DEIM Forum 2011 E4-3

大規模位置イベントデータからの空間的近接パターン発見アルゴリズム

関根　　渓†　　宇野　毅明††　　有村　博紀†††

† 北海道大学工学部　〒 060–0814 北海道札幌市北区北 14 条西 9 丁目
†† 国立情報学研究所　〒 101–8430 東京都
††† 北海道大学大学院情報科学研究科　〒 060–0814 北海道札幌市北区北 14 条西 9 丁目
E-mail: †libra1015sekine@hotmail.com, ††uno@nii.ac.jp, †††arim@ist.hokudai.ac.jp

あらまし　近年，移動体センサーやストリーム技術の発展により，大規模な位置イベントデータから特徴的な規則性やパターンを発見するための効率よい技術が望まれている．そのための基礎技術として，一次元空間において，与えられた幅の区間と，入力点集合の共通部分として表される異なるすべての部分集合を列挙する問題を考察し，この問題を解く線形時間アルゴリズムを与える．また，二次元空間において，与えられた大きさの長方形と，入力点集合の共通部分として表される異なるすべての部分集合を列挙する問題を考察し，この問題を解くアルゴリズムについて議論する．

キーワード　空間データマイニング，位置データ，幾何図形マイニング，空間クラスタリング，色つき点集合

Efficient Discovery of Spatial Proximity Patterns from Massive Point Location Event Data

Kei SEKINE†, Takeaki UNO††, and Hiroki ARIMURA†††

† Faculty of Engineering, Hokkaido University, N14 W9, Sapporo 060–0814, JAPAN
†† National Institute of Informatics, Tokyo 101–8430, JAPAN
††† Grad. School of Inform. Sci. and Tech., Hokkaido University, N14 W9, Sapporo 060–0814, JAPAN
E-mail: †libra1015sekine@hotmail.com, ††uno@nii.ac.jp, †††arim@ist.hokudai.ac.jp

1. はじめに

一次元または二次元空間に与えられた点集合から，点の近接関係を記述する空間的パターンを発見する問題は重要である．このような空間的近接パターンの発見アルゴリズムは，画像からの位置依存の特徴抽出や，地理的配置からの空間データマイニング，移動体マイニングなどの応用に役立つ．

本稿では，空間的近接パターン発見のための基礎技術として，一次元空間と二次元空間において，与えられた幅の区間と，入力点集合の共通部分として表される異なるすべての部分集合を列挙する問題を考察する．主結果として，一次元空間において，この問題を解く「尺取虫法」の線形時間アルゴリズム 1DimInchWorm を与える．さらに，このアルゴリズムを二次元空間へ拡張し，アルゴリズム 2DimInchWorm を与える．最後に，一次元のアルゴリズムを実装し，人工データ上で評価実験を行う．

関連研究として，最近，近接文字列集合 [3] やグラフモチーフ [2] の名前で，文字列やグラフ上の近接パターン発見問題が盛んに研究されている．これらは従来の構造マイニングの制約を緩めて，要素データの順序や構造情報を無視して，距離関係のみを抽出するようなマイニングと考えられる．本稿は，これらの近接モチーフを空間データに拡張するものである．

2. 準　　備

本稿では，$\mathbb{R}$ と $\mathbb{N} = \{0, 1, \ldots\}$ で，それぞれ，実数全体と自然数全体の集合を表す．実数 $a, b \in \mathbb{R}$ に対して，上限 a と下限 b をもつ $\mathbb{R}$ 上の**閉区間**（closed interval）は $[a, b] = \{\, x \in \mathbb{R} \mid a \leqq x \leqq b \,\}$ であり，**開区間**（open interval）は $(a, b) = \{\, x \in \mathbb{R} \mid a < x < b \,\}$ である．半閉区間 $[a, b)$ や $(a, b]$ も，同様に定める．整数 $i \leqq j$ に対して，**整数区間**を $[i..j] = \{i, i+1, \ldots, j\}$ と定める．以下では，とくに断らないかぎり，実数を $x, y, \alpha, \beta, \ldots$ で表し，実数ベクトルを $\mathbf{x}, \mathbf{y}, \alpha, \beta, \ldots$ で表す．

2.1　d 次元空間のデータと領域

正整数を $d \geqq 0$ とおき，**次元**（dimension）という．集合 $\mathbb{R}^d$ を**実 d 次元空間**（continuous d-dimensional space, or contin-

uous d-dim space）という．本稿では，次元が $d=1$ と $d=2$ の場合のみを扱う．実 d 次元空間の**点**（d-dim point）は，ベクトル $\mathbf{x}=(x_1,\ldots,x_d)\in\mathbb{R}^d$ である．われわれの問題の入力データは $\mathbb{R}^d$ の**点集合**（point set）

$$P=\{\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(n)}\}\subseteqq\mathbb{R}^d\ (n\geqq 0)$$

である．

$\mathbb{R}^d$ の部分集合を**領域**（region）という．**サイズベクトル**（size vector）は，実 d ベクトル $\alpha=(\alpha_i)_{i=1}^d\in\mathbb{R}^d$ である．任意の点 $\mathbf{x}\in\mathbb{R}^d$ とサイズベクトル $\alpha\in\mathbb{R}^d$ に対して，d **次元直交領域**（d-dim orthogonal range），または，d **次元長方形**（d-dim hyper rectangle）とは，次のように定められる領域

$$B=B(\alpha,\mathbf{x})=\prod_{i=1}^{d}[x_i,x_i+\alpha_i]=\{\,\mathbf{z}\in\mathbb{R}\mid x_i\leqq z\leqq x_i+\alpha_i\,\}$$

である．ここに，B は**サイズ**（size）$\alpha_1\times\cdots\times\alpha_d$ の長方形 $\prod_{i=1}^d[0,\alpha_i]$ を，左隅の頂点が位置（position）$\mathbf{x}$ に重なるよう平行移動したものである．この B を**ウィンドウ**（window）または**箱**（box）とも呼ぶ．

