修 士 論 文

計算幾何学問題に対する
メモリ制約付きアルゴリズム

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

小長谷松雄

2012 年 3 月

修 士 論 文

計算幾何学問題に対する
メモリ制約付きアルゴリズム

指導教官　浅野哲夫 教授

審査委員主査　浅野哲夫 教授
審査委員　上原隆平 教授
審査委員　面 和成 准教授

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

1010022 小長谷松雄

提出年月: 2012 年 2 月

概 要

本稿では，2 つの計算幾何学の問題に対するメモリ制約付きアルゴリズムを提案する．メモリ制約付きアルゴリズムとは，できるだけ少ない作業領域の使用を考慮したアルゴリズムである．入力データが格納されている領域を作業用として利用しないように，アルゴリズムの入力データは読み出し専用配列に格納されているとする．よって，この配列内の要素の削除や並び替えは禁止されている．

提案する最初のアルゴリズムは，最遠点ボロノイ図を描画する定数領域アルゴリズムである．これは，ボロノイ図を構成する頂点や辺の列挙を $\mathrm{O}(1)$ の作業領域のみで実現することである．列挙された頂点や辺は，別の計算幾何の問題を解くアルゴリズムに組み込むことができる．

もう一方は，線分交差列挙のメモリ制約付きアルゴリズムである．平面に与えられた直線分に対し，$\mathrm{O}(s)$ の作業領域だけで，交差する全ての線分対を検出する．ここで，s はアルゴリズムが処理に用いる記憶領域のパラメータを表す．本稿では，線分交差列挙問題に対して，指定されたサイズの記憶領域だけを用いて最も効率良く問題を解くことができるアルゴリズムを提案すると共に，記憶領域と計算時間の間のトレードオフについても解析する．

目 次

第1章　はじめに

1.1　研究の背景

1.1.1　研究の背景

メモリ制約付きアルゴリズムとは，作業領域を効率良く使用するアルゴリズムである．即ち，利用する作業領域をできるだけ少なく設定し，アルゴリズムを設計する．近年では，CPU は高速かつ高機能になり，メモリは安価になってきたおかげで，処理に多くのメモリを用いるアプリケーションが増えている．また，iPad のような高機能端末やデジタルカメラのような組み込みシステム機器も流行している．このような機器はコンパクトサイズであるにもかかわらず，多くの高機能なアプリケーションを実行できる．しかし，動作環境を考えると利用できるメモリは通常の計算機より小さいので，メモリ不足に陥り易い．よって，できるだけメモリを使わずに計算を行うアプリケーションを開発する必要があるが，アルゴリズムの設計の段階でこのような取り組みは積極的に行われていない．

作業領域が効率良く利用されているかは，アルゴリズム用いる作業領域の大きさで評価されるため，入力データが格納されている配列の扱いは重要である．in-place という種類のアルゴリズムは，作業領域として入力配列と定数個の変数だけを使用する．入力配列を作業用として扱うとは，アルゴリズムが入力配列の書き換え (並び替えや削除) を許すという意味である．例として，ヒープソートがある．多くの in-place アルゴリズムの結果が残されており，幾何学問題に対するアルゴリズムも存在する [8].

一方，メモリ制約付きアルゴリズムでは，入力データが格納されている配列は読み出し専用 (read-only) であると仮定する．read-only 配列では，配列内の要素の削除や並び替えなどが禁止されているので，作業領域として利用できない．従って，入力配列以外で用いる作業領域をできるだけ少なくしたものが，メモリ制約付きアルゴリズムの作業領域である．また，この仮定は実用的なものでもある．ある入力データに対し，異なるアルゴリズムを並列に用いる環境では，一方のアルゴリズムが，入力要素を削除したり並び替えたりできる状況は望ましくない．例えば，デジタルカメラの処理における入力は画像であるが，この種の入力に対して書き換えをするべきでない．

作業領域として定数個の変数のみを用いるアルゴリズムを定数領域のアルゴリズムと呼ぶ．定数領域アルゴリズムは，かなり厳しいメモリ制約を意味している．入力サイズが n であるとき，各々の変数は少なくとも $\mathrm{O}(\log n)$ ビットの領域が必要となる．何故なら，これより少ない記憶領域では n 個のものを区別するためのポインタでさえ保持できない

ため，全ての入力に対してアルゴリズムが正常に動作しないからである．$O(\log n)$ ビットの領域を用いたアルゴリズムは，対数記憶領域アルゴリズムと呼ばれ，計算複雑性理論の分野で昔から研究されているが，その研究は理論的なものが多い．本稿では，組み込みソフトとしての実用化も視野に入れたいため，これらの研究結果はそのまま利用できないという問題がある．

作業領域が $O(1)$ より少し多くの作業領域を利用できるとき，例えば $O(\sqrt{n})$ 作業領域が利用可能であるとき，どのくらい速いアルゴリズムを設計できるかという問題は非常に興味深い．一般に，大きな作業領域を用いれば，時間計算量は減少する．逆に，メモリの使用量を抑えると，計算時間は増加する．記憶領域のサイズと計算時間の間にこのような相関があるものを記憶領域と計算時間のトレードオフと呼ぶ．トレードオフがあるアルゴリズムは，任意の大きさの作業領域で実行できる．よって，時間を優先するかメモリ消費を優先するかで，マシンを効率的に利用できる．

メモリ制約付きアルゴリズムの欠点は，多くの作業領域を用いることができないため，複雑なデータ構造が使用できないことである．しかし，これは逆に利点でもある．複雑なデータ構造を用いることがない代わりに，アルゴリズムの構造は単純化するため，アルゴリズムの解析やコーディングが容易になる．例えば，最遠点ボロノイ図の描画を行う既存のアルゴリズムは，$O(n)$ の作業領域を用いて，二重連結辺リストと呼ばれるデータ構造を用いるため，その実装は簡単ではない．しかし，これは定数領域で実現でき，更にそのコードは非常に簡単である [3].

1.1.2 計算のモデル

本稿を通して用いる計算機は，一般的な RAM(random access machine) を想定する．また，入力データは読み出し専用配列 (read-only array) に格納されていると仮定する．この配列では，配列内の内容を変更するような一切の操作が禁止されているが，任意の要素へのランダムアクセスは可能である．各要素へのアクセスやその他の基本的な計算は $O(1)$ 時間で行われるとする．

作業領域の大きさは $o(n)$ で，単位領域当たり $O(\log n)$ ビットの長さのポインタやカウンタが格納できるとする．特に，$O(1)$ の作業領域とは，$O(\log n)$ ビットの領域を意味する．再帰呼び出しで必要となるスタックなども作業領域とする．よって，全体の作業領域を評価するとき，各呼び出しで用いられる作業領域も考慮しなければならない．

1.1.3 関連研究

メモリ制約付きアルゴリズムという枠組みの研究は 30 年前から始められており，既に多くの研究結果が得られている．中でも，下記に述べる研究結果はエイポックメイキングなものとして注目を集めたものである．

中央値発見問題:

n 個の数の中央値を発見する．入力データは read-only array に格納されていると仮定する．Munro と Raman [12] は，$O(1/\epsilon)$ の作業領域で $O(n^{1+\epsilon})$ 時間で計算するアルゴリズムを提案した．ここで，$\epsilon > 0$ は任意の小さな定数である．線形の作業領域を使えば線形時間で中央値の計算ができることは既に知られていて [7]，多くのアルゴリズムの基礎を成している．ここで紹介した結果は，定数作業領域でもほぼ線形時間で中央値が求まることを示したものであり，画期的である．

グラフの連結性判定問題:

無向グラフが read-only array に与えられたとする．このとき，任意の 2 頂点間が連結かどうかの判定を行う．この問題は長い間未解決問題であったが，Reingold [13] が 2005 年に定数領域を用いた多項式時間アルゴリズムを提案した．しかし，そのアルゴリズムは大掛かりな理論に基づいたものであり，実用的なアルゴリズムとはいえない．

凸包構成問題:

n 個の点が read-only array に与えられているとき，与えられた点の凸包を計算する．Chan と Chen [11] は，$O(s)(s \leq n)$ の作業領域を用いて $O(n(n/s + \log n))$ 時間で凸包を計算するアルゴリズムを提案した．提案されたアルゴリズムは，記憶領域と計算時間の最適なトレードオフがある．この論文は，計算時間と作業領域の大きさの間のトレードオフに関する議論を計算幾何学の問題に最初に適用したものである．

1.2 研究の目的

本論文では，入力が読み出し専用配列に格納されていると仮定し，2 つの計算幾何学問題に対するメモリ制約付きアルゴリズムを提案する．また，組み込みソフトとしての利用を想定し，実装可能なアルゴリズムの開発を目指す．

