Vol.2014-AL-147 No.13
2014/3/4

既存点までの距離誤差を最小にする点位置発見アルゴリズム

中村 茂幹[1,a)]　浅野 哲夫[1,b)]　Siu-Wing Cheng[2,c)]

概要：平面に n 個の点が配置されているときに，新たな 1 点を挿入する問題を考える．新たに挿入する点と既に配置された各点との間には距離が指定されており，新たな点を，既存点までの距離誤差の総和 (正確には距離を 2 乗した誤差の総和) が最小になる位置へ挿入したい．本稿では，この問題を円のアレンジメントなど幾何的な性質を利用して解く 2 つのアルゴリズムを提案する．1 つは，$O(n^2 \log n)$ 時間，$O(n^2)$ 領域で解くアルゴリズム，もう 1 つは，$O(n^3)$ 時間，$O(n)$ 領域で解くアルゴリズムである．

キーワード：アルゴリズム，計算幾何学，円のアレンジメント

An Algorithm for Finding the Point Minimizing the Distance Error to Given Points

Shigeki Nakamura[1,a)]　Tetsuo Asano[1,b)]　Siu-Wing Cheng[2,c)]

Abstract: Given n points in the plane, we want to insert a new point in the distance specified by the input from each existing point. An optimal point is the one that minimizes the sum of squared errors. In this paper, we present two efficient algorithms for solving this problem using geometric properties such as an arrangements of circles. One runs in $O(n^2 \log n)$ time and $O(n^2)$ space and the other in $O(n^3)$ time and $O(n)$ space.

Keywords: algorithms, computational geometry, arrangements of circles

1. はじめに

各個体間の類似性や非類似性を数量化したデータを基に，各個体を平面上の点として配置し，視覚的に表すことは重要であり，現在までさまざまな研究が行われてきた．この問題を解く手法は多次元尺度構成法 (Multidimensional Scaling, MDS) と呼ばれ，個体間の関連性を表すデータが与えられたときに，各個体をユークリッド空間内に似ているものどうしを近くに, 似ていないものどうしは遠くに配置することにより，各個体間の関連性を視覚的に表現する方法である [1].

ここで，各個体が既に平面上に点として配置されており，その中に新たな個体を 1 つ点として挿入する問題を考える．このとき，既存点の配置は変化させない．配置を変化させない理由は，新たな固体に着目して，既に配置された個体との関連性を視覚的に表現したいためである．

この問題を解く 1 つの方法は，既に配置されている固体と新たな固体に対して MDS を適用して，全体を平面へ配置する方法である．しかし，MDS は対象全体を平面に配置する方法である．新たな個体を挿入する場合は，既に配置された点の位置まで変化する可能性があるので，この方法は好ましくない．そこで，この問題を解くアルゴリズムの開発が必要とされる．

本稿では非類似性を数量化したデータとして距離を扱う．新たな固体と既に配置された固体との間には距離が指定されており，その指定された距離と実際に挿入した点間との

[1] 北陸先端科学技術大学院大学 情報科学研究科
Graduate School of Information Sciences, JAIST, Ishikawa, Japan
[2] Department of Computer Science and Engineering, HKUST, Hong Kong
[a)] snakamura@jaist.ac.jp
[b)] t-asano@jaist.ac.jp
[c)] scheng@cse.ust.hk

距離誤差の総和が最小になる点の挿入位置を発見したい．

本稿の目的は，まず，既存点までの距離誤差を最小にする点位置発見問題を定義することである．そして，この問題を効率的に解くアルゴリズムを開発することである．

本稿の結果として，この問題を解くアルゴリズムを 2 つ開発した．1 つは，$O(n^2 \log n)$ 時間，$O(n^2)$ 領域で解くアルゴリズム，もう 1 つは $O(n^3)$ 時間，$O(n)$ 領域で解くアルゴリズムである．

2. 問題の定義

本稿で取り組む問題を以下のように定義する．

定義 2.1. 平面上に n 個の点 $p_i\,(i = 1, \ldots, n)$ が既に配置されている．ここに新たな点 q を挿入したい．点 q と既に配置された各点との間には距離 δ_i が指定されている．そのとき，それぞれの p_i と q のユークリッド距離 $d(p_i, q)$ が，指定された δ_i となるべく近くなる最適な位置を求める (図 1)．この問題を既存点までの距離誤差を最小にする点位置発見問題と呼ぶ．

具体的には，指定された距離を 2 乗したものとユークリッド距離を 2 乗したものとの誤差の総和を最小化する問題であり，以下のように定式化できる．

$$\min \quad f(q) = \sum_{i=1}^{n} |d(p_i, q)^2 - \delta_i^2|. \qquad (1)$$

距離を 2 乗して計算している理由は，2 乗しないと平方根が分母に含まれる方程式を解くことになり，最適解を求めることが難しくなるからである．

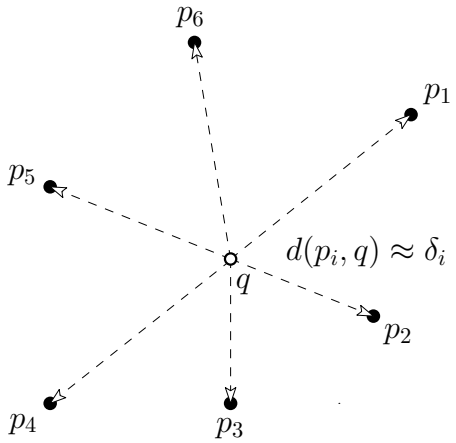

図 1 既存点までの距離誤差を最小にする点位置発見問題．点 p_i と点 q のユークリッド距離 $d(p_i, q)$ が，δ_i となるべく近くなる位置を求める．