以下では，$\mathcal{B}_d(\alpha)$ で，サイズ α の d 次元長方形全体の族を表す．1 次元の場合は，領域 B は区間 $B=[x,x+\alpha]$ であり，α **区間**とよぶ．2 次元の場合は，$\mathbf{x}=(x,y)$ と $\alpha=(\alpha,\beta)$ とすると，B は辺が x 軸と y 軸に平行な矩形 $B=[x,x+\alpha]\times[y,y+\beta]$ であり，$\alpha\times\beta$ **長方形**と呼ぶ．

2.2 直交領域走査問題

$\mathcal{B}_d(\alpha)$ をサイズが α の長方形の族とする．入力点集合 P の部分集合を**パターン**（pattern）という．本稿では，d 次元の長方形 $B\in\mathcal{B}_d(\alpha)$ と入力点集合 P の**共通部分** $B(P)=P\cap B$ として表されるような異なる全ての点集合を，$\mathcal{B}_d(\alpha)$ に関するパターンと考えて，これらを列挙するアルゴリズムを考察する．このとき，$B(P)\subseteqq P$ を，B **が表す** P **の点集合**（the point set defined by B）という．

d 次元直交領域走査問題（d-dim range scan problem）

入力：任意の入力点集合 $P=\{p_1,\ldots,p_n\}\subseteqq\mathbb{R}^d$ とサイズベクトル $\alpha\in\mathbb{R}^d$.

出力： サイズ α の長方形が表す P の部分集合の族

$$\mathcal{R}=\{\,P\cap B(\alpha,\mathbf{x})\mid\mathbf{x}\in\mathbb{R}^d\,\}$$

の要素 $S\in\mathcal{R}$ $(S\subseteqq P)$ を，重複することなく，全て列挙せよ．ここに，$B(\alpha,\mathbf{x})$ は，幅 α で位置 $\mathbf{x}$ の長方形である．

これは，言いかえると，固定されたサイズの長方形によって定義される P 上のパターンを列挙する問題である．以下の節では，$d=1$ 次元と $d=2$ 次元の場合について，上記の問題を解く効率よいアルゴリズムを与える．

3. 解の数の解析

アルゴリズムを与える前に，d 次元直交領域走査問題の解の数 s_d の上限と下限を解析する．解の数 s_d について，次が成立する．P を任意の入力点集合とし，$\alpha>0$ を任意の実数とする．以下で，$n=|P|$ は入力点集合 P のサイズである．

［補題 1］（1 次元の部分集合の特徴づけ） P を任意の 1 次元の入力点集合とし，$\alpha>0$ を任意の実数とする．ある正実数 $\varepsilon>0$ に対して，任意の区間 B に対して,「ある入力点 $z\in P$ が，B の右辺の上または，B の左辺の ε だけ内側に存在する」という条件（$z_1=x_1+\alpha_1$ または $z_1=x_1+\varepsilon$）を満たすある区間 $B'=B(\alpha,x)$ が存在して，$B\cap P=B'\cap P$ が成立する．

［証明］ 補題 3 の (i) の証明と同様にして示される． □

補題 1 の条件を満たすとき，α 区間 B' は**正規**（canonical）であるという．

［補題 2］ $d=1$ 次元の場合，$s_1=\Theta(n)$ が成立する．

次の補題は，2 次元の場合の上限を示すのに重要である．P を任意の 2 次元の入力点集合とし，$\alpha=(\alpha_1,\alpha_2)$ を任意のサイズベクトルとする．

［定義 1］ 2 次元の点集合 P に対して，α 長方形 $\hat{B}$ が**正規**であるとは，ある正実数 $\varepsilon>0$ に対して，$\hat{B}$ が次の条件 (i) と (ii) を満たすことをいう．

(i) ある入力点 $\mathbf{z}\in P$ が，B の右辺の上または，B の左辺の ε だけ内側に存在する（$z_1=x_1+\alpha_1$ または $z_1=x_1+\varepsilon$）．

(ii) ある入力点 $\mathbf{w}\in P$ が，B の下辺の上または，B の上辺の ε だけ内側に存在する（$w_2=x_2+\alpha_2$ または $w_2=x_2+\varepsilon$）．

［補題 3］（2 次元の部分集合の特徴づけ） 任意の α 長方形 $B=B(\alpha,\mathbf{x})$ に対して，正規な α 長方形 $\hat{B}$ が存在して，$B\cap P=\hat{B}\cap P$ が成立する．

［証明］ 正数 ε を十分に小さくとると次が示せる．(i) 長方形 B の左辺と右辺に平行な直線 ℓ_1 と ℓ_2 ではさまれた幅 α_1 の帯状の領域（ストライプ）X を考える．初めに，B をこの帯上の領域内で上方（y 軸の正方向）へ平行移動したときに，B の上辺と下辺のそれぞれに最初に触れる入力点 $\mathbf{y}$ と $\mathbf{z}$ までの距離を，それぞれ，f_2 と r_2 とおく．(i.a) もし $f_2\leqq r_2$ ならば，部分集合 $B(P)$ を変えることなく，B を平行移動して上辺を点 $\mathbf{y}$ 上におくことができる．(i.b) もし $f_2>r_2$ ならば，部分集合 $B(P)$ を変えることなく，B を平行移動して下辺を点 $\mathbf{z}$ から ε の内側におくことができる．(ii) 次に，上記の位置で，B の上辺と下辺に平行な直線 ℓ_1 と ℓ_2 を考える．(i) と同様にこれら二つの直線ではさまれた高さ α_2 の帯状の領域 Y の内側において，B を右側（x 軸の正方向）に平行移動することで，条件を満たす点 $\mathbf{w}$ の存在を示せる．以上より，示された． □

［定理 1］ $d=2$ 次元の場合，上限 $s_2=O(n^2)$ が成立する．

［証明］ （上限）補題 3 より，サイズ α の任意の長方形 B について，補題 3 の条件をみたす入力点の対 $\mathbf{z},\mathbf{w}\in P$ が存在することが保障される．ただし $\mathbf{z}=\mathbf{w}$ も許す．さらに，各点対に対して可能な長方形の位置は，高々 4 通りしかない．よって，これらの可能な対と長方形の位置をすべて列挙することで，部分集合の候補は高々$4n$ で上から抑えられることがわかる． □