一つ目は，平面に与えられた n 点に対する最遠点ボロノイ図を描画する定数領域のアルゴリズムを設計する．ボロノイ図を描画するとは，ボロノイ図を構成する頂点や辺の全列挙を意味する．列挙された頂点や辺を用いて，別の計算幾何学問題への応用が可能である．提案したアルゴリズムは，$O(1)$ の作業領域で，$O(n^2)$ 時間で最遠点ボロノイ図を描画する．

もう一つは，平面上の n 線分に対し，交差する線分対を全列挙するメモリ制約付きアルゴリズムを提案する．この問題に関しては，記憶領域と計算時間の間にトレードオフがあるアルゴリズムを考える．即ち，s 本の線分を保持できる作業領域 $(s = o(n))$ を用いて，効率的に線分交差を列挙するアルゴリズムを考える．

1.3　本論文の流れ

2 章では最遠点ボロノイ図を描画する定数作業領域アルゴリズムを提案する．また，このアルゴリズムを用いて最小包含円問題を解く定数領域アルゴリズムを考える．

3 章では，記憶領域と計算時間にトレードオフをもつ線分交差列挙アルゴリズムを設計する．本論文で提案する線分交差列挙のアルゴリズムは，非常に複雑であるため，実装が難しい．与えられる線分が水平線分と垂直線分のみであれば，比較的簡単な手法で理想的なトレードオフをもつアルゴリズムを設計できるので，それについても述べる．

4 章で以上のまとめと今後の課題について述べる．

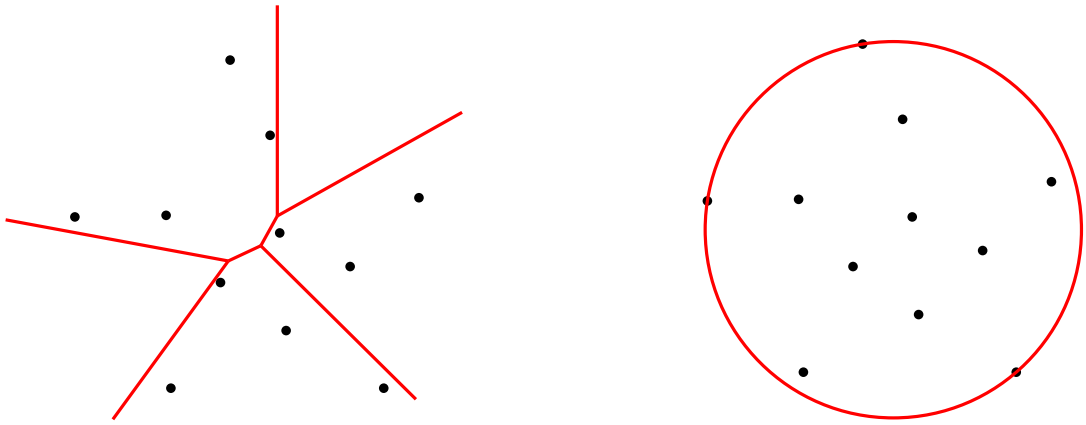
第2章　最遠点ボロノイ図描画

平面上に n 個の点集合 S が与えられているとする．本章では，S に対する最遠点ボロノイ図を描画する定数領域アルゴリズムを提案する [3]．入力点データは読み出し専用配列に格納されていると仮定する．

2.1　最遠点ボロノイ図

S に対する最遠点ボロノイ図 $FV(S)$ とは，平面を凸領域 $FVR(p_1), \ldots, FVR(p_k)$ に分割したものである．このとき，領域 $FVR(p_i)$ の任意の点は S の点の中で $p_i \in S$ が一番遠い点となる．

最遠点ボロノイ図の描画とは，ボロノイ図の境界線を構成する頂点や辺を全て列挙するという意味を持つ．列挙された辺や頂点は，他の計算幾何学問題を解くアプリケーションとして利用される．その問題の一つとして，本稿では最小包含円問題を取り上げる．



(a) 最遠点ボロノイ図　　(b) 最小包含円

図 2.1: 点集合 S に対する最遠点ボロノイ図と最小包含円

最小包含円問題とは，平面に与えられた点集合 S に対して，S の点を全て包含する半径最小の円を求める問題である．最遠点ボロノイ図を用いた既存の手法では，ボロノイ図を計算し，$O(n)$ の作業領域に二重連結辺リストとして保持する．リストの構築に $O(n \log n)$

時間を要する．しかし，リストを用いると一つのボロノイ頂点，辺の列挙を $\mathrm{O}(1)$ 時間で実現でき，ボロノイ領域の境界を辿ることができる．全てのボロノイ頂点と辺を $\mathrm{O}(n)$ 時間で列挙できれば，同じ時間で S を包含する最小の円を発見できる．

本稿で提案するアルゴリズムは，定数領域でも二重連結辺リストを保持しているかのように振る舞う．即ち，ボロノイ頂点やボロノイ辺の全列挙や，あるボロノイ領域の境界を辿る操作を $\mathrm{O}(1)$ の領域で実現する．これらの操作に要する時間は $\mathrm{O}(n^2)$ である．また，最小包含円問題も同じ時間で実現できることも示す．

2.2 準備

n 点の集合に対する最遠点ボロノイ図 $FV(S)$ を描画する定数領域を用いたアルゴリズムを考える．ここでは，アルゴリズムで用いる関数を準備する．

初めに，いくつかの用語や記号を定義をする．簡単のために，与えられた点は一般の位置にあるとする．即ち，どの 4 点も同一円周上に存在しないので，ボロノイ図 $FV(S)$ の頂点はちょうど 3 本のボロノイ辺に接続する．$p_i \in S$ に対するボロノイ領域 $FVR(p_i)$ の任意の点は，S の点の中で p_i が一番遠い点となる．各ボロノイ領域は，無限凸領域となることが知られており，その境界は，0 本以上の線分をなす有向辺と 2 本の半直線で囲まれている．本稿では，ボロノイ領域 $FVR(p_i)$ の境界線を構成するボロノイ辺は方向付けられているとする．ボロノイ辺を辿るとき，p_i に対するボロノイ領域 $FVR(p_i)$ が常に左側に位置するように方向付ける．各ボロノイ辺は 2 つのボロノイ領域を分割している．$E(p_i, p_j)$ を 2 つのボロノイ領域 $FVR(p_i)$ と $FVR(p_j)$ を分割するボロノイ辺を表すとする．また，$E(p_i, p_j)$ の始点から終点へ辿るとき，左側に領域 $FVR(p_i)$ が位置し，右側に $FVR(p_j)$ がある．この辺の反対方向のボロノイ辺は $E(p_j, p_i)$ と表す．仮定から，ちょうど 3 本のボロノイ辺は 1 点で交わる．この点を，ボロノイ頂点とよぶ．従って，各ボロノイ頂点 $V(p_i, p_j, p_k)$ は，3 点 $p_i, p_j, p_k \in S$ で定義できると仮定する．ボロノイ領域を持つ S の点は，S の凸包上の頂点だけであることが知られている [6]．S の凸包とは，S を包含する最小の凸多角形のことである．最遠点ボロノイ図の例を図 2.2 に表す．

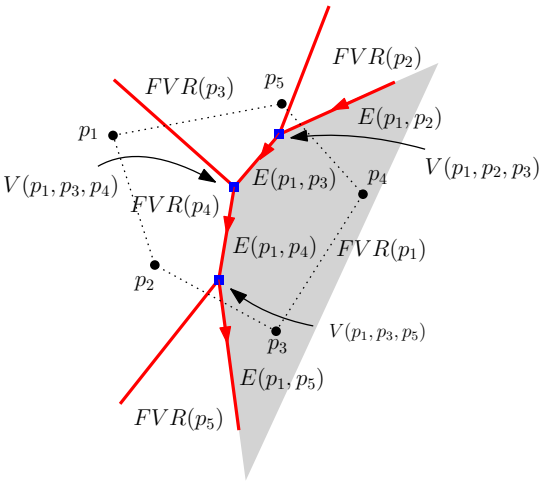


図 2.2: 最遠点ボロノイ図．$\{p_1,\ldots,p_2\}$ は凸包上の点，$FVR(p_i)$ は点 p_i に対するボロノイ領域，$E(p_i,p_j)$ は 2 点 p_i,p_j に対するボロノイ辺を表す．

点線の多角形は S の凸包を示している．図例において，最も左にある入力点を p_1 として，S の凸包上の点を反時計回り順で並べたものを $p_2,\ldots,p_5$ とする．p_1 のボロノイ領域 $FVR(p_1)$ は図の灰色の領域である．よって，最遠点ボロノイ図は，ボロノイ頂点，方向をもつ直線分ボロノイ辺や半直線ボロノイ辺，無限ボロノイ領域によって構成されている．最遠点ボロノイ図を表現する方法として，二重連結辺リストが用いられる．次のような項目が保持されているリストである．

頂点レコード (Vertex record):
頂点 v の座標が格納されている．また，v を始点とする有向辺も保持する．

面レコード (Face record):
面 f の最初のボロノイ辺へのポインタを保持する．ボロノイ辺を次々に辿ることにより，面を表現する．最遠点ボロノイ辺の最初の辺は，無限遠からやってくる半直線である．

辺レコード (Edge record):
同じ境界線で，次のボロノイ辺をさすポインタを保持する．自身の辺を境界とする面へのポインタも保持する．

上記のリストに対して，各種の問い合わせ行う．具体的には下に列挙した関数を仮定する．実際，これらの関数の実装は容易である．

関数 FirstExtremePoint(S)
S の点の中で最も左に位置する点を返す．正確には，最小の x 座標を持つ S の点のインデックスを返す．

関数 CouterClockwiseNexExtremePoint(p_i)
p_i から反時計回り順で次の凸包上の頂点のインデックスを返す.