3. アルゴリズムの概要

3.1 グレースケール画像を出力する簡便法

図 2 の画像は，平面をピクセル単位で区切り，各ピクセルごとに目的関数の値を計算して，その結果をグレースケール画像として出力したものである．画像の色は黒色に近いほど最適解に近く，白色に近いほど最適解から遠いことを表している．黒丸の点は既存の点，白丸の点は新たな点の最適な挿入位置を表している．

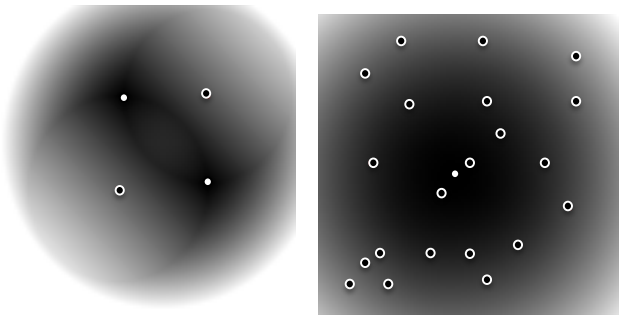

図 2 平面をピクセル単位で区切り，各ピクセルごとに式 (1) の値を出力したグレースケール画像．(a) 既存点が 2 点の場合．(b) 既存点が多数ある場合．

図 2(a) の画像では，既存点を中心とした円が 2 つあるように見える．この円の半径は既存点からの指定された距離 δ_i であり，局所的に距離誤差が小さくなるため，このように円に見える．よって，2 つの円が交わるところが 2 つの既存点からの距離誤差が 0 になる位置であり，最適な挿入位置となる．

このように，全ての円が交わる点が存在すれば，最適解の位置を予想するのは容易である．しかし，図 2(b) のように多数の点が与えられた場合は，問題は単純ではない．図 2(a) の画像と比べ，全体がはっきりしないでぼやけている．どの位置に最適解が存在するかの予想は難しい．

3.2 円のアレンジメントに基づくアルゴリズム

第 3.1 節で示した通り，多数の点が与えられた場合は，最適解の位置を予想するのは難しい．この問題を解くために，円のアレンジメントと呼ばれる幾何的な構造を利用する．円のアレンジメントを次のように定義する．

定義 3.1. C を平面上の n 個の円の集合とする．集合 C によって，平面は頂点，辺，面に分割される．各円の交点が頂点，円弧が辺，辺で囲まれた領域が面である．この平面分割を C によって誘導される円のアレンジメントと呼ぶ．

本稿で提案するアルゴリズムのアイデアは，円のアレンジメントにより平面分割された面を全探索し，各面に対し局所最適解を求めることにより，全体の最適解を求める方法である．

図 3 に円のアレンジメントの例を示す．点 $p_i\,(i = 1, ..., n)$ に対して，点 p_i を中心とする半径 δ_i の円 C_i 集合に対して，円のアレンジメントを構成する．円集合により，平面は頂点，辺，面に分割される．本稿では，縮退はなく，どの 3 つの円も 1 点で交わることはないと仮定する．つまり，必ずどの頂点にも 4 つの辺が接続している．

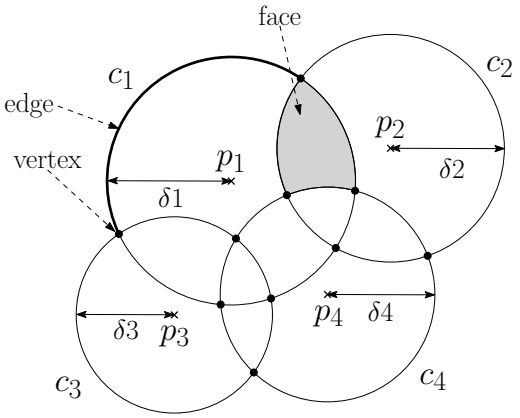

図 3 円のアレンジメント．円集合により，平面は頂点 (vertex)，辺 (edge)，面 (face) に分割される．

円のアレンジメントについて次の補題が成り立つ．

補題 3.2. 円のアレンジメントにより平面分割された頂点，辺，面の最大個数はそれぞれ $O(n^2)$ 個となる．

証明. 頂点，辺，面の個数が最大になるのは，どの 2 つの円も 2 点で交わっているときである．そのとき，2 つの円の組み合わせが $\binom{n}{2}$ 通りあるので，頂点の個数 V は，

$$V = 2 \cdot \binom{n}{2} = 2 \cdot \frac{n(n-1)}{2} = n^2 - n$$

となる．1 つの円周上に交点が $2(n-1)$ 個あるので，円周は $2(n-1)$ 個の辺に分かれる．よって辺の個数 E は，

$$E = 2(n-1)n = 2n^2 - 2n$$

となる．オイラーの多面体定理 $V - E + F = 2$ より面の数 F は，

$$F = n^2 - n + 2$$

となる． □

円のアレンジメントに関する詳しい計算量の説明については [2] を参照されたい．

本稿では，2 つのアルゴリズムを提案する．1 つは 2 重連結辺リストにより，円のアレンジメント全体をデータ構造として蓄える方法であり，第 4 節で説明する．もう 1 つは平面走査法により，円のアレンジメントを部分的に構成する方法であり，第 5 節で説明する．