［定理 2］ $d=2$ 次元の場合，下限 $s_2=\Omega(n^2)$ が成立する．

［証明］ （下限）任意の α_1,α_2 と，任意の $k\geqq 1$ を考えて，$\delta_i=\alpha_i/k$ とおく．このとき，次のように，原点 $o=(0,0)$ を中心として，十字状に $n=4(k-1)+1=4k-3$ 個の点を配置す

る．まず，原点 $o=(0,0)$ に点を 1 個おく．次に，x 軸上の区間 $[-\alpha_1,+\alpha_1]$ 上に $2k-1$ 個の点を間隔 δ_1 で均等に配置する．さらに，同様に y 軸上の区間 $[-\alpha_2,+\alpha_2]$ 上に $2k-1$ 個の点を間隔 δ_2 で均等に配置する．このとき，全ての組 $0 \leqq i,j \leqq k$ に対して，サイズ α かつ位置 $\mathbf{x}=(-\delta_1 i,-\delta_2 j)$ の長方形は P の異なる部分集合を含む．よって，少なくとも解の数は $k^2=(1/4)n^2$ 以上である．これが無限個の $n \geq 0$ に対して成立するから，よって $s_2=\Omega(n^2)$ が成立する． □

［系 3］ $d=2$ 次元の場合，解数は $s_2=\Theta(n^2)$ である．

［定理 4］ $d \geqq 1$ 次元の場合，任意のサイズ α の任意の n 個の点からなる点集合は，少なくとも $s_2=\frac{n}{2^d}$ が成立する．

［証明］ サイズベクトル $\alpha=(\alpha_i)_{i\leqq i\leqq d}$ に対して，実空間 $\prod_{1\leqq i\leqq d}[0..2\alpha_i]$ を考える． □

定理 1 の解数の上限から，2 次元の場合の素朴なアルゴリズムを構成することができる．これには，$O(n^2)$ 個の入力点対を列挙し，さらに 4 通りの長方形の配置を列挙する．この場合，$O(n^2)$ で含まれる点のリストを計算したあとリストをキーとした辞書を用いて重複解の除去をする必要があり，最悪時に全体で $O(n^3)$ 時間と $O(n^2)$ 領域が必要である．5. 節では，$O(n\log n+s)$ 時間と $O(1)$ 領域の出力線形時間アルゴリズムを与える．

4. 一次元空間のアルゴリズム

本節では，一次元実数空間 $\mathbb{R}$ におけるアルゴリズムである ε-尺取虫法を説明する．

4.1 尺取虫法の概要

ε-尺取虫法による一次元空間における点集合パターンの列挙アルゴリズム 1DimInchWorm を，Algorithm 1 に示す．ウィンドウの初期位置を決定する手続き SetWindow を，Algorithm 2 に与える．このアルゴリズムは，指定された 1 次元空間 $U=[l,h]$ 内の幅 m の区間 B すべて考えて，入力点集合 P と B の共通部分として表される部分集合を列挙する．

入力として，サイズ $n \geqq 0$ の入力点集合 P が，x 座標で整列された n 個の点の配列 $P=P[0]\cdots P[n-1]$ に格納されて与えられたと仮定する．ε-尺取虫法では，補題 3 による特徴付けに基づいて，ウィンドウの位置を表す二組の変数 (L,R) と (r,f) を漸増的に更新しながら，点集合パターンを重複なく全て列挙する．ここに，(L,R) は，ウィンドウに含まれる点の中で最も左および右に位置する点の添え字の組 $(0 \leqq L \leqq R \leqq n-1)$ であり，(r,f) は，ウィンドウの左端と最左点 $P[L]$ の距離 r とウィンドウの右端と最右点 $P[R]$ の距離 f の組である（図 1）．

以後，簡便のために解となる部分集合 $S=P\cap B$ を組 (L,R) と同一視する．

［補題 4］ 任意の入力点集合 P と区間サイズ $m>0$ に対して，1 次元領域走査問題の解 $\mathcal{R}=\{[L_i,R_i]\}_{i=1}^{s}$ $(s \geqq 0)$ を，任意の $1 \leqq i<s$ に対して，次の条件 (i)–(iii) を満たすように整列させることができる．

(i) $L_i \leqq L_{i+1}$ かつ $R_i \leqq R_{i+1}$，かつ

(ii) $L_{i+1}-L_i \leqq 1$ かつ $R_{i+1}-R_i \leqq 1$，

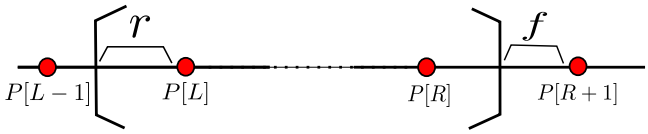

図 1 ε-尺取虫法の説明

(iii) $R_i \neq R_{i+1}$ または $L_i \neq L_{i+1}$．

［証明］ すべての解を，L の昇順が先で，次が R の昇順の辞書式順序で，$(I_i=[L_i,R_i])_{i=1}^{s}$ と整列したと仮定する．このとき，重複を含まないから，すべての組は異なる．もし上記の (i) と (iii) を満たさない組があるとすると，このとき，$L_i<L_{i+1}$ かつ $R_i>R_{i+1}$ が成立する．しかし，ウィンドウの幅が定数 α なのでこれは不可能であり，(i) と (iii) は成立する．もし (ii) が成立しないと仮定し，さらに $L_{i+1}-L_i>1$ と仮定する．このとき，$L_i \leqq L \leqq L_{i+1}$ と $R_i \leqq R \leqq R_{i+1}$ なる添え字 L,R で，$L \notin \{L_i,L_{i+1}\}$ なものが存在する．このとき，$I=[L,R]$ は正しく幅 m の区間と P の共通部分を表しており，同時に I_i,I,I_{i+1} の順で補題の順序を満たす．しかし，このことは I_i の後者が I_{i+1} であることに反する．$R_{i+1}-R_i>1$ のときも同様に矛盾が導かれる．よって，(ii) も成立する． □