関数 FrontEndOfVoronoiEdge($E(p_i, p_j)$)
ボロノイ辺 $E(p_i, p_j)$ の終点を返す．この点はボロノイ頂点である．実際には，$V(p_i, p_j, p_k)$ とすると，この点を定義する点 p_k のインデックス k を返す.

関数 BackEndpointOfVoronoiEdge($E(p_i, p_j)$)
ボロノイ辺 $E(p_i, p_j)$ の始点を返す．実際には，ボロノイ頂点 $V(p_j, p_i, p_k)$ であるとき，この点を定義する点 p_k のインデックス k を返す.

関数 NextVoronoiEdge($E(p_i, p_j), V(p_i, p_j, p_k)$)
ボロノイ領域 $FVR(p_i)$ 上の境界を辿るとき，ボロノイ辺 $E(p_i, p_j)$ の次のボロノイ辺 $E(p_i, p_k)$ を返す．始点となる $V(p_i, p_j, p_k)$ の情報も渡すことにより，次の辺は簡単に求められる．正確には，ボロノイ辺を定義する p_i, p_k のインデックス i と k を返す.

2.3 関数の動作

アルゴリズムの設計の前に予め S の重心 c(centroid) を計算しておく．S の重心とは S の全ての点の x 座標の平均値と y 座標の平均値によって定まる座標である．重心は，必ず与えられた点の凸包内部に位置する.

前節で定義した全ての操作は O(1) の領域で以下のように実現できる.

FirstExtremePoint(S):
S の最左点を見つける．最左点を通る垂直線より左側には S の点が存在しないため，この点は必ず S の凸包上の点である．この操作は，配列をスキャンし，最小の x 座標をもつ点のインデックスを求める．O(1) の作業領域で実現できる.

CouterClockwiseNextExtremePoint(p_i):
p_i から反時計回りに次の凸包上の点 p_j を見つける．$p_i \in S$ を S の凸包上の点とし，p_i から，重心 c と反対の方向に延びる半直線を考える (図 2.3 参照)．この半直線を p_i まわりに反時計回り順に回転させて初めにぶつかる S の点が p_j である．実際には半直線などは生成せずに，p_j の以下の性質を利用して見つける.

(1)p_j は有向辺 $\overrightarrow{cp_i}$ の左側にあるので，(c, p_i, p_j) は反時計回りである．

(2)p_j は先に定義した半直線を反時計回りに回したとき最初にぶつかる点であるので，$\overrightarrow{p_k p_i}$ の左側に位置する．即ち，任意の $p_k \in S \setminus \{p_i, p_j\}$ に対して，3 点 p_k, p_i, p_j の順序は反時計回りとなる．

従って，p_j の発見は，$\mathrm{O}(1)$ の作業領域で $\mathrm{O}(n)$ 時間で実現できる．

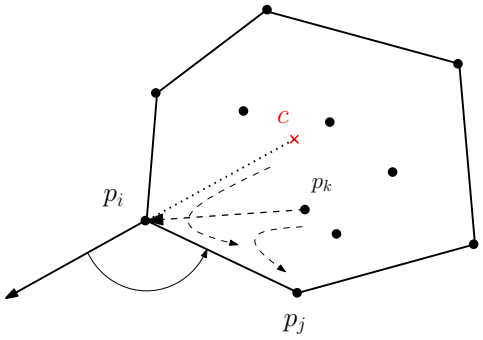


図 2.3: 重心 c と p_i を始点とする半直線を利用し，p_i の反時計回りに次の凸包上の頂点を見つける．

FrontEndpointOfVoronoiEdge($E(p_i, p_j)$):

$E(p_i, p_j)$ の終点である $V(p_i, p_j, p_k)$ を返す．各ボロノイ辺 $E(p_i, p_j)$ の端点は，S の包含円と対応する．凸包上の点を反時計回り順で辿るので，点 p_k は $\overrightarrow{p_i p_j}$ の左側に位置する S の点である．$\overrightarrow{p_i p_j}$ の左側に位置する各 S の点 p' に対して，3 点 p_i, p_j, p' を通る円を計算し，その中で S の点を包含する円を定義する点を探せばよい．

3 点 p_i, p_j, p' を通る円の中心点は，線分 $\overline{p_i, p_j}$ の垂直二等分線上に現れる．$\overrightarrow{p_i p_j}$ の左側に位置する中心点の中で，p_i, p_j から最も離れた点を中心点として持つ円は S の包含円である．但し，全ての中心点が $\overrightarrow{p_i p_j}$ の右側に現れる場合，p_i, p_j に最も近い点が S の包含円の中心点である．正確には，円を定義する p_k のインデックスを返す．従って，$\mathrm{O}(1)$ の作業領域で，$\mathrm{O}(n)$ 時間で計算できる．

図 2.4 は，$E(p_i, p_j)$ の終点を見つける例を示している．$E(p_1, p_3)$ が与えられたとき，$\overrightarrow{p_1 p_3}$ の左側にある S の凸包上の点は p_4 と p_5 だけである．このどちらかの点と p_1, p_3 で定義される円を計算したとき，p_1, p_3, p_4 を通る円の方が大きい円であるので，この円の中心点 $V(p_1, p_3, p_4)$ が $E(p_1, p_3)$ の終点である．

ある点のボロノイ領域に対しその境界であるボロノイ辺を反時計回り順に辿るとき，最後のボロノイ辺には終点が存在しない．よって，そのようなボロノイ辺が与えられたとき，この関数は未定義な値を返す．

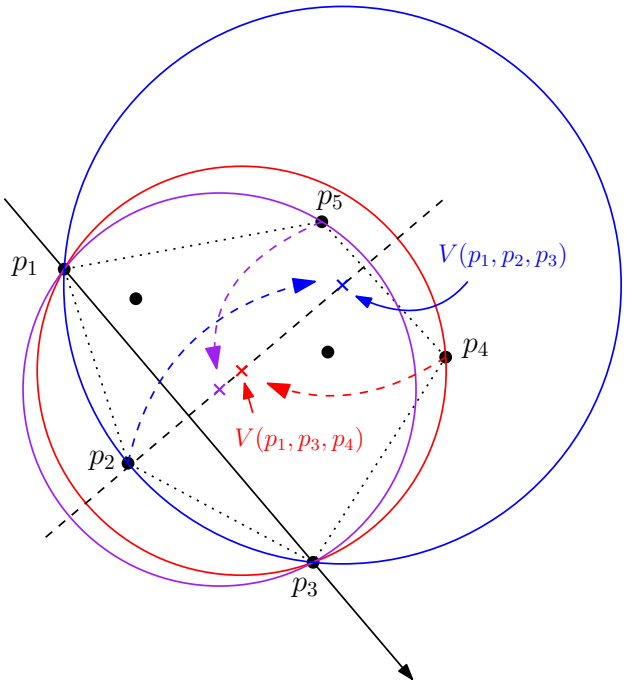


図 2.4: $E(p_1, p_3)$ の終点を計算する．有向直線 $\overline{p_1p_3}$ の左側にある S の凸包上の点は p_4 と p_5 がある．(p_1, p_3, p_4) で定義される円の方が大きいため，$E(p_1, p_3)$ の終点は $V(p_1, p_3, p_4)$ である．逆に，有向直線 $\overline{p_3p_1}$ の左側には p_2 だけがある．よって，$E(p_3, p_1)$ の終点は，$V(p_3, p_1, p_2) = V(p_1, p_3, p_2)$ である．

2.4 $FV(S)$を描画する定数領域アルゴリズム

2.4.1 アルゴリズム

与えられたn点に対応する最遠点ボロノイ図を描画するアルゴリズムを示す．即ち，O(1)の作業領域を用いて，全てのボロノイ頂点と辺をO(n^2)時間で列挙する．アルゴリズムと定理を以下に示す．

Algorithm 1 最遠点ボロノイ図の描画

入力: n個の点集合$S = \{p_1, \ldots, p_n\}$

出力: Sに対する最遠点ボロノイ図

p_i = FirstExtremePoint(S).
$i_0 = i$.
repeat {
  p_j = CounterClockwiseNextExtremePoint(p_i).
  p_k = FrontEndpointOfVoronoiEdge(p_i, p_j).
  一つ目の半直線ボロノイ辺$E(p_i, p_j)$を計算し出力.
  loop {
    $p_j = p_k$.
    p_k = FrontEndpointOfVoronoiEdge(p_i, p_j).
    if (p_kが存在しない) **then**
      exit the loop.
    end if
  }
} **until** ($i = i_0$)
二つ目の半直線ボロノイ辺$E(p_i, p_j)$を計算し出力.
p_i = CounterClockwiseNextExtremePoint(p_i).