4. 2 重連結辺リストを用いたアルゴリズム

具体的なアルゴリズムの手順は次の通りである (図 4)．まず 2 重連結辺リスト，双対グラフ，全域木，オイラーグラフを順に構成し，問題を解くために必要なデータ構造を作る．そして，構成したデータ構造をもとに，オイラーツアーにより各面の局所最適解を求め，全体の最適解を求める．第 4.1 節では，問題を解くために必要なデータ構造の構成方法について説明する．第 4.2 節では，オイラーツアーにより各面の局所最適解を求め，全体の最適解を求める方法について説明する．このアルゴリズムの計算量は $O(n^2 \log n)$ 時間，$O(n^2)$ 領域である．各手順の計算量は表 1 に示す．

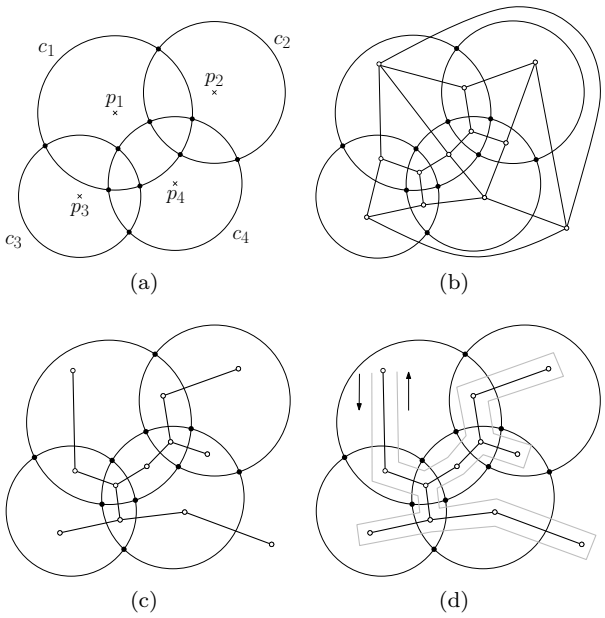

図 4 アルゴリズムの手順．(a)2 重連結辺リスト．(b) 双対グラフ．(c) 全域木．(d) オイラーグラフ．

表 1 2 重連結辺リストを用いたアルゴリズムの各手順の計算量．

手順	時間計算量	作業領域
円のアレンジメントの構成	$O(n^2 \log n)$	$O(n^2)$
双対グラフの構成	$O(n^2)$	$O(n^2)$
全域木の構成	$O(n^2)$	$O(n^2)$
オイラーグラフの構成	$O(n^2)$	$O(n^2)$
オイラーツアーによる面の全探索	$O(n^2)$	$O(n)$

4.1 2 重連結辺リストに基づくデータ構造

この節では問題を解くために必要なデータ構造の構成方法について説明する．

4.1.1 2 重連結辺リストの構成

円のアレンジメントを 2 重連結辺リストと呼ばれるデータ構造を用いて実現する．2 重連結辺リストは，平面分割のそれぞれの面，辺，頂点に対するレコードを含んだデータ構造である．レコードには幾何学的情報と位相情報などが含まれている．2 重辺連結リストの特徴として，すべての頂点を列挙できる，すべての有向辺を列挙できる，任意の面を指定すると，その境界を辿ることができる，ある面から境界辺を共有する隣接面に移ることができる，などがあげられる．

円のアレンジメントはデータ構造として 2 重連結辺リストを用いることで $O(n^2 \log n)$ 時間，$O(n^2)$ 領域で構成できる [3]．2 重連結辺リストの詳しい説明は [4] を参照されたい．

4.1.2 双対グラフの構成

円集合により平面分割された平面グラフに対して，双対グラフを構成する．双対グラフとは，面を頂点に対応させ，2 つの面が辺により隣接しているときに，対応する 2 頂点間に辺を引いてできるグラフのことである．データ構造として面の隣接リストを作成して実現する．2 重連結辺リストは双方向の辺を持つデータ構造である．ある面に対してその面に隣接する面を求めたい場合は，まず，面を構成する各辺に対して対になっている辺を取り出す．そして，その辺が属する面を調べることで求めることができる．

双対グラフを構成するのに必要な計算量は $O(n^2)$ 時間，$O(n^2)$ 領域である．双対グラフを構成する計算時間は，辺あたり $O(1)$ 時間かかり，辺は全部で $O(n^2)$ 本あるので $O(n^2)$ 時間かかる．作業領域は，面の隣接関係を隣接リストに保存し，面の数は全部で $O(n^2)$ 個あるので，$O(n^2)$ 領域必要である．よって，双対グラフを構成するのに必要な計算量は $O(n^2)$ 時間，$O(n^2)$ 領域である．

4.1.3 全域木の構成

双対グラフから全域木を構成する．双対グラフの任意の 1 つの頂点を始点とし，その始点から深さ優先探索を行う．深さ優先探索では，始点から始めて，新たに訪問する頂点がなくなったときに 1 つ前の頂点に戻り，そこからまたできる限り深く訪問するというのを繰り返す．双対グラフの各頂点と，深さ優先探索で訪れた辺により構成されるグラフが全域木となる．