上の補題より，r と f の大小関係の比較に基づいて L と R を高々一つづつ進めながら，常に一次元空間上でウィンドウを前進させることができる．われわれは，この挙動が尺取虫の動きに似ていることから，アルゴリズムを 1DimInchWorm（ε-尺取虫法）と名付けた．

4.2 ウィンドウの初期設定

アルゴリズムは，初めに手続き $SetWindow$ を呼び出し，1 次元空間 $U=[l,h]$ の左端 l をはみ出さないように，ウィンドウの初期位置を決定する．具体的には，ウィンドウの右端が，x 座標 $l+m$ の右にある最左の点に重なるようにウィンドウを配置する．

さらにウィンドウの右端が空間の右端 h をはみ出さないように，走査の終了を知らせる「番兵」（sentinel）として，P の最後の点の次に，出力されない仮想的な点を追加し，その x 座標を $P[n]=h$ とする．ウィンドウの初期位置が決まったら最初の解として出力する．

4.3 ウィンドウの移動

次に，アルゴリズムは，ウィンドウ B の右端が一次元空間の右端に達するまで，以下の操作を繰り返す：r と f の値を比較し，その結果によって次に B をどう動かすかを決定し，(L,R) の値を変えてウィンドウ B を動かしたら，それに含まれる入力点を出力し，(r,f) の値を更新する．

解となる点集合 B(P) を重複なく全て見つけるためには，ウィンドウに含まれる点集合の構成が変化する度に，以下の 2 点に着目してチェックを行う．

1. 点がウィンドウに入った瞬間．このときは，点がウィンドウの右端上に置かれた状態である．

2. 点がウィンドウから出た瞬間このときは，点がウィンドウの左端から外側に ε だけ離れた状態である．

Algorithm 1 手続き 1DimInchWorm$(P[0..n-1], n, m, u)$

$N \leftarrow n$; $P[N] \leftarrow \infty$; //A sentinel for the right end.
$((L,R),(r,f),N) \leftarrow$ SetWindow(P,n,m,u);
ウィンドウ (L,R) とそれが含む点の情報を出力する;
while $R < N$ **do** //ウィンドウを一つ進める
  if $(r > f)$ **then**
    $(L,R) \leftarrow (L, R+1)$;
    $(r,f) \leftarrow (m - (P[R] - P[L]),\ P[R+1] - P[R])$;
  else if $(r = f)$ **then**
    $(L,R) \leftarrow (L, R+1)$;
    $(r,f) \leftarrow (0,\ P[R+1] - P[R])$;
  else if $(r < f)$ **then**
    if $(R = L$ **and** $P[R+1] - P[L] > m)$ **then**
      $(L,R) \leftarrow (L+1, R+1)$;
      $(r,f) \leftarrow (m - (P[R] - P[L]),\ P[R+1] - P[R])$;
    else
      $r_{\mathrm{old}} \leftarrow r$;
      $(L,R) \leftarrow (L+1, R)$;
      $(r,f) \leftarrow (P[L] - P[L-1] - \varepsilon,\ f - r_{\mathrm{old}} - \varepsilon)$;
    end if
  end if
  ウィンドウ (L,R) とそれが含む点の情報を出力する;
end while

Algorithm 2 手続き SetWindow(P,n,m,u)

$N \leftarrow n$;
$(L,R) \leftarrow (0,0)$;
while $P[R] < l + m$ **and** $R < N$ **do**
  $R \leftarrow R+1$;
end while
while $P[L] < P[R] - m$ **and** $L \leqq R$ **do**
  $L \leftarrow L+1$;
end while
$r \leftarrow P[L+1] - P[L]$; $f \leftarrow P[R+1] - P[R]$;
return $((L,R),(r,f),N)$;

以下に，ウィンドウの動かし方の場合分けの詳細を与える．

Case 1: $r > f$ のとき．このとき，ウィンドウの右端と $P[R+1]$ が重なるようにウィンドウを右にずらすと，点 $P[R+1]$ が新しく入る．(図 2)

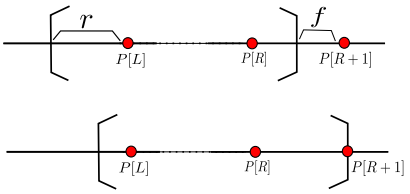

図 2　$r > f$ のときの動かし方

Case 2: $r = f$ のとき．$P[L], P[R+1]$ がそれぞれウィンドウの右端，左端に重なるようにウィンドウをずらすと，点 $P[R]$ が新しく入る．(図 3)

Case 3: $r < f$ のとき．この場合は，以下の二つの場合

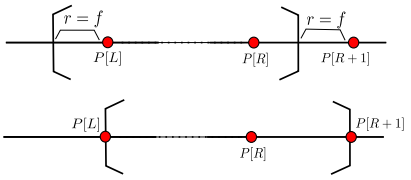

図 3　$r = f$ のときの動かし方

がある．

Case 3.(1): $P[L] = P[R]$ かつ $P[R+1] - P[R] > m$ のとき．このとき，ウィンドウの右端と $P[R+1]$ が重なるようにウィンドウをずらすと，点 $P[L]$ が出て，点 $P[R+1]$ が入る．(図 4)

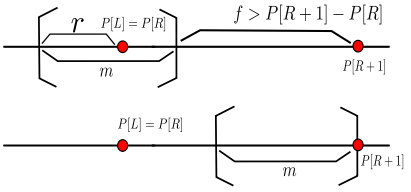

図 4　$r < f$ かつ $P[L] = P[R]$ かつ $P[R+1] - P[L] > m$ のときの動かし方

Case 3.(2): 上記以外のとき．このとき，ウィンドウの左端から $P[L]$ が外側に ε だけ離れるようにウィンドウをずらすと，点 $P[L]$ が出る．(図 5)