定理 2.4.1 平面上に与えられた任意のn点に対して，O(1)の作業領域だけを用いてO(n)時間で最遠点ボロノイ図を描画するアルゴリズムが存在する．

2.4.2 最小包含円問題への応用

最遠点ボロノイ図の頂点や辺を列挙することで，最小包含円問題を解くことができる．Sの最小包含円となりうる候補は以下のうちのどちらかである．

(1) ボロノイ頂点を中心点とする円

(2) S の凸包上の 2 点の距離を直径とする円

(1) は S に対する最遠点ボロノイ図のボロノイ頂点として現れる．ボロノイ頂点 $V(p_i, p_j, p_k)$ は，3 点 p_i, p_j, p_k から最も遠く，かつ等距離となる点である．一方，$V(p_i, p_j, p_k)$ は，3 点を通り S を包含する円の中心である．よって，(1) は S の最小包含円の候補となる．

(2) はボロノイ辺を決める凸包上の 2 点として現れる (図 2.5 参照)．線分をなすボロノイ辺 $E(p_i, p_j)$ の両端点は，2 点 p_i, p_j を通る円の中心である．この 2 つの円 C_1, C_2 を定義する 3 点を頂点とする三角形が，両方とも鈍角三角形ならば，p_i, p_j を直径とする円は C_1, C_2 よりも小さい円である．よって，(2) は S の最小包含円の候補となる．

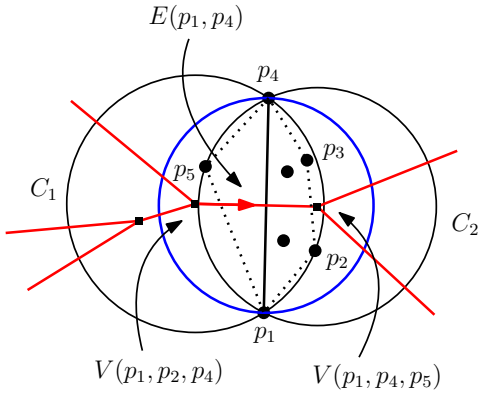


図 2.5: 青い円は線分 $\overline{p_1p_4}$ を直径とする円である．$E(p_1, p_4)$ の始点と終点は，それぞれ $V(p_1, p_2, p_4), V(p_1, p_4, p_5)$ である．よって，$(p_1, p_2, p_4), (p_1, p_4, p_5)$ で定義される円をそれぞれ C_1, C_2 とすると，どちらの円も与えられた点を包含する円である．しかし，C_1, C_2 よりも線分 $\overline{p_1p_4}$ を直径とする円の方が小さい．

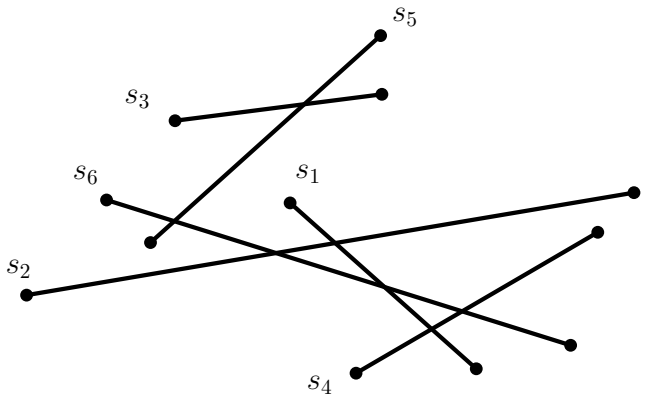
以上より，全てのボロノイ頂点とボロノイ辺を列挙できれば，S の最小包含円を計算できる．n 点に対する最遠点ボロノイ図のボロノイ辺とボロノイ頂点の総数は $O(n)$ であるから，アルゴリズム 1 を用いて，$O(n^2)$ 時間で計算できる．また，作業領域は $O(1)$ である．

従って以下の系を得る．

系 2.4.1 最遠点ボロノイ図 $FV(S)$ を描画する定数領域アルゴリズムを用いると，S の最小包含円を $O(1)$ の作業領域と $O(n^2)$ 時間で求めることができる．

第3章　線分交差列挙問題

平面に n 本の直線分の集合が与えられているとする (図 3.1 参照)．本章では，与えられた線分集合の交差する線分対を全て列挙するメモリ制約付きアルゴリズムを設計する．s を $\mathrm{O}(1)$ から $\mathrm{O}(n)$ までの間のパラメータとして，$\mathrm{O}(s)$ のサイズの作業領域だけを用いるアルゴリズムの計算時間を求め，作業領域のサイズと計算時間の間のトレードオフを考える．最も良いトレードオフを達成するアルゴリズムを設計することが本章の主題である．ただし，n 本の線分は読み出し専用配列に格納されていると仮定する．



- Input: $S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$
- Output: $\{\{s_1, s_2\}, \{s_1, s_4\}, \{s_1, s_6\}, \{s_2, s_6\}, \{s_3, s_5\}, \{s_4, s_6\}, \{s_5, s_6\}\}$

図 3.1: 線分交差列挙問題．この例では 6 本の線分に対して 7 個の交点が線分対の形で報告される

3.1　背景

線分交差列挙問題は，計算幾何学の分野における基本的な問題の一つで，多くの研究結果がある．Bentley と Ottman [5] は，$\mathrm{O}(n + K)$ の作業領域を用いて，$\mathrm{O}((n + K) \log n)$ 時間で，与えられた線分集合に対する全ての線分交差を列挙できるアルゴリズムを提案した．ここで，K は交差対の数を表す．彼らのアルゴリズムは平面走査を用いたもので，線分交差列挙の最も基本的なアルゴリズムとして知られている．平面走査による線分交差列挙の計算時間は，入力サイズだけでなく交点の数にも依存する．後に，同じ計算時間で，

用いる作業領域を $O(n)$ に改善した．Balaban [4] は，$O(n)$ の領域で，より効率的に線分交差を列挙できるアルゴリズムを提案した．このアルゴリズムは平面走査に基づいたものであるが，その計算時間は $O(n \log n + K)$ である．

また，使用する作業領域をより少なくするという観点からも研究されている．平面走査のアルゴリズムでは，入力配列の他に木構造を保持するために，$\Omega(n)$ の記憶領域が必要となる．Chan と Chen [10] は，このような入力配列以外に用いる作業領域を $O(\log^2 n)$ に改善し，これを用いた平面走査アルゴリズムの計算時間として，$O((n+K)\log^2 n)$ を得た．また，$O(1)$ の領域だけでも，力任せに行えば全ての線文交差を列挙できる．このとき，n 本の各線分に対し $n-1$ 本の線分と交差する可能性があるので，全て調べるのに $O(n^2)$ 時間を要する．

3.2 提案するアルゴリズム

n 本の線分が格納されている読み出し専用配列を S とする．よって，この配列を作業領域として利用できない．$O(s)$ の領域しか利用できないとすると，$s < n$ のとき S の全ての線分を保持できないので，S を部分集合 $S_1, \ldots, S_{\lceil n/s \rceil}$ に分割し，必要なものだけを $O(s)$ の領域に保持して処理を行う (図 3.2 参照)．S_i に含まれる線分数は高々 s 本である．

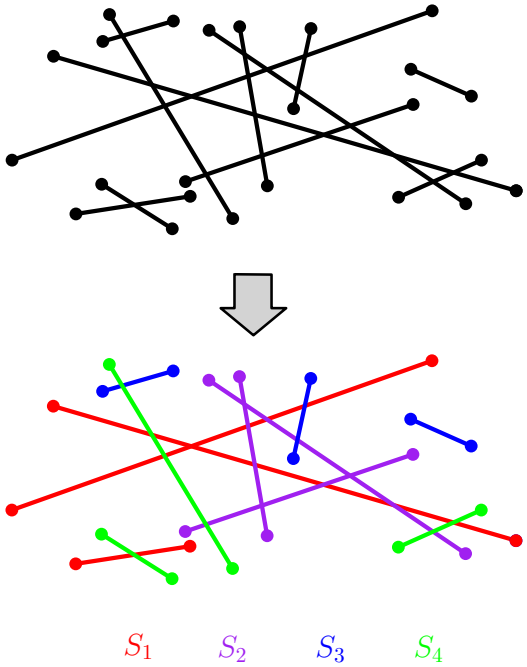


図 3.2: S をそれぞれちょうど 3 本の線分を含む部分集合 S_1, S_2, S_3, S_4 に分割．$|S_i| = 3$, $i = 1, 2, 3, 4$.