全域木を構成する計算量は $O(n^2)$ 時間，$O(n^2)$ 領域である．深さ優先探索にかかる時間は頂点数 $O(n^2)$ と辺の数 $O(n^2)$ を足した程度であるので計算時間は $O(n^2)$ かかる．作業領域は，最悪頂点の数だけ領域が必要となるので，$O(n^2)$ 領域必要である．よって，全域木を構成する計算量は $O(n^2)$ 時間，$O(n^2)$ 領域である．

4.1.4 オイラーグラフの構成

効率的に全域木をたどるために，ある始点から，すべての辺を 1 回だけたどり始点へと戻る一筆書きの順で，各頂点を訪れたい．そのためにはオイラーグラフを構成してオイラーツアーを持つ必要がある．グラフ理論の結果として，すべての頂点の次数が偶数ならば，そのグラフはオイラーグラフを持つことが知られている．構成した全域木の各辺を双方向の 2 本の有向辺で置き換えることで，オイラーグラフを得る．

オイラーグラフを構成する計算量は $O(n^2)$ 時間，$O(n^2)$ 領域である．深さ優先探索により，訪れた辺を双方向にすればよいので，オイラーグラフを構成する計算量は $O(n^2)$ 時間，$O(n^2)$ 領域である．

4.2 オイラーツアーによる面の全探索

第 4.1 節で構成したデータ構造をもとに，オイラーツアーにより各面を順にたどる．その際，各面の目的関数に対し，その性質に合わせた解法で局所最適解を求める．すべての面の局所最適解を求め，全体の最適解を求める．この節では，各面の局所最適解を効率的に求める方法について説明する．

4.2.1 各面での目的関数の効率的な式の立て方

まず，各面の目的関数を計算するにあたり，$d(p_i, q)$ を計算する必要がある．$d(p_i, q)$ は 2 点間のユークリッド距離なので，点 p_i の座標を (x_i, y_i)，点 q の座標を (x, y) とすると次のように変換できる．

$$d(p_i, q) = \sqrt{(x - x_i)^2 + (y - y_i)^2}.$$

すると，目的関数は次のように変換できる．

$$f(x, y) = \sum_{i=1}^{n} |(x - x_i)^2 + (y - y_i)^2 - \delta_i^2|.$$

次に，絶対値を含む式 $|(x - x_i)^2 + (y - y_i)^2 - \delta_i^2|$ を計算したい．各面ではどの円の内部にありどの円の外部にあるかという位置関係が分かる．円の内部に点 q を挿入した場合は次式のように，

$$|(x - x_i)^2 + (y - y_i)^2 - \delta_i^2| = -(x - x_i)^2 - (y - y_i)^2 + \delta_i^2$$

円の外部に点 q を挿入した場合は次式のように，

$$|(x - x_i)^2 + (y - y_i)^2 - \delta_i^2| = (x - x_i)^2 + (y - y_i)^2 - \delta_i^2$$

絶対値の中身を計算する．

例として，図 5(a) の塗りつぶされた面 $C_1 \cap \overline{C_2} \cap \overline{C_3} \cap \overline{C_4}$ は次のように目的関数の絶対値を外すことができる．

$$\begin{aligned} f(x, y) = & -(x - x_1)^2 - (y - y_1)^2 + \delta_1^2 \\ & + (x - x_2)^2 + (y - y_2)^2 - \delta_2^2 \\ & + (x - x_3)^2 + (y - y_3)^2 - \delta_3^2 \\ & + (x - x_4)^2 + (y - y_4)^2 - \delta_4^2. \end{aligned}$$

次に図 5(b) で塗りつぶされた面 $C_1 \cap \overline{C_2} \cap C_3 \cap \overline{C_4}$ へ移動したい．このとき，面の移動に対し 1 つの円を横切るだけである．横切る円に対してのみ式の符号を変えるだけで，

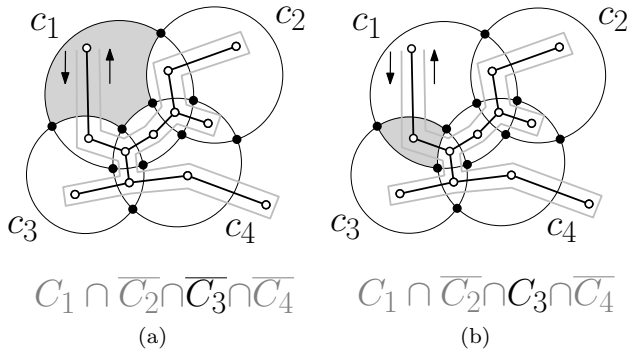

図 5 面の移動に対する各円の位置関係．(a) で塗りつぶされた面から (b) で塗りつぶされた面への移動．

効率的に次の面の式を立てることができる．よって，次のように目的関数の絶対値を外すことができる．

$$\begin{aligned} f(x,y) = & -(x-x_1)^2-(y-y_1)^2+\delta_1^2 \\ & +(x-x_2)^2+(y-y_2)^2-\delta_2^2 \\ & -(x-x_3)^2-(y-y_3)^2+\delta_3^2 \\ & +(x-x_4)^2+(y-y_4)^2-\delta_4^2 \ . \end{aligned}$$