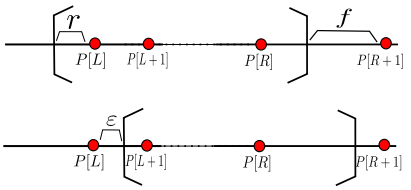

図 5　$r < f$ かつ上記以外のときの動かし方

以上で，1 次元実数区間上でのアルゴリズムは終わりである．

4.4　1 次元離散空間の場合

1 次元離散空間 $U = \{0, \ldots, u-1\}$ $(u \geqq 1)$ の場合にも，アルゴリズムを少し修正して適用できる．この場合は，ウィンドウを格子上の整数きざみでしか動かせないので，図 6 のように $f > r$ かつ $f - r = 1$ となる場合の場合分けが追加される．このとき，ウィンドウから点 $P[L]$ が出て，点 $P[R+1]$ が入る．

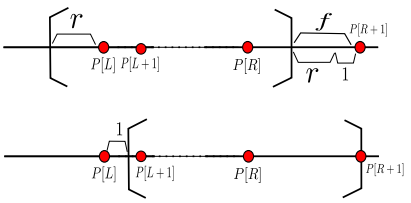

図 6　一次元グリッド空間における $r < f$ かつ $r - f = 1$ のときのウィンドウの動かし方

ε-尺取虫法は前述した極小数 ε を追加することで場合分けが少なくなり，より単純なアルゴリズムとなっている．

Algorithm 3 手続き SetWindowBounded$(P, n, m, (l, h), \varepsilon)$

```
Check if l ≦ h;
L ← P.successor(l);
R ← P.successor(l + m);
N ← P.successor(h + ε);
if P[R] = N then
    return EMPTY;
end if
if P[L] = N then
    return EMPTY;
end if
r ← P[L + 1] − P[L];  f ← P[R + 1] − P[R];
return ((L, R, r, f), N);
```

4.5 計算量の解析

1 次元実数空間におけるアルゴリズム 1DimInchWorm の計算時間を議論する．初めに，入力点集合 P が，あらかじめ x 軸に関して昇順で整列されていると仮定する．そうでない場合は，$O(n \log n)$ 時間で整列すれば良い．

手続き SetWindow は，3 行目と 3 行目の while 文が共に高々 $O(n)$ 回しか実行されないので，明らかに $O(n)$ 時間である．

［補題 5］ 図 1 の手続き 1DimInchWorm は，すべての解を出力する．

［証明］ 補題 4 の条件を満たすように，1 次元領域走査問題の解を $\{I_i = [L_i, R_i]\}_{i=1}^{s}$ $(s \geqq 0)$ と整列したとする．添え字 $1 \geqq i \leqq s$ に関する帰納法で，第 i 番目の解 I_i がアルゴリズムによって見つかることを示す．初めに，$i = 1$ のとき，明らかに，最初の解 I_0 は SetWindow によって見つけられる．次に $i - 1$ 以下の時，主張が成立すると仮定する．このとき，補題 4 より，第 i 番目の解 I_i は前節の Case 1 から Case 3 のいずれかに該当し，アルゴリズムの対応する条件部で見つけることができる．よって示された． □

［補題 6］ 図 1 の手続き 1DimInchWorm は，4 行目の while 文が実行される度に，一回ウィンドウを動かし，少なくとも一つ解を出力する．さらに全ての出力は重複しない．

［証明］ アルゴリズムの構成から，while 文が実行される度に，条件部のいずれかが実行され，解 (L, R) が出力される．さらに，各条件部では L または R のどちらかの値が 1 増加し，補題 4 の順序を満たすことがわかる．よって，同じ区間は二度出力されない． □

補題 6 と補題 5 からアルゴリズム 1DimInchWorm の正当性がいえる．補題 6 からアルゴリズム中の while 文の実行回数は高々解の数 s で抑えられる．さらに，アルゴリズムの構成から，while 文 1 回の実行は $O(1)$ 時間である．よって，次の定理が示せる．

［定理 5］ サイズ n の任意の入力点集合 P と区間幅 $\alpha > 0$ が与えられたとき，図 1 のアルゴリズム 1DimInchWorm は，1 次元直交領域走査問題を $O(n + s) = O(n)$ 時間と，$O(n)$ 領域で解く．

4.6 領域制限付き 1 次元直交領域走査問題への拡張

これまでの節では，全領域 $U = [0, u]$ 全体で考えたが，ここでは，入力の一部として U の部分領域 $H = [l, h]$ $(0 \leqq l \leqq h \leqq u)$ が動的に指定されたときに，H に区間を制限して $O(n \log n)$ 時間で部分集合を列挙するアルゴリズム 1DimBoundedInchWorm を与える．

そのために，先の図 1 の主アルゴリズム 1DimInchWorm を次のように修正する．まず第一番目に，入力点集合を，$O(1)$ 時間の逐次操作と，$O(\log n)$ 時間の探索が可能なデータ構造で置き換える．データ点は，挿入後に整列されて管理されるものとする．これには，平衡二分探索木を用いればよい [1]．領域は $O(n)$ である．

二番目に，主アルゴリズム 1DimInchWorm の 2 行目の副手続き SetWindow の呼び出しを，図 3 の手続き SetWindowBounded に置き換える．これは，もとの手続き SetWindow で初期位置を求めていた逐次操作の代わりに，二分探索による後者演算（successor）を用いて，初期値を求める．ここに，後者演算 successor(x) は，与えられた座標 x 以上で，最小の座標をもつ入力点の添え字（ポインタ）を $O(\log n)$ 時間で返す演算である．

［補題 7］ 任意の領域 $H = [l, h]$ の中にパターンを制限したとき，図 3 の手続き SetWindowBounded は $O(\log n)$ 時間で，領域に含まれる初期解と，上限を超えた点のポインタを見つける．

三番目に，この手続きの返り値が $EMPTY$ だったときには，呼び出し側の主アルゴリズム自体を解を出さずに終了させるよう主プログラムを修正する．以上の修正を加えたアルゴリズムを 1DimBoundedInchWorm とする．

