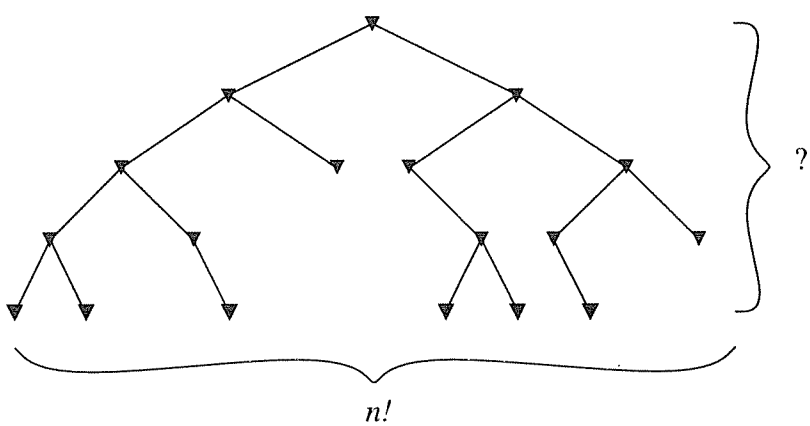
が成立する。前節まででこれらすべてに$O(n\log n)$時間アルゴリズムが存在することを示した。変換可能性の定義から以下の事実はほぼ自明である。

定理 7.1 $\mathcal{P}_1 \propto_{\tau(n)} \mathcal{P}_2$ であるとき、以下のように時間計算量の上界や下界が算定できる。

[上界] $\mathcal{P}_2$ が$T_2(n)$時間で解けるなら、$\mathcal{P}_1$ は$T_2(n)+O(\tau(n))$時間で解ける。
[下界] $\mathcal{P}_1$ を解くのに$T_1(n)$時間必要ならば、$\mathcal{P}_2$ を解くには$T_1(n)-O(\tau(n))$時間必要である。

上記のすべての問題に対して$O(n\log n)$時間アルゴリズムが存在することを知ってしまった今となっては、この定理を用いて上界を算定することは面白くはない。しかし、この定理と整列問題には$\Omega(n\log n)$時間が必要であるという事実を用いて、これらの問題を解くにはどうしても$\Omega(n\log n)$時間を必要とすることが示せるのは興味深いことである。すなわち、ここで紹介したアルゴリズムは下界を実現するという意味で最適な時間計算量をもつアルゴリズムである。一般に問題の難しさの下界を示すことは難しい。なぜならば、すべてのアルゴリズムに対して下界以上の時間を必要とすることを示す必要がある。整列問題については、$\Omega(n\log n)$という下界が比較的簡単に示せる。このことを最後に説明しよう。

ここでは簡単のために、整列問題の入力数列には同じ数が含まれないと仮定する。長さnの数列の要素の数値に応じて、整列の答えは$n!$通りある。本講座で考えている計算機モデルでは、数の大小比較のみを用いて整列を行っている。すなわち、どんなソーティングアルゴリズムも数列の要素の大小比較のみで、$n!$通りの可能性の中から正しい答えを求めている。任意のソーティングアルゴリズムの動きは次のような（系統木に似た）木構造で表現することができる。



この木構造の点は、アルゴリズムの動いているときの時点を意味し、特に頂上の点は入力がされた時点を意味している。アルゴリズムは各時点で2つの数x_iとx_j（添字は$i<j$とす

る）を比較し、$x_i < x_j$ と $x_i > x_j$ の場合に応じて処理を分ける。木構造では、この時点に対応した点から $x_i < x_j$ ならば左下に進み、$x_i > x_j$ ならば右下に進むとする。すなわち、木構造で下の点に進むことは、x_i と x_j を比較する前に可能性のある答えの集合を、$x_i < x_j$ であるものと $x_i > x_j$ であるものに2分割し、選択した方に進むことを意味する。下のない点は、答えが見つかったことを意味し、ここでは可能性のある答えはただ一つに定まっていなければならない。また、アルゴリズムの最悪時の時間計算量は、この木構造の深さになっている。

この木構造はちょうど $n!$ 個の葉（下に点のない点）を持ち、各点は高々2つの子（すぐ下の点）しか持たない。このような条件のもとで最も深さを浅くするには、各頂点でできる限り2つの子を持つようにし、左右のバランスの取れたものである。理想的な木構造では、深さ0の点は1つ、深さ1の点の個数は2つ、一般に深さ k の点の個数は 2^k となる。すなわち、葉が $n!$ 個ある木構造の深さ d は

$$d \geq \log_2(n!) \geq \log_2((n/2)^{n/2}) = \Omega(n \log n)$$

となり、整列問題の下界 $\Omega(n \log n)$ を得る。

8 おわりに

本講座では、計算幾何学、計算量理論、アルゴリズム理論の基礎的なところを紹介した。これより先については、以下にあげる和書を参考とされたい。

参考文献

[1] 浅野哲夫, 計算幾何学, 朝倉書店, 1990.

[2] D. Avis, 今井浩, 松永信介, 計算幾何学・離散幾何学, 朝倉書店, 1994.

[3] M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest and R.E. Tarjan, "Time bounds for selection," J. Computer and System Science, **7**, 448–461, 1972.

[4] H. Edelsbrunner 著／今井浩・今井桂子 訳, 組合せ幾何学のアルゴリズム, 共立出版, 1987／1995.

[5] 伊理正夫 監修／腰塚武志 編集, 計算幾何学と地理情報処理（第2版）, 共立出版, 1993.

[6] F.P. Preparata and S.J. Hong, "Convex hulls of finite sets of points in two and three dimensions," Comm. ACM, **20**, 87–93, 1977.

[7] E.P. Preparata and M.I. Shamos 著／浅野孝夫・浅野哲夫 訳, 計算幾何学入門, 総研出版, 1985 ／ 1992.

[8] 杉原厚吉, “幾何アルゴリズムの位相優先設計法,” （室田一雄 編）, 離散構造とアルゴリズム III, 近代科学社, 35–76, 1994.

[9] 徳山豪, “計算幾何学と組合せ論,” （藤重悟 編）, 離散構造とアルゴリズム I , 近代科学社, 1–55, 1992.