この場合，円 C_3 を横切るだけなので，円 C_3 の部分の符号を変えるだけで良い．

一般に，オイラーツアーにより各面を訪れるとき，1 つの円を横切り次の面に移るので，目的関数の計算は $O(1)$ 時間でできる．始点の面こそ式を立てるのに $O(n)$ 時間かかるが，各面を移動する場合，$O(1)$ 時間で次の面の式を立てることができる．

4.2.2 各面での局所最適解の求め方

目的関数の x と y の 2 次項の係数の符号により，各面に対して，局所最適解を求める手法が変わる．次の 3 つの式の場合に分けられる．

$$f(x,y) = a(x^2+y^2)+bx+cy+d \quad \text{if } a>0, \tag{2}$$
$$f(x,y) = a(x^2+y^2)+bx+cy+d \quad \text{if } a<0, \tag{3}$$
$$f(x,y) = bx+cy+d \quad \text{if } a=0. \tag{4}$$

式 (2) は下に凸な x と y の 2 変数の 2 次関数である．式 (3) は上に凸な x と y の 2 変数の 2 次関数である．式 (4) は平面である．

式 (2) で極値点が面の内部にある場合は極値が局所最適解となる．しかし，その他の場合，局所最適解を求めることは単純ではない．局所最適解は面の境界上に存在し，面を構成するすべての辺，頂点を調べる必要がある．

式 (2) で極値点が面の外部にある場合は極値点から最も近い面の位置が局所最適解となる．そのような候補は，極値点と面を構成する各円の中心を通る直線との交点，または面を構成する頂点である．図 6 の (a) から，(b)，(c)，(d) の順に操作を行い，局所最適解を求める．

式 (3) の場合は極値点から最も遠い面の位置を求めたい．式 (2) で極値点が面の外部にある場合と同様の手順を踏み，局所最適解を求める．

式 (4) の場合，局所最適解の候補の位置を予想するのは難しい．よって，目的関数の平面の式と面を構成する各円との連立方程式 (式 (5)) から極値点を求め，その位置から局所最適解の候補を探す．

$$\begin{cases} ax+by+cz+d=0 \\ (x-e)^2+(y-f)^2=r^2 \end{cases} \tag{5}$$

極値点が面の境界上にあれば，極値が局所最適解の候補となる．極値点が面の境界上になければ，極値点から最も近い面の境界上の位置である他の円との交点が局所最適解の候補となる．面を構成する各円に対してこの操作を行い，局所最適解を求める．

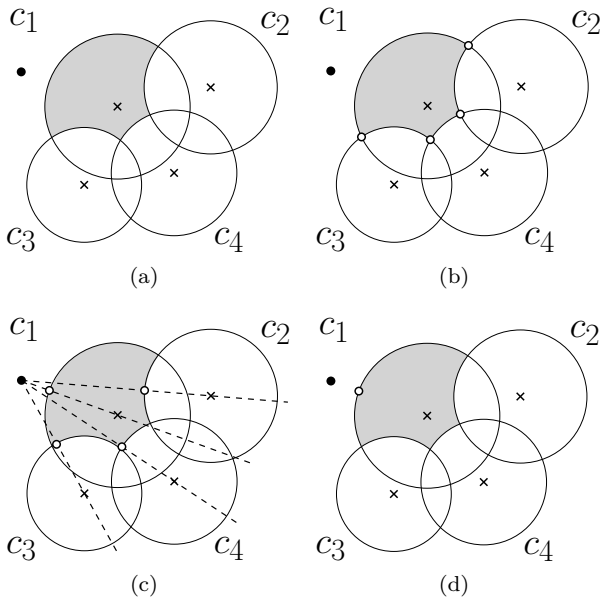

図 6 式 (2) で極値点が面の外部にある場合の局所最適解の求め方．(a) 式 (2) で極値点が面の外部にある場合の例．(b) 面を構成する各円の交点．(c) 極値点と面を構成する各円の中心を通る直線との交点．(d) 局所最適解の位置．

Algorithm 1 オイラーツアーによる面の全探索

Input: オイラーグラフ G と始点 $s \in V$
Output: 最適解 ans およびその座標

```
u ← s
while G に対して始点 s からのオイラーツアーが終了していない do
    u の面に対する目的関数を立てる．
    if 目的関数が式 (2) then
        if 極値点が面の内側にある then
            tmp ← 局所最適解 (極値点)
        else if 極値点が面の外側にある then
            tmp ← 局所最適解 (極値点から最も近い面の位置)
        end if
    else if 目的関数が式 (3) then
        tmp ← 局所最適解 (極値点から最も遠い面の位置)
    else if 目的関数が式 (4) then
        tmp ← 局所最適解 (目的関数と面を構成する各円との連立方程式から求めた位置)
    end if
    if ans > tmp then
        ans ← tmp
    end if
    u ← 次に訪れる頂点 v
end while
```

以上がオイラーツアーによる面の全探索の手順である．オイラーツアーによる面の全探索のアルゴリズムを Algorithm1 に示す．また，補題 3.2 より次の定理が成り立つ．