このとき，入力領域に制限を加えた場合にも，完全性の補題 5 はそのまま成立することが容易にわかる．解依存の計算量の補題 6 は，次のように修正されて成立する．

［補題 8］ $H = [l, h]$ の中の領域にパターンを制限したとき，図 1 の手続き 1DimInchWorm は，4 行目の while 文が実行される度に，一回ウィンドウを動かし，少なくとも一つ領域 H 内の解を出力する．さらに全ての出力は重複しない．

これより，領域制限付の 1 次元問題において，領域 $H = [l, h]$ と区間幅 α が動的に与えられるクエリ版に対して，次の定理が成立する．

［定理 6］ 前処理されたサイズ n の任意の入力点集合 P に対して，クエリとしてパターンの区間幅 $\alpha > 0$ と任意の区間 $H = [l, h] \subseteqq [0, u]$ が与えられるとき，修正されたアルゴリズム 1DimBoundedInchWorm は，区間 H に領域制限された 1 次元直交領域走査問題を，前処理時間は $O(n \log n)$ 時間と $O(n)$ 領域を用いて，1 回あたり $O(s)$ 時間で解く．

5. 二次元空間における全点集合パターン列挙アルゴリズム

本節では，二次元空間における全点集合パターン列挙アルゴリズムである二次元尺取虫法 ($2dim\text{-}Inchworm\text{-}Method$) を与える．

5.1 基本的なアイディア

このアルゴリズムでは，まず点集合を x 座標の昇順でソートして点列 Y を作る．Y の点に対して，二次元空間上で，$u \times a$ の水平な帯 (以下ではこれをストライプと呼ぶ) を上から下へ移動させながら (図 7)，その内部の点を x 座標の昇順で整列して点列 X を作る．さらに，各時点ごとにストライプの内部で，サイズ $a \times b$ の長方形型のウィンドウ B を左から右に移動させて (図 7)，B に含まれる点集合を計算する．

以上の y 軸方向のストライプの移動と，ストライプ内での x 軸方向のウィンドウの移動の両方に，先の 4. 節で与えた 1 次元実数空間におけるアルゴリズム 1DimInchWorm を用いる．以下，詳細を説明する．

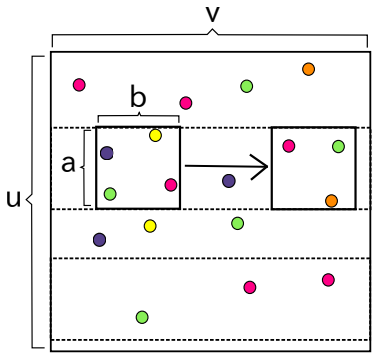

図 7　ウィンドウとストライプ

5.2 ウィンドウの動かし方

交差尺取虫法の擬似コードをアルゴリズム 4 に示す．はじめに，交差尺取虫法におけるウィンドウの動かし方について述べる．ストライプ内部の点は予め x 座標に関してソートされて XS に格納されているので，XS 対して前節で述べた ε-尺取虫法を用いれば良い．このソート方法については後述する．一次元空間との大きな違いは，同ストライプ内に，x 座標が等しい点の組が存在し得るということである．そういった点の組をどう扱うかという問題は，予め前処理を行うことで，その組の点のうちの一つにまとめたり，ある一点以外を順次 ε だけ微小変動させるという方法をとることで解決できる (図 8)．本稿では前者の方法を用いることにする．具体的には，同座標の点は連結リストとしてひとまとめにしておく方法をとる．

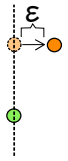

図 8　点の微小変動

5.3 ストライプの動かし方と重複解の除去

次にストライプの動かし方について述べる．ストライプの動かし方もウィンドウと同様で，あらかじめ用意された y 座標に関して昇順でソートされた点列を参照しながら，ε-尺取虫法を用いれば良い．ただし，ストライプを動かした後，ウィンドウを動かしていく場合，新たにストライプに追加，またはストライプから削除された点の x 座標近辺以外を走査するとき，古いストライプで出力した解をもう一度出力してしまう可能性がある (図 9)．その問題を回避するために，変化した点の近辺のみでウィンドウを走査する必要がある．そこで，手続き $MarginInterval$ を用いて新しいストライプ内の適切な走査範囲の集合を求める．手続き $MarginInterval$ の処理には $O(n)$ 時間かかる．

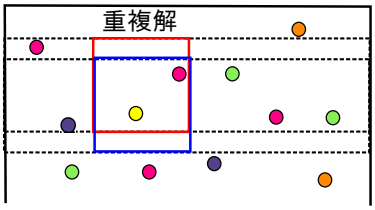

図 9　重複解の例

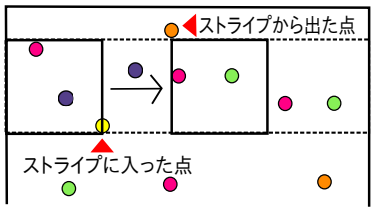

図 10　ストライプの入出点

5.4 ストライプ内の点のソート

次に，ストライプ内の点を x 座標でソートする方法であるが，ストライプを動かす度に毎回ソートしていたのでは，ストライプを動かす度に最悪で $O(n \log n)$ かかってしまい，アルゴリズム全体として大幅なタイムロスになることが予想できる．したがってここでは次のような方法を用いる．最初にストライプを二次元空間上に置いたときに，Y を参照しながら，初期ストライプに入る点を X に入れて，x 座標に関してソートする．ストライプを動かしたら，Y を参照しながらストライプから出る点を X から取り除き，ストライプに入る点を X に加える．以上の操作を $O(n \log n)$ 未満の計算時間で行うためには，XS は配列ではなく連結リストや二分木などのデータ構造を用いる必要があるが，本稿では二分木を用いることにする．まず，最初のストライプに含まれる点集合に関する二分木を構成するが，構成と同時に点の順序付けが為されるため，ソートが終了した状態となる．ここまでで $O(n \log n)$ かかる．さらに，二分木を用いるため X への点の挿入，削除は手続き $insert$ および $delete$ を用いて $O(\log n)$ で行うことが可能であり，挿入を行った時点で点は適切な位置，つまりソートされた状態で格納される．また，一度ストライプから削除された点は二度とストライプに追加されることがないということを考慮すると，点の挿入，削除にはアルゴリズム全体で $O(n \log n)$ かかることがわかる．