アルゴリズムの構造は簡単で，二段階の手順で処理する．S の線分で交差する線分対の集合を $Int(S)$ と表す．アルゴリズムの第一段階として，分割した各々の S_i に対し，$O(s)$

の作業領域を用いて $Int(S_i)$ を求める．第二段階では，2 つの部分集合の和集合 $S_i \cup S_j$ に対して，$O(s)$ の領域を用いて，交差線分対の集合 $Int(S_i, S_j) = \{\{s, s'\} \mid s \in S_i, s' \in S_k$，かつ s と s'が交差する $\}$ を求める．以下にアルゴリズムを示す．

Algorithm 2 S の全ての交差する線分対を列挙

入力: 平面に与えられた n 線分の集合 S

出力: 全ての交差線分対

仮定: S は読み出し専用配列

```
S → S_1 ∪ S_2 ∪ ··· ∪ S_⌈n/s⌉
for each S_i, i=1,...,⌈n/s⌉ do {
  (第一段階)
  Int(S_i) を計算して出力
  (第二段階)
  for each S_j, j=1,...,⌈n/s⌉ do {
    Int(S_i, S_j) 計算して出力        // 交差 2 線分のグループが異なるなら交差を報告
  }
}
```

第一段階の操作では，Balaban のアルゴリズム [4] を用いることで，$Int(S_i)$ を $O(s)$ の作業領域で，$O(s \log s + K_{S(i)})$ 時間で計算できる．$K_{S(i)} = |Int(S_i)|$ とする．

第二段階の操作も Balaban のアルゴリズムを用いて交差対を求める．$|S_i \cup S_j| \leq 2s$ であるから，作業領域は $O(s)$ で十分である．Balaban のアルゴリズムは，与えられた $S_i \cup S_j$ の線分交差を全て計算するので，$Int(S_i, S_j)$ だけでなく $Int(S_i)$ と $Int(S_j)$ も同時に検出する．このとき，$Int(S_i)$ と $Int(S_j)$ も出力してしまうと，全体を通して，何度も同じ交差対を出力する問題が起こる．ある $i \in \{1, \ldots, \lceil n/s \rceil\}$ に対して，S_i の線分たちと交差する可能性があるグループの数は，$O(\lceil n/s \rceil)$ である．よって，$Int(S_i)$ が出力される回数は，入力サイズに比例する．一つの交差対の出力はちょうど一回に抑えたい．そこで，同グループの交差する線分対は報告しないように，交差する 2 本の線分が互いに異なるグループに属しているときだけその対を出力するようにする．

以上により S の全ての線分交差対をちょうど一回で列挙できるが，このアルゴリズムはある問題を負っている．全ての線分交差対を何度も出力することはないが，交差対の検出は何度も行われる．第二段階における線分交差の出力では，$S_i \cup S_j$ のある 2 本の線分の交差を検出した後に，出力するか否かの判定が行われる．即ち，$Int(S_i)$ と $Int(S_j)$ は出力されないだけで検出はされるので，アルゴリズム全体で，何度も同じ交差対を検出することになる (図 3.3 参照).

アルゴリズムの計算時間は交差する線分対の数にも依存するので，同じ交差を何度も検出することは明らかに計算時間の浪費である．従って，$O(s)$ の作業領域を用いたときの

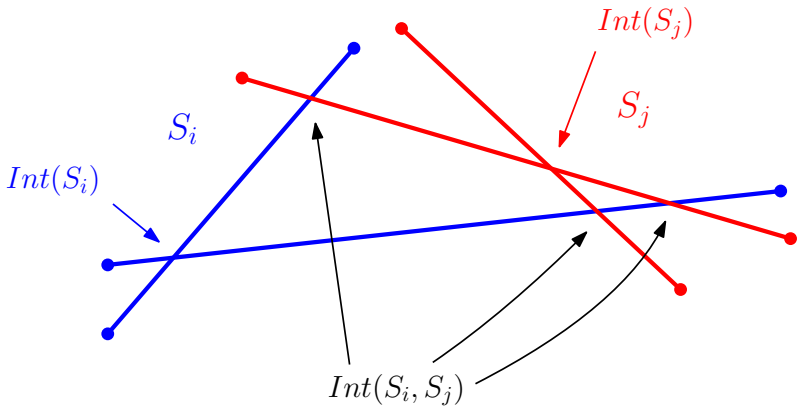


図 3.3: 第二段階で検出される交差

計算時間は以下のように得られる．

$$\sum_{i=1}^{\lceil \frac{n}{s} \rceil} \sum_{j=1}^{\lceil \frac{n}{s} \rceil} \mathrm{O}(s \log s + K_{S(i),S(j)} + K_{S(i)} + K_{S(j)})$$
$$= \mathrm{O}\left(\frac{n^2}{s} \log s + \frac{n}{s} K_{S(1,\ldots,\lceil \frac{n}{s} \rceil)} + K_{S(1,\ldots,\lceil \frac{n}{s} \rceil),S(1,\ldots,\lceil \frac{n}{s} \rceil)} \right).$$

ここで，$K_{S(i)} = |Int(S_i)|$，$K_{S(i),S(j)} = |Int(S_i, S_j)|$，$K_{S(1,\ldots,\lceil n/s \rceil)} = \sum_{i=1}^{\lceil n/s \rceil} K_{S(i)}$，$K_{S(1,\ldots,\lceil n/s \rceil),S(1,\ldots,\lceil n/s \rceil)} = \sum_{i=1}^{\lceil n/s \rceil} \sum_{j=1}^{\lceil n/s \rceil} K_{S(i),S(j)}$ とする．以上より，提案するアルゴリズムはメモリと時間のトレードオフがある．しかし，$K_{S(i)} = \mathrm{O}(s^2)$ と考えると，s をどのように設定しても，$\mathrm{O}(n^2)$ 時間を要するアルゴリズムとなる．よって，提案アルゴリズムは，力任せで線分交差を列挙する計算時間 $\mathrm{O}(n^2)$ より悪くなってしまう．

3.3 第二段階の修正

3.3.1 CSW のデータ構造

$S_i \cup S_j$ の線分交差を報告するとき，$Int(S_i, S_j)$ だけを求めたい．平面走査法のように，入力された線分集合に対し全ての交差線分対を検出する方法では，アルゴリズム全体で，同じ線分対を何度も検出するという問題が起こる．よって，第二段階で $Int(S_i)$ や $Int(S_j)$ を検出しないようにしなければならない．

本節では，Agarwal と Sharir [1] が提案したアルゴリズムとデータ構造を利用する．ここで，線分集合 S と問い合わせ線分 s が与えられたとする．このデータ構造は CSW の技法を用いたデータ構造で，$Int(\{s\}, S)$ を効率よく列挙するために，S に対して構築される．

CSW のデータ構造とは，”見かけ上最適”な単体領域探索を可能とするデータ構造で，Chazell，Sharir，Welzl [9] が提案した．”見かけ上最適”とは，時間や領域を $\mathrm{O}(n^\epsilon)$ だけ犠

牲にして，全体の効率を良くするという意味をもつ．ϵ は正の定数で，任意の十分小さい数である．単体領域探索とは，平面 (二次元空間) の場合，点集合 S と問い合わせ三角形 τ が与えられたとき，$S \cap \tau$ の点の数を数えたり列挙を行うものである．データ構造の構築はメインアルゴリズムのサブルーチンとして利用でき，これを用いることにより，記憶領域と時間のトレードオフがあるアルゴリズムも設計できる．

3.3.2 第二段階の修正

同じグループの線分交差を検出しないように，第二段階を次のように修正する．$Int(S_i)$ を報告した後，S_i に対するデータ構造を構築する．そして，ある線分 $s \in S \setminus S_i$ に対して，s と交差する S_i の線分を出力するようにこのデータ構造に問い合わせる．この操作を，全ての線分 $s' \in S \setminus S_i$ に対して行う．即ち，第二段階では $Int(S_i, S \setminus S_i)$ を列挙する．

Algorithm 3 S の全ての交差する線分対を列挙 (第二段階の修正)

入力: 平面に与えられた n 線分の集合 S

出力: 全ての交差する線分対

仮定: S は読み出し専用配列

$S \to S_1 \cup S_2 \cup \cdots \cup S_{\lceil n/s \rceil}$
for each S_i, i=1, $\ldots$, $\lceil n/s \rceil$ **do** {
  (第一段階)
  $Int(S_i)$ を計算して出力
  (第二段階)
  S_i に対する CSW のデータ構造 $\mathcal{D}_i$ を構築
  for each $s' \in S \setminus S_i$ **do** {
    $\mathcal{D}_i$ を用いて $Int(\{s'\}, S_i)$ を計算して出力
  }
}