定理 4.1. オイラーツアーによる面の全探索に必要な計算量は $O(n^2)$ 時間，$O(n)$ 領域である．

証明. 円のアレンジメントにより平面分割された面の数は $O(n^2)$ 個である．よって，すべての面を訪れるのに $O(n^2)$ 時間かかる．また，各面で局所最適解を求めるのに，面を構成する辺と頂点を参照する．辺と頂点の数はそれぞれ $O(n^2)$ 個あり，各辺は 2 つの面に接しているので 2 回ずつ，各頂点は 4 つの面に接しているので 4 回ずつしか参照されない．よって，それぞれ参照される回数は全体で $O(n^2)$ 回に抑えられる．以上より，オイラーツアーによる面の全探索に必要な計算量は $O(n^2)$ 時間である．作業領域は，各面でどの円の内部にありどの円の外部にあるかという位置関係を，n 個の円に対して保持しなければならないので $O(n)$ 領域が必要である． □

5. 平面走査法によるアルゴリズム

平面走査法とは，図 7 のように垂直な走査線 ℓ を仮定し，これを左から右に動かしながら最適解の候補を順次求めていき，全体の最適解を求める方法である [5]．このアルゴリズムの計算量は $O(n^3)$ 時間，$O(n)$ 領域である．平面走査法によるアルゴリズムでは，2 重連結辺リストを用いたアルゴリズムに比べ，作業領域が $O(n^2)$ から $O(n)$ に改善されている．2 重連結辺リストを用いたアルゴリズムでは，円のアレンジメント全体の情報を蓄え，その情報をもとに各面を順に訪れ，各面の局所最適解を出して全体の最適解を求めていた．円のアレンジメント全体の情報を蓄えるのには $O(n^2)$ 領域が必要である．それに対し，平面走査法によるアルゴリズムの基本的な考えは，円のアレンジメントで必要な部分だけを構成して，走査とともに必要な更新を行うというものである．このアイデアにより作業領域を $O(n)$ に抑えることができる．第 5.1 節では，平面走査法による円弧の交点を列挙する方法を説明する．第 5.2 節では，平面走査法による面を全探索する方法を説明する．各手順の計算量は表 2 に示す．

表 2 平面走査法によるアルゴリズムの各手順の計算量．

手順	時間計算量	作業領域
円弧の交点列挙	$O(n^2)$	$O(n)$
局所最適解の候補を求める	$O(n)$	$O(n)$

5.1 平面走査法による円弧の交点列挙

この節では，平面走査法による円弧の交点を列挙する方法および，必要なデータ構造の説明をする．

5.1.1 イベント点での処理

円を上下左右の極点により 4 つの区切られた円弧に分ける (図 7)．円弧の端点，またはその分けられた円弧の交点をイベント点と呼ぶ．走査線がイベント点に到達したときにアルゴリズムは処理を実行する．各イベント点では状態に変化が生じるが，どのような変化が生じるかはイベント点の種類によって定まる．各イベント点における走査線の状態は，走査線と交差する円弧の集合によって定まる．

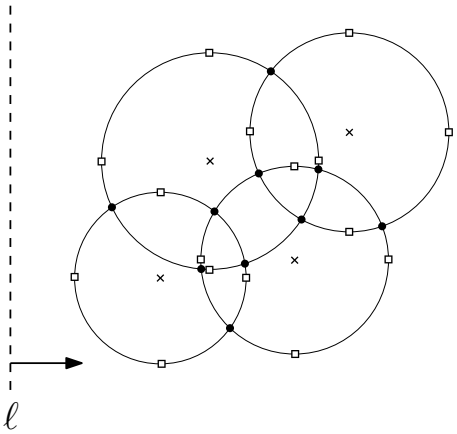

図 7 平面走査法．ℓ が走査線である．白四角の点が各円弧の端点，黒丸の点が各円弧の交点であり，イベント点を表す．

(a) 走査線が円弧の左端点に達すると，その円弧と新たに交差するようになる．

(b) 走査線が円弧の右端点に達すると，それ以降はその円弧とは交差しなくなる．

(c) 走査線が交点に達すると，交点を構成する 2 つの円弧の走査線上での上下関係が変わる．

イベント点での操作を以下に示す．(a) のとき，新たな円弧が走査線と交差し始めるので，状態に加えなければならない．また，この円弧と，すでに走査線と交差している各円弧との間で交差判定を行う．(b) のとき，その円弧はそれ以降走査線とは交差しなくなるから，状態から削除しなければならない．このようにして，走査線と交差するような円弧対だけに交差判定を限定することができる．(c) のとき，交点を構成する 2 つの円弧の走査線上での上下関係が変わるので，状態を変化させる必要がある．

5.1.2 平面走査法を効率的に実行するためのデータ構造

平面走査法を効率的に実行するには，イベント点を効率良く順に列挙し，各イベント点での状態変化と交点の列挙を効率良く行う必要がある．

x リスト

まず，端点を x 座標の順にソートすることにより，イベント点が起こる順序を決める．イベント点を順に並べたリストを x リストと呼ぶ．最初に，円弧の端点のソート列を x リストとして求めておく．ソート順に従ってイベント点を順に見ていく．走査線が円弧の左端点に達したとき，その円弧が他の円弧と交差しているかを調べる．交差している場合は，交点を x リストに追加する．ただし，検出された交点が走査線よりも左にあるなら，その交点はすでに処理されているので，x リストには加えない．効率よく交点を追加するために，x リストは優先順位つきキューを用いて実装する．

y リスト

交点列挙を効率良く行うためには，走査線と交差する円弧の集合を垂直方向にソートされた状態で保つことが重要である．そこで，走査線と交差する各円弧を，交点の y 座標値をキーとする平衡 2 分探索木で管理する．左端点における y リストへの円弧の挿入，右端点における y リストへの円弧の削除を行う．このようにして，走査線と交差するような円弧対だけに交差判定を限定することができる．また，円弧の交点である場合，y リストにおける円弧の順序の変更，および円弧の交差判定を行う．交点を列挙する場合は，対象とする円弧に対して，その円弧のすぐ上とすぐ下の円弧に対してだけ交差判定をすればよい．走査線と交差している円弧の数は高々 $2n$ 個なので，円弧の挿入と削除，円弧の順序の変更，そして，すぐ上と下の円弧の発見はいずれも $O(\log n)$ 時間で実行できる．