Algorithm 4 手続き $2dim\text{-}Inchworm\text{-}Method(u, v, a, b, P)$

```
for i = 0 to n − 1 do
    insert(P[i], Y)
end for
X ← SetStripe()
ε-Inchworm-Method(b, n, m, X)
while ストライプが下端に到達していない do
    if J が空ではない then
        for i = 0 to インターバル数 − 1 do
            ε-Inchworm-Method(b, n, m, J[i])
        end for
    end if
    GetInOut(Y, Inc, Dec)
    for i = 0 to 入点数 − 1 do
        insert(Inc[i], X)
    end for
    for i = 0 to 出点数 − 1 do
        delete(Dec[i], X)
    end for
    J ← MarginInterval(Inc, Dec, X)
end while
```

5.5 計算時間

Y の生成に $O(n \log n)$ 時間，X への点の挿入，削除に全体で $O(n \log n)$ 時間，手続き $GetInOut$ に $O(n \log n)$ 時間，手続き $MarginInterval$ に $O(n)$ 時間，そして出力される点集合パターンの総数を S とすると，その計算には全体で $O(S)$ かかる．したがって計算時間は $O(n \log n) + O(S)$ 時間と見積もれる．

6. 実　　験

本章では，前章で述べた ε-尺取虫法を計算機上で実装し，データセットを用いて実験を行った．はじめに，データと実験方法について述べ，そのあとで結果と考察について述べる．

6.1 デ ー タ

実験には，テキストとして次のデータを用いた．簡便さのため，データ点の座標は非負整数を用いた．具体的には，次の手続きで点集合データを生成した．

- 点集合データの生成

一次元空間の幅 l と，点集合の要素数 n を受け取り，$\{0, ..., l\}$ までの数字から n 個の整数を乱数を用いて重複なく昇順にソートされた状態で出力する．

6.2 実験方法

第三章の手続き $\varepsilon\text{-}Inchworm\text{-}Method$ を C 言語で実装し，実験を行った．全ての実験は以下の環境で行った．(表 1)

OS	Ubuntu GNU/Linux 11.10
CPU	Intel Core2 Duo CPU U7500 @ 1.06GHz × 2
メモリ	2.0 GiB
プログラミング言語	C
コンパイラ	GNU Compiler Collection 4.60

表 1　実 験 環 境

実験では，アルゴリズムの各入力パラメータを目的に応じて変化させた時の計算時間をそれぞれ求めた．具体的には，以下のような手順で行った．

（1） テキストデータとパラメータをプログラムに読み込ませる．

（2） ウィンドウの初期位置を決定するサブルーチンに入る直前で，時刻 t_1 を計測する．

（3） 空間上の全ての解の計算が終了した時点で，時刻 t_2 を計測する．

（4） ウィンドウの初期位置決定計算を除いた計算時間 $T = t_2 - t_1$ を記録する．

時間計測において，全動作時間から，次の時間を除いて計測した．

- データとパラメータの入力時間．これは，ファイルの入出力に時間がかかるためである．
- 解の出力時間．これを除いた理由は次の通りである．アルゴリズムの主要部分である，ウィンドウを動かすという作業に，一つの解あたり $O(1)$ 時間かかるのに対し，解を出力するサブルーチンの配列を辿る過程は，一つの解あたり最悪でウィンドウの幅分の $O(m)$ 時間かかってしまい，最終的に全体の実行時間が後者の計算時間に依存し，アルゴリズムの評価が正しく行えないと予想出来たためである．

なお，時間計測には gettimeofday 関数を用い，マイクロ秒オーダーの精度で計測した．次節から，実験手法の詳細とその結果及び考察を述べる．

6.3 実験 1：入力データに対する計算時間

この実験では，ウィンドウ幅 m を固定し，空間の幅 l とデータ数 n を同じ割合で増加させた．一定長さあたりの点の密度 n/l を一定にすることで，データ数が大きくなった時に計算時間がどのように変化していくかを調べた．

実験の繰り返し回数 $i = 1, 2, .., 100$ のとき，$l = 10000 \times i$, および $n = 1000 \times i, m = 100$ として実験を行った．

6.3.1 結　　果

図 11 に，縦軸に計算時間 T，横軸に入力データ数 n をとったグラフを示す．

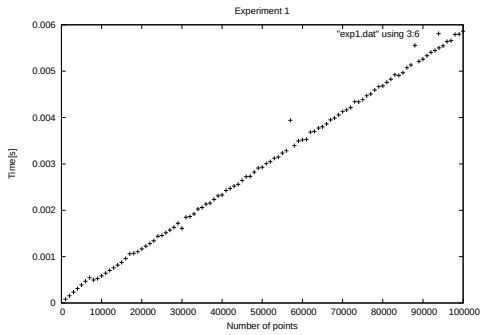

図 11　入力データに対する計算時間 1

6.3.2 考　　察

アルゴリズムの構造上，入力データが増加すると解の個数も増えていくので，計算時間もそれに伴って増加していくことが

予想される．結果の図 11 より，要素数が増加すると計算時間が線形に増加していることが確認できた．したがって，アルゴリズムの節でで議論したように，計算時間が線形時間となることが改めて確認できた．

6.4 実験 2：ウィンドウ幅と計算時間

この実験では，空間の幅 l とデータ数 n を固定して，ウィンドウ幅 m を増加させていき，それによって計算時間がどのように変化していくかを調べた．

実験の繰り返し回数 $i = 1, 2, .., 100$ のとき，$l = 1000000$，および $n = 100000, m = 10000 \times i$ として実験を行った．