S_i に対するデータ構造 $\mathcal{D}_i$ を保持するために，$O(s^{1+\epsilon})$ の作業領域を利用してよいとする．第一段階では，S_i の線分交差の検出は Balaban のアルゴリズムを用いるので，$O(s)$ 作業領域で，$O(s \log s + K_{S(i)})$ 時間で列挙できる．第二段階では，Agarwal と Sharir [1] の技法を用いて，$O(s^{1+\epsilon})$ の領域に $\mathcal{D}_i$ を保持する．$\mathcal{D}_i$ の構築に要する時間は $O(s^{(1+\epsilon)^2})$ である．このデータ構造を用いると，一本の $s' \in S \setminus S_i$ と交差する S_i の線分を，$O(s^{1/2(1+\epsilon)} + K_{S(i),\{s\}})$ 時間で発見できる．よって，$Int(S_i, S \setminus S_i)$ の列挙に要する時間は，$O(ns^{1/2(1+\epsilon)} + K_{S(i),S(1,\ldots,\lceil n/s \rceil)})$ である．

以上より，全体の計算時間 $T(n)$ と定理を得る．

$$\begin{aligned}T(n) &= \sum_{i=1}^{\lceil \frac{n}{s} \rceil} \mathrm{O}(s\log s + K_{S(i)} + s^{(1+\epsilon)^2} + ns^{\frac{1}{2}(1+\epsilon)} + K_{S(i),S(1,\ldots,\lceil \frac{n}{s} \rceil)}) \\ &= \mathrm{O}\left(\frac{n}{s}s\log s + \frac{n}{s}s^{(1+\epsilon)^2} + \frac{n}{s}ns^{\frac{1}{2}(1+\epsilon)} + K_{S(i,\ldots,\lceil \frac{n}{s} \rceil)} + K_{S(1,\ldots,\lceil \frac{n}{s} \rceil),S(1,\ldots,\lceil \frac{n}{s} \rceil)}\right) \\ &= \mathrm{O}(n\log s + ns^{\epsilon^2+2\epsilon} + n^2 s^{-\frac{1}{2}(1-\epsilon)} + K).\end{aligned}$$

定理 3.3.1 S を平面上に与えられた n 本の線分集合とし，S を $\mathrm{O}(1)$ から $\mathrm{O}(n)$ までの範囲の任意のパラメータとする．このとき，ϵ を任意に小さな正の定数として，$\mathrm{O}(s^{1+\epsilon})$ の作業領域で，

$$\mathrm{O}(n\log s + ns^{\epsilon^2+2\epsilon} + n^2 s^{-\frac{1}{2}(1-\epsilon)} + K)$$

の時間で S 内の全ての交点を求めるアルゴリズムが存在する．ただし，K は交点の総数である．

また，作業領域を $\mathrm{O}(s)$ としたとき，以下の計算時間を得る．

定理 3.3.2 S を平面上に与えられた n 本の線分集合とし，S を $\mathrm{O}(1)$ から $\mathrm{O}(n)$ までの範囲の任意のパラメータとする．このとき，$\mathrm{O}(s)$ の作業領域で，

$$\mathrm{O}\left(\frac{n}{1+\epsilon}\log s + ns^{1+\epsilon-\frac{1}{1+\epsilon}} + n^2 s^{-\frac{1-\epsilon}{2(1+\epsilon)}} + K\right)$$

の時間で S 内の全ての交点を求めるアルゴリズムが存在する．ただし，K は交点の総数である．

3.4 水平線分と垂直線分の交差列挙

前節のアルゴリズムとデータ構造を用いて，領域と時間にトレードオフがあるアルゴリズムを設計できる．しかし，CSW のデータ構造を用いた方法は，理論上 $\mathrm{O}(s)$ の作業領域で計算可能であるが，実際の構造は非常に複雑である．

ここで，与えられる線分が水平垂直線分のみであれば，簡単な方法で線分交差列挙のトレードオフがあるアルゴリズムを設計できる．まずは，配列 S に与えられた n 本の線分が水平線分と垂直線分だけであるとき，S の交差する線分対を列挙する効率の良いアルゴリズムを提案する．簡単のために，全ての線分の端点は一般の位置にあると仮定する．平面走査法 [5, 6] を用いて線分交差を発見する．これは，走査線と呼ばれる垂直な直線で平面をスキャンすることで，交差する線分対を発見する．スキャン時に走査線と交差する線分を読み取り，その線分達の交差を調べる (図 3.4 参照)．

まずは，必要なデータ構造について説明する．一つは，走査線と呼ばれる垂直線 ℓ と交差する水平線分を，平衡 2 分探索木 $\mathcal{T}$ の内部節点に蓄えたものである．そのような水平

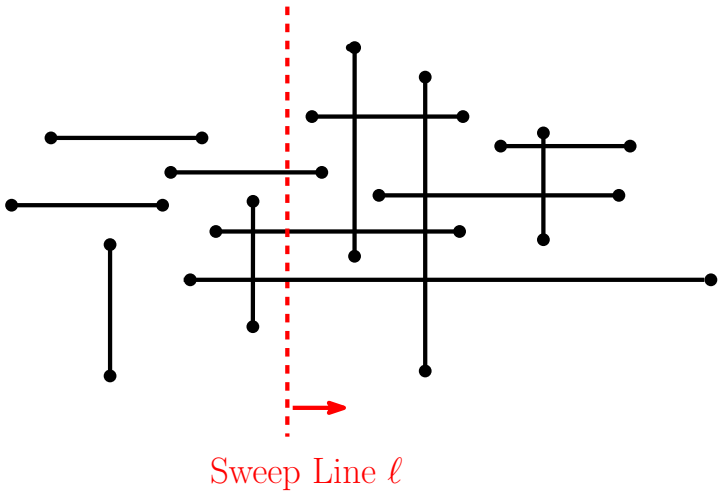


図 3.4: 水平・垂直線分の平面走査法

線分の集合を走査線状態と呼ぶ (図 3.5 参照)．探索木 $\mathcal{T}$ は，ℓ が平面を走査しているとき，現在 ℓ と交差する S の水平線分を y 座標の順序で保持する．ℓ が左から右に動くとき，走査線状態は動的に変化する．例えば，走査線が水平線分の左端点に達した時は，新たにその水平線分と交差するようになるので，走査線状態に加えなければならない．一方，右端点に達した時はその線分と交差しなくなるので，走査線状態から削除する必要がある．$\mathcal{T}$ は平衡 2 分探索木であるから，線分の挿入や削除は $O(\log n)$ 時間でできる．また，各線分は $\mathcal{T}$ に高々一回しか格納されないので，この木の保持に必要な領域は $O(n)$ である．

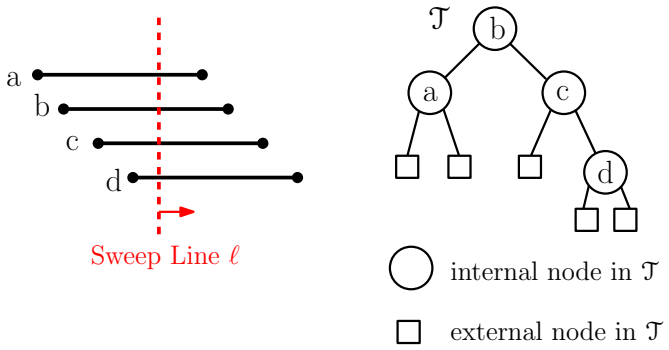


図 3.5: 平面の走査線状態と探索木 $\mathcal{T}$

もう一つは，イベント点を蓄えるためのイベントリスト $\mathcal{Q}$ である．イベント点とは水平線分の左端点，右端点，垂直線分の x 座標である．走査線が平面を左から右へ移動するとき，連続的に移動するのではなくイベント点と呼ばれる点を離散的に訪れる．走査線が各イベント点を訪れたとき，線分の交差検出や $\mathcal{T}$ の更新などを行う．2 つのイベント点を p, q としたとき，q の x 座標より p の x 座標が小さいならば，走査線は先にイベント点 p へ訪れるようにする．そこで，走査線が次に訪れるべきイベント点を，x 座標の昇順に従ってイベントリスト $\mathcal{Q}$ に蓄える．

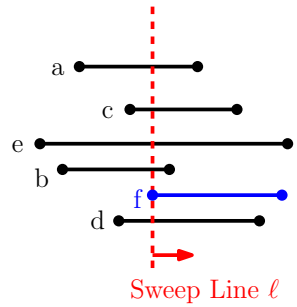
次に交差列挙のアルゴリズムについて説明する．始めに，Υを初期化する．また，Sの線分の端点をx座標に従って昇順にソートし，イベントリストΩを得る．このリストを順に見て各イベント点に従って以下の操作を行う．よって，このイベントリストを得るのに$O(n \log n)$時間を要する．明らかに$O(n)$の記憶領域が必要となる．イベント毎の操作を以下に示す.