以上の操作により各円の交点をすべて列挙する．円弧の交点列挙のアルゴリズムを Algorithm2 に示す．

なお，交点列挙のアルゴリズムに関しては Bentley-Ottmann のアルゴリズム [6] を参考にしている．詳しくはそちらも参照されたい．また，次の定理が成り立つ．

定理 5.1. 平面走査法による円弧の交点列挙の計算量は，$O(n^2 \log n)$ 時間，$O(n)$ 領域である．

証明. 各円に対して円弧の端点の数は 4 つなので，端点の数は $4n$ 個となる．また，各円弧の交点の数は補題 3.2 より $O(n^2)$ 個となる．よって，イベント点の数は $O(n^2)$ 個である．

イベント点をすべて記憶した場合，$O(n^2)$ 領域が必要だが，平面走査法は，円のアレンジメントを部分的に構成していく方法である．必要な作業領域は，x リストについては，各円弧の端点と走査線と交わっている円弧の交点なので，$O(n)$ 領域である．y リストについては，走査線と交差している円弧の数は高々 $2n$ 個なので，$O(n)$ 領域である．x リストと y リストをともに平衡 2 分探索木で実現すれば，円弧の端点と交点の処理は，1 つあたり $O(\log n)$ 時間で実行できる．したがって，全体の処理時間は $O(n^2 \log n)$ である． □

Algorithm 2 平面走査法よる円弧の交点列挙

Input: 各既存点の座標 (x_i, y_i) と指定された距離 δ_i.

for 各円 **do**
  円を 4 つの円弧に分割する端点を x リストに挿入する.
end for
repeat
  次のイベントを E として取り出す.
  if E が円弧の左端点 **then**
    E に関連する 2 つの円弧 s と t を y リストに挿入する.
    if s と交差する円弧がある **then**
      s と交わる点 c を x リストに挿入する.
    end if
    if t と交差する円弧がある **then**
      t と交わる点 d を x リストに挿入する.
    end if
  else if E が円弧の右端点 **then**
    E に関連する 2 つの円弧 s と t を y リストから削除する.
  else if E が 2 つの円弧 s と t の交点 **then**
    y リストにおける s と t の順序を変更する.
    if s と交差する円弧がある **then**
      s と交わる点 c を x リストに挿入する.
    end if
    if t と交差する円弧がある **then**
      t と交わる点 d を x リストに挿入する.
    end if
  end if
until x リストが空になる

5.2 平面走査法による面の全探索

第 5.1 節では交点を全て列挙する方法を説明した．この節では，列挙した交点に接続する各面の局所最適解の候補を求める方法を説明する．手順としては，まず，交点の目的関数を評価する．交点は局所最適解の候補なので，交点の目的関数を評価する必要がある．次に，交点に接続する 4 つの面について目的関数と局所最適解の候補を求める．縮退はないと仮定しているので，各交点には必ず 4 つの面が接続する (図 8)．まず，各面の目的関数の式を立てる．しかし，平面走査法によるアルゴリズムでは，2 重連結辺リストを用いたアルゴリズムのように効率的に式を立てることはできない．$O(n)$ の時間が必要である．詳しくは第 5.2.1 節で説明する．また，各面に対しては交点に接続する辺の情報しかないため，面の局所最適解が求まるとは限らない．しかし，面を構成する頂点を全て訪れるため，面の境界をすべてたどることができる．よって，各面の局所最適解を求めることができる．詳しくは第 5.2.2 節で説明する．

円弧の交点を列挙するアルゴリズムでは，計算時間は $O(n^2 \log n)$ である．しかし，各交点で目的関数の式を立てるのに $O(n)$ 時間が必要なため，平面走査法によるアルゴリズム全体では $O(n^3)$ 時間かかる．

5.2.1 各面での目的関数の式の立て方

まず，面の目的関数を計算する．ここで注意したいのは，2 重連結辺リストを用いたアルゴリズムでは，オイラーツアーによる一筆書きの順で各面を訪れた．横切る円に対してのみ式の符号を変えるだけで効率的に次の面の式を立てることができた．平面走査法によるアルゴリズムでは，面を隣り合う順に探索するわけではない．平面走査法で各交点を訪れるたびに，各面でどの円の内部にありどの円の外部にあるかという位置関係を，毎回 n 個の円に対して計算しなければならない．よって各交点で $O(n)$ 時間必要である．