6.4.1 結　　果

縦軸に解の個数，横軸にウィンドウ幅をとったグラフを図 12 に示し，縦軸に計算時間 T，横軸にウィンドウ幅をとったグラフを図 13 に示す．

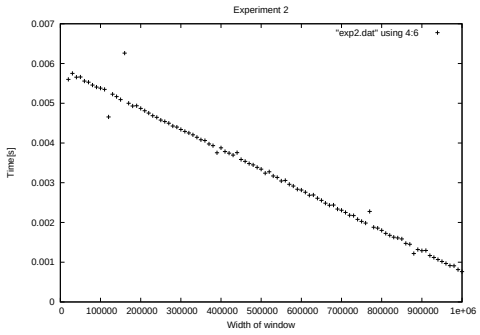

図 12　ウィンドウ幅と計算時間 1

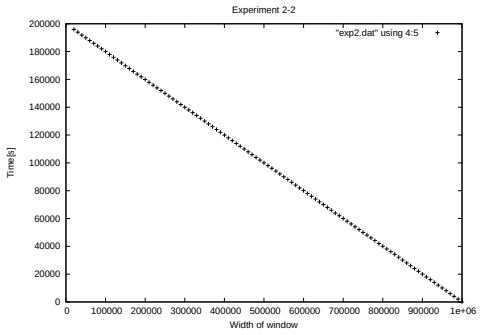

図 13　ウィンドウ幅と計算時間 2

6.4.2 考　　察

アルゴリズムの構造上，ウィンドウ幅が大きくなるほど，空間上でウィンドウを動かせる範囲が制限されていくので，出力される解の個数が少なくなり，またそれに伴って計算時間も小さくなることが予想される．図 12 より，ウィンドウ幅が大きくなり，空間の幅に近づくにつれて出力される解の個数が線形的に減少していくことが確認できた．アルゴリズムの節で述べたように，アルゴリズムの計算時間は解の個数に関して線形であることが確認できた．

6.5 実験 3：点の密度と計算時間

この実験では，空間の幅 l とウィンドウ幅 m を固定した状態で，データ数 n を増加させた．データ数の増加，つまり一定長あたりの点の密度 n/l が増加するにつれて，計算時間がどのように変化していくかを調べた．

実験の繰り返し回数 $i = 1, 2, .., 990$ のとき，$l = 100000$，および $n = (i \times 100) + 1000, m = 1000$ として実験を行った．

6.5.1 結　　果

縦軸に計算時間 T，横軸に点の密度 n/l をとったグラフを図 14 に示し，縦軸に解の個数，横軸に点の密度 n/l をとったグラフを図 15 に示す．

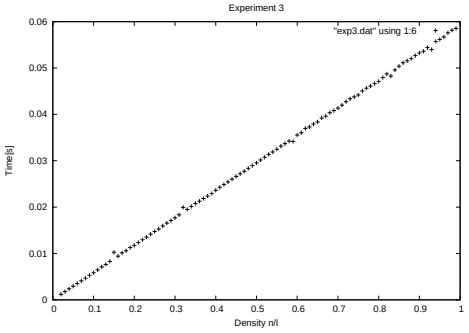

図 14　点の密度と計算時間 1

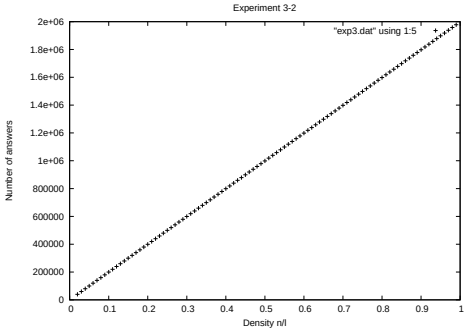

図 15　点の密度と計算時間 2

6.5.2 考　　察

アルゴリズムの計算時間は要素数に関して線形であるから，空間内の点の密度が高くなる，というパラメータの変化は要素数を増加させることに等しい．したがって，実験 1 に近い結果が得られることが予想される．図 15 より，密度の増加に対して，解の個数は線形に増加していることが確認できた．さらに，図 14 より，空間内の密度が高くなるにつれて，計算時間が線形に増加していることが確認できた．

7. おわりに

本稿では，空間的近接パターン発見のための基礎技術として，一次元空間において，与えられた幅の区間と，入力点集合の共通部分として表される異なるすべての部分集合を列挙する問題を考察した．主結果として，この問題を解く線形時間アルゴリズム (尺取虫法，*Inchworm-Method*) を与えた．さらに，この問題の二次元空間への拡張について議論した．

結果としては，一次元空間におけるアルゴリズムに関しては，実験を行うことで，予め想定していたアルゴリズムの挙動が概ね正しかったということを確認することができたが，今回提案した手法が，ナイーブなアルゴリズムと比べてどれほどの改善が見られるかという検査を行うことができなかった．二次元空間におけるアルゴリズムに関しては，アルゴリズムの基本

的なアイディアは提案出来たものの，実装および実験には至らなかった．

今後の検討事項として挙げられるのは，一次元空間におけるナイーブなアルゴリズムを実装し，比較実験を行うことで，提案手法の有用性を検証すること，そしてさらにその上で，二次元空間上のアルゴリズムの実装を行い，人工データではなく，地理データ等を用いてその挙動を確認することである．

文　献

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, Third Edition, MIT Press, pp.1312, 2009.

[2] R. Dondi, G. Fertin, S. Vialette, Finding approximate and constrained motifs in graphs, in: the 22nd Annual Symposium on Combinatorial Pattern Matching (CPM'11), Vol. 6661 of Lecture Notes in Computer Science, Springer, 2011, pp. 388–401.

[3] K. Sadakane, H. Imai, Fast algorithms for k-word proximity search, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E84-A (9) (2001) 2311–2318.

[4] Endre Boros, Toshihide Ibaraki, Hiroya Ichikawa, Koji Nonobe, Takeaki Uno and Mutsunori Yagiura, Heuristic Approaches to the Capacitated Square Covering Problem, *Pacific Journal of Optimization*, 1 (2005) 465-490.