$e \in \Omega$を現在のイベント点とする

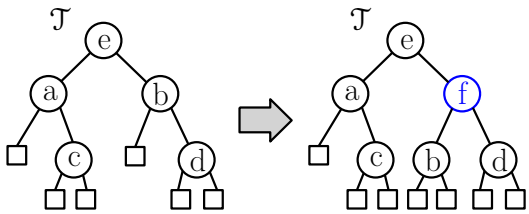
eが水平線分s_hの左端点なら，
Υに水平線分s_hを追加する (図 3.6 参照).

eが水平線分s_hの右端点なら，
Υから水平線分s_hを削除する (図 3.7 参照).

eが垂直線分s_vのx座標なら，
s_vの上端点と下端点との間にある水平線分をΥから見つけて，その線分たちを列挙する.

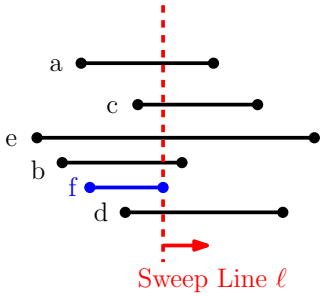


(a) 走査線状態に線分 f を追加

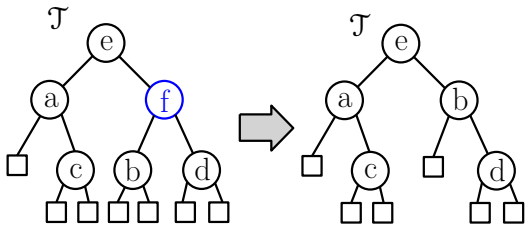


(b) $\mathcal{T}$ の変化

図 3.6: e が s_h の左端点イベントのとき



(a) 走査線状態から線分 f を削除



(b) $\mathcal{T}$ の変化

図 3.7: e が s_h の右端点イベントのとき

３番目の操作について，もう少し詳しく述べる．垂直線分 s_v と交差する水平線分は，Υ を中間順走査することで発見できる．即ち，s_v の上端点と下端点との間にある水平線分だけを求める．そのような線分だけを求めるためには，Υ の探索において，左の子に進むとき，s_v の上端点より上に位置する水平線分に対応する節点には訪問しないようにする．同様に，右の子に進むとき，s_v の下端点より下に位置する線分の節点には訪問しないようにする．後は，訪れた節点に対し，その節点に格納されている線分と交差判定し，交差するときに出力する．よって，s_v との交差判定は訪れた節点だけである．Υ は 2 分木であるから，訪れる節点の個数は列挙される節点の個数に線形である．s_v の上端点と下端点の探索は，Υ が平衡 2 分探索木であるから，$O(\log n)$ 時間である．従って，問い合わせの時間は $O(\log n + K')$ である．ここで，K' は s_v と交差する Υ の水平線分の数である．

垂直線分の数は $O(n)$ であるから，全体の線分交差を列挙する時間は $O(n \log n + K)$ である．他の操作に関して，まず走査線状態 Υ における水平線分の挿入，削除は 1 線分当たり $O(\log n)$ を要する．また，Ω のソートは $O(n \log n)$ 時間で得られる．従って，$O(n \log n + K)$ の計算時間で，水平線分と垂直線分の交差を列挙できる．また，必要となる記憶領域は $O(n)$ で十分である．

3.5 水平線分と垂直線分の交差列挙 ($O(s)$ の作業領域)

入力線分集合 S が水平線分と垂直線分のみであるとき，交差線分対を求めるための記憶領域と計算時間のトレードオフがあるアルゴリズムを考える．入力の線分は読み出し専用配列に格納されていると仮定し，$O(s)$ の作業領域が利用可能であるとする．線分の本数は n とする．

アルゴリズムは非常に簡単で，二段階の処理で行われる．前処理として，S の線分を部分集合 $S_1, \ldots, S_{\lceil n/s \rceil}$ に分割する．各部分集合には高々 s 本の線分を含むようにする．

第一段階は，各 $S_i (i = 1, \ldots, \lceil n/s \rceil)$ に対して交差する線分対を求める．線分対の列挙するために，前節の平面走査アルゴリズムと $O(s)$ の作業領域を用いる．第二段階は，2 つの部分集合の和集合 $S_i \cup S_j$ に対して，交差する線分対を列挙する．このとき，S_i を水平線分の集合 S_{ih} と垂直線分の集合 S_{iv} に分割する．S_j についても同様に，S_jh と S_jv に分ける．そして，$S_{ih} \cup S_{jv}$ と $S_{jh} \cup S_{iv}$ それぞれに対して，平面走査法を用いて交差する線分対を列挙する．アルゴリズムを以下に示す．

Algorithm 4 S の全ての交差する線分対を列挙 (水平・垂直線分)

入力: 水平線分と垂直線分のみで構成された集合 S. 但し，$|S| = n$

出力: 全ての交差する線分対

仮定: 入力された線分は読み出し専用配列に格納

$S \to S_1 \cup S_2 \cup \cdots \cup S_{\lceil n/s \rceil}$
for each $S_i,\ i$=1, . . . , $\lceil n/s \rceil$ **do** {
  (第一段階)
  $Int(S_i)$ を列挙
  (第二段階)
  for each $S_j,\ j$=1, . . . , $\lceil n/s \rceil$ **do** {
    S_i を水平線分のみから成る部分集合 S_{ih} と垂直線分のみから成る部分集合 S_{iv} に分割．同様に S_j も S_{jh}, S_{jv} に分割.
    $Int(S_{ih}, S_{jv})$ を計算して出力.　　// 平面走査法
    $Int(S_{jh}, S_{iv})$ を計算して出力.　　// 平面走査法
  }
}

第一段階の $Int(S_i)$ の列挙では，平面走査法により $\mathrm{O}(s)$ の作業領域を用いて，$\mathrm{O}(s \log s + K_{S(i)})$ 時間で線分交差を計算できる．$K_{S(i)}$ は与えられた線分集合で交差する線分対の数を表す．第二段階では，S_i と S_j から $S_{ih} \cup S_{jv}$ と $S_{jh} \cup S_{iv}$ を生成する．これは，$\mathrm{O}(s)$ 時間で計算できる．生成したものそれぞれに対し，$\mathrm{O}(s)$ 作業領域を用いることで，平面走査法より $\mathrm{O}(s \log s + K_{S(i),S(j)})$ 時間で $Int(S_i, S_j)$ を列挙できる．従って，全体の計算時間は $\mathrm{O}((n^2/s) \log s + K)$ で，必要な記憶領域は $\mathrm{O}(s)$ で十分である．

この計算時間は最適である．なぜなら，s =O(1) のときの計算時間は $\mathrm{O}(n^2)$ となり，s =O(n) のときの計算時間は $\mathrm{O}(n \log n + K)$ となるため，それぞれの作業領域を利用する最適なアルゴリズムの計算時間と一致するからである．

このアルゴリズムは本章の初めに提案された手法と似ているが，一つの交差対はちょうど一回しか出力されない．一体，何が違うのだろうか．S'_i と S'_j を一般の線分集合として，初めに提案されたアルゴリズムが何度も同じ交点を列挙してしまう原因が何であったかをもう一度述べる．アルゴリズムの第二段階において，$S'_i \cup S'_j$ に対して $Int(S'_i, S'_j)$ だけを求めるとき，$Int(S'_i)$ と $Int(S'_j)$ も同時に検出してしまうためであった．これは，交差を列挙するアルゴリズムが，与えられた $S'_i \cup S'_j$ に含まれる全ての線分交差を検出してしまう他，重要な点として，S'_i と S'_j に交差する線分対が存在する場合があるからである．従って，S'_i と S'_j に交差が存在しないならば，$S'_i \cup S'_j$ の全ての線分交差を検出したとしても，$Int(S'_i, S'_j)$ だけが検出される．S は水平線分のみの集合と，垂直線分のみの集合に分割できるので，交差対がない 2 つの集合を構築できる．

逆に，初めのアルゴリズムが同じ交差を検出しないためには，一般の線分集合 S' に対して，各々に交差が存在しないような部分集合に分割できればよい．しかし，一般の線分集合 S' に対しては，単に交差しないように分割すれば良い訳ではない．例えば，S' の全ての線分がお互いに他と交差している状態であるとすると，分割数は $\mathrm{O}(|S'|)$ で，素朴な方法で分割をすると $\mathrm{O}(|S'|^2)$ 時間を要する (図 3.8 参照)．このとき，どのように分割すれば効率よく処理できるかは知られていない．一方，S の線分は水平か垂直な線分であるから，必ず高々二つの集合に分割できる．また，その分割は $\mathrm{O}(|S|)$ 時間でできる．