目的関数の式の立て方の具体例を図 8 に示す．頂点に対して，どの円の内部にあるか，どの円の外部にあるかの位

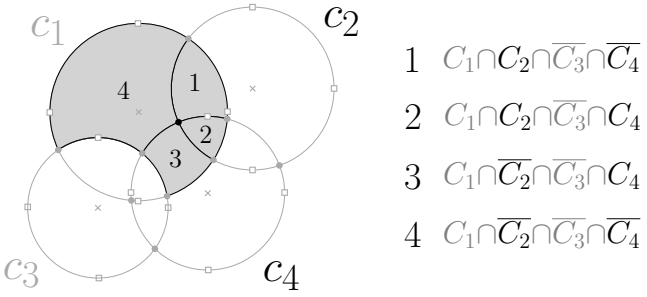

図 8 頂点に接続する 4 つの面に対する各円の位置関係．左側に頂点に接続する 4 つの面，右側に各面番号に対応する各円の位置関係を表している．

置関係を調べる．頂点を構成する 2 つの円 C_2 と C_4 はちょうど円周上にある．この 2 つの円以外の円の位置関係は，頂点に接続する 4 つの面に対して変わらない．まず，2 つの円以外の各円 C_1 と C_3 に対して，円の内部にあるか，外部にあるかの位置関係を調べる．そして，残りの 2 つの円の位置関係により，4 つの面の各円の位置関係を調べる．あとは，その情報をもとに目的関数の式を立てる．

5.2.2 各面での局所最適解の求め方

各面での局所最適解を求める方法は，4.2.2 節と同様である．2 重連結辺リストを用いたアルゴリズムでは，面の境界上をたどることができたので，面を 1 回訪れれば，その面の局所最適解を求めることができた．しかし，平面走査法によるアルゴリズムでは，1 回の面の探索で局所最適解を求めることができるとは限らない．なぜなら，1 回の面の探索でわかる情報は，面を構成する 1 つの頂点と頂点に接続する 2 つの辺のみだからである．しかし，各面の頂点はすべて列挙されるので，走査線が右端に到達する頃には，面の境界上をすべて探索できることになる (図 9)．よって，各面での局所最適解の候補はすべて求めることができ，全体の最適解を求めることができる．

6. おわりに

本稿では，まず既存点までの距離誤差を最小にする点位置発見問題を定義した．そして，この問題を解く 2 つのアルゴリズムを開発した．1 つは，データ構造に 2 重連結辺リストを用いた $O(n^2 \log n)$ 時間，$O(n^2)$ 領域で解くアルゴリズム，もう 1 つは，平面走査法による $O(n^3)$ 時間，$O(n)$ 領域で解くアルゴリズムである．

今後の課題として，既存点までの距離誤差を最小にする点位置発見問題を解くアルゴリズムのさらなる改善があげられる．2 重連結辺リストを用いたアルゴリズムでは $O(n^2)$ の作業領域が必要であり，平面走査法によるアルゴリズムにより作業領域を $O(n)$ に改善することができた．しかし，計算時間が $O(n^3)$ となり，効率が悪くなった．さらに効率のよいアルゴリズムに改善できるかの検討が必要である．

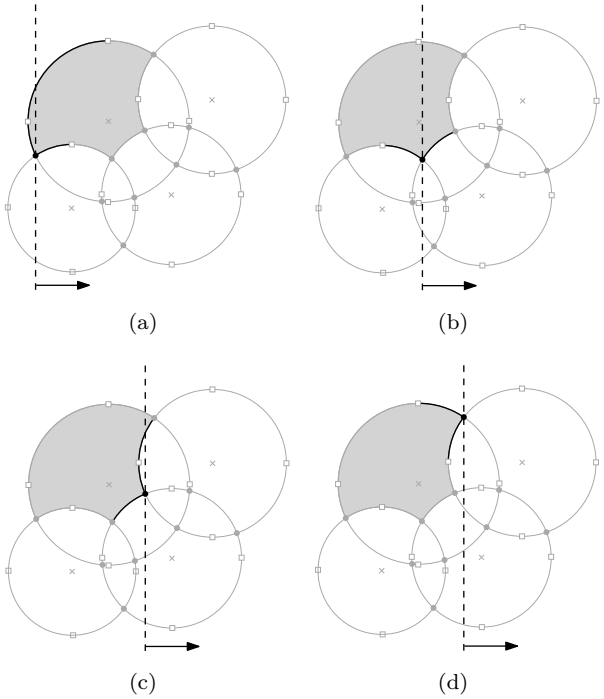

図 9 面の探索．(a)，(b)，(c)，(d) の順に，塗りつぶされた面の交点を訪れる．面の境界をすべて探索できていることがわかる．

参考文献

[1] T.F. Cox and M.A.A. Cox. *Multidimensional Scaling, Second Edition*. Chapman and Hall/CRC, second edition, 2000.
[2] N. Alon, H. Last, R. Pinchasi, and M. Sharir. On the complexity of arrangements of circles in the plane. *Discrete Computational Geometry*, 26:2001, 2001.
[3] H. Edelsbrunner, L.J. Guibas, J. Pach, R. Pollack, R. Seidel, and M. Sharir. Arrangements of curves in the plane - topology, combinatorics and algorithms. *Theoretical Computer Science*, 92(2):319–336, 1992.
[4] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, third edition, 2008.
[5] 浅野 哲夫. **計算幾何：理論の基礎から実装まで**. 共立出版, 2007.
[6] J.L. Bentley and T.A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, C-28:643–647, 1979.