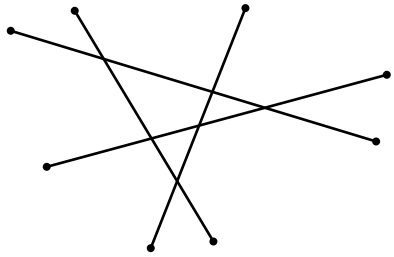


図 3.8: 分割数が最悪となる場合

本節の結果として以下の定理を得る．

定理 3.5.1 S を平面上に与えられた n 本の水平・垂直集合とし，S を $\mathrm{O}(1)$ から $\mathrm{O}(n)$ までの範囲の任意のパラメータとする．Algorithm 4 は，$\mathrm{O}(s)$ の作業領域で，$\mathrm{O}((n^2/s)\log s + K)$ の時間で S 内の全ての交点を求める．ただし，K は交点の総数である．

3.6 c 種類の傾きを含む線分の交差列挙

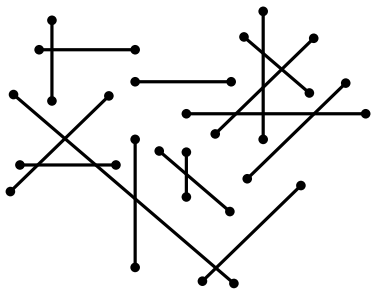


図 3.9: 4 種類の傾きをもつ線分集合の例

与えられた線分集合 S が高々 c 種類の傾きに分類できるならば (図 3.9 参照)，水平・垂直線分の交差列挙と似たようなトレードオフをもつアルゴリズムを設計できる．以下にアルゴリズムを記述する．

Algorithm 5 S の全ての交差する線分対を列挙

入力: c 種類の傾きを含む線分集合 S. 但し，$|S| = n$

出力: 全ての交差する線分対

仮定: S は読み出し専用配列

```
S → S_1 ∪ S_2 ∪ ··· ∪ S_⌈n/s⌉
for each S_i, i=1,...,⌈n/s⌉ do {
  (第二段階)
  Int(S_i) を列挙
  (第一段階)
  for each S_j, j=1,...,⌈n/s⌉ do {
    S_i → S_i1, S_i2, ..., S_ic        // c 種類の傾きで S_i を分割
    S_j → S_j1, S_j2, ..., S_jc        // c 種類の傾きで S_j を分割
    for each k,l=1,...,c do {
      Int(S_ik, S_jl) を見つける.      // 平面走査法
    }
  }
}
```

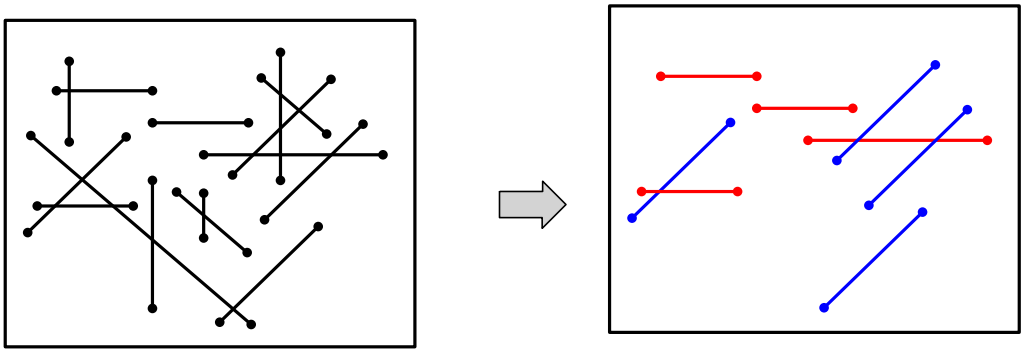
S_i と S_j のそれぞれを c 種類の傾きで分類するには $\mathrm{O}(s)$ 時間を要する．S_i, S_j はそれぞれ以下のように分割される．

$$\begin{aligned} S_i &\rightarrow S_{i1}, \ldots, S_{ic} \\ S_j &\rightarrow S_{j1}, \ldots, S_{jc} \end{aligned}$$

その上で，$k, l = 1, \ldots, c$ に対して，$Int(S_{ik}, S_{jl})$ を列挙する (図 3.10 参照)．これは $\mathrm{O}(s)$ の作業領域で，$\mathrm{O}(s \log s + K_{S(ik),S(jl)})$ 時間で計算できる．ある S_{ik} と交差を持つ可能性があるグループの数は高々 c であるので，全部で高々 c^2 回の線分交差の列挙が行われる．しかし，c は定数であるから $Int(S_i, S_j)$ の計算時間は $\mathrm{O}(s \log s + K_{S(ik),S(jl)})$ である．

従って，全体の計算時間は，$\mathrm{O}((n^2/s) \log s + K)$ で，使用する作業領域は $\mathrm{O}(s)$ である．以上を定理としてまとめる．

定理 3.6.1 S を平面上に与えられた n 本の c 種類の傾きをもつ線分集合とし，S を $\mathrm{O}(1)$ から $\mathrm{O}(n)$ までの範囲の任意のパラメータとする．Algorithm 5 は，$\mathrm{O}(s)$ の作業領域で，$\mathrm{O}((n^2/s) \log s + K)$ の時間で S 内の全ての交点を求める．ただし，K は交点の総数である．



(a) 4 種類の傾きをもつ線分集合

(b) 各 2 種類の異なる傾きの線分集合に対して交差を計算

図 3.10: 4 種類の傾きをもつ線分集合の交点を求める.

第4章　おわりに

本稿では，2つの計算幾何学の問題に対するメモリ制約付きアルゴリズムを提案した．最遠点ボロノイ図の描画に関しては，定数領域を用いたアルゴリズムを設計した．今後の課題として，記憶領域と計算時間のトレードオフがあるアルゴリズムの設計である．特に，$O(\sqrt{n})$ の作業領域で，効率よく最遠点ボロノイ図を描画するアルゴリズムは大変興味深いテーマである．

線分交差列挙問題に対しては，$O(s)$ の作業領域を用いて，交差する線分対を全て列挙するアルゴリズムを提案した．また，c 種類の傾きをもつ線分交差であれば，比較的簡単に効率の良いトレードオフがあるアルゴリズムを設計できることを示した．しかし，一般の線分の場合は傾きの種類は無限であるため，互いが交差しないような線分集合を生成することは簡単ではない．一方，本稿で提案するアルゴリズムとデータ構造は複雑で実用とし難い．従って今後は，より簡単な手法で一般の線分の交差を列挙するアルゴリズムの提案を目指す．

謝辞

本研究を進めるにあたり，日ごろより懇切丁寧な指導を賜りました浅野哲夫教授には深く感謝致します．また，上原隆平教授，岡本吉央特任准教授，清見礼助教をはじめとする研究室の方々には，公私に渡り大変お世話になりました．ここに厚く御礼申し上げます．最後に，この場所で新しく出会えた友人達に心からの敬意と感謝の意を表します．

参考文献

[1] P. K. Agarwal and M. Sharir. Applications of a New Space-Partitioning Technique. *Discrete Comput. Geom.*, 9:11–38, 1993.

[2] A. Aggarwal, L. J. Guibas, J. B. Saxe, and P. W. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon, *Discrete Coput. Geom.*, 4:591–604, 1989.

[3] T. Asano and M. Konagaya. Zero-space data structure for farthest-point Voronoi diagram, *Abstract of the 4th Annual Meeting of Asian Association for Algorithm and computation*, p.44, 2011.

[4] I. J. Balaban. An optimal algorithm for finding segments intersection. In *Proc. 11th Sympos. on Comput. Geom.*, pp. 211–219, 1995.

[5] J. L. Bentley and T. A. Ottman. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput. C*-28 pp. 643–647, 1979.

[6] M.de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applicaations*, 3rd edition, SpringerVerlag, 2008.

[7] M. Blum, R. W. Floyd, V. Pratt, R. Rivest, and T. Tarjan. Time bounds for selection, *J. Comput. System Sci.* 7, pp.448-461, 1973.

[8] H. Brönnimann, E. Y. Chen, and T. M. Chan. Towards in-place geometric algorithms and data structures, In *Proc. 20th Annual ACM Sympos. on Comput. Geom.*, pp.239–246, 2004

[9] B. Chazelle, M.Sharir, and E. Welzl. Quasi-optimal upper bounds for simplex range searching and new zone theorems. In *Proc. 6th ACM Sympos. on Comput. Geom.*, pp.23–33, 1990.

[10] E. Y. Chen and T. M. Chan. A space-efficient algorithm for segment intersection, In *Proc. 15th Canadian Conference on Comput. Geom.*, pp. 68–71, 2003.

[11] E. Y. Chen and T. M. Chan. Multi-pass geometric algorithms, In *Discrete and Computational Geometry,* 37(1), pp. 79–102, 2007.

[12] J. I. Munro and V. Raman. Selection from read-only memory and sorting with minimum data movement, *Theoretical Computer Science* 165, pp. 311–323, 1996.

[13] O. Reingold. Undirected st-connectivity in log-space, In *Proc. ACM Symp. on Theory of Cmputing*, pp. 376–385, 2005.