修 士 論 文

距離変換の一般化に関する研究

北陸先端科学技術大学院大学
情報科学研究科 情報処理学専攻

野木 慶太

2009 年 3 月

修 士 論 文

距離変換の一般化に関する研究

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 准教授
審査委員　石原 哉 准教授

北陸先端科学技術大学院大学
情報科学研究科 情報処理学専攻

0710055 野木 慶太

提出年月: 2009 年 2 月

概 要

距離変換とは、2 値行列に対して各 0 要素から見たとき、その要素から最も近い 1 要素までの距離を求める問題である．この問題はパターン認識など画像処理の様々な分野で応用されていることで知られている．

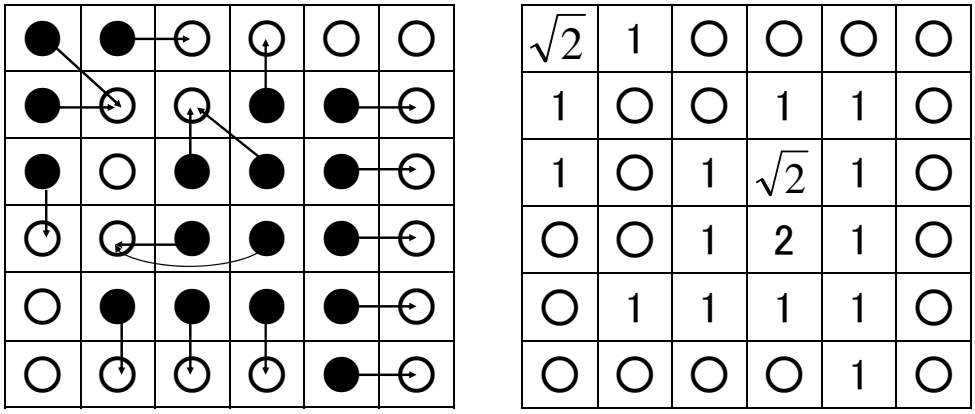
本論文では，距離変換の一般化として，実数値の行列に対して，各要素からそれより大きい値を持つ最も近い要素までの距離を求める問題を考える．距離変換については，すでに線形時間で解くアルゴリズムが提案されているが，一般化距離変換については，まだ距離変換のような効率の良いアルゴリズムは提案されていない．このため，一般化距離変換を解く効率の良いアルゴリズムについて考える．特に，$n \times n$ 実数値行列が与えられたとき，一般化距離変換を $\mathrm{O}(n^2 \sqrt[3]{n})$ 時間で解くアルゴリズムを提案する．

目 次

第1章　はじめに

1.1　研究の背景

距離変換とは0,1からなる2値行列に対して，各0要素から見たとき，最も近い1要素までの距離を求める問題である．図1.1に0要素を黒丸,1要素を白丸で表示した2値行列と各0要素から最も近い1要素までのユークリッド距離を行列に入力した例を示す.



(a) 各黒丸から最も近い白丸　　(b) 最も近い白丸までの距離の行列

図 1.1: 2値画像に対するユークリッド距離変換の例

この問題は特にユークリッド距離に対して考えられ，画像処理において様々なことに応用されている [1][2]．しかし，単純に距離の近い要素から順に調べていく方法では，$O(n^4)$ 時間かかってしまう．そのため，より効率の良いアルゴリズムが求められてきたが，1995年と1996年に初めて線形時間のアルゴリズムが考案された．1995年に提案されたKirkpatrick[3]らの方法はボロノイ図の考え方を用いたものである．また，1996年にHirata[4] によって提案された方法は放物線の下側エンベロープの計算に還元したものである．このように2値画像に対しては，ユークリッド距離変換を線形時間で解くアルゴリズムが知られている．しかしながら，実数値を要素とする行列に関しては，距離変換のような効率の良いアルゴリズムはまだ提案されていない．このため，一般化距離変換として，実数値行列が与えられたとき各要素からそれより大きい値を持つ要素までの最小距離を計算する方法について考える．この一般化距離変換は，地形図の尾根線を求めることなどに利用できると

考えられる．例えば，標高が行列の要素として与えられたとき，一般化距離変換を求めることで，尾根線の概形を推測することができる．

1.2 研究の目的

本稿では，距離変換の一般化について考える．距離変換は入力を 2 値行列で考えているが，距離変換の一般化として入力を実数値行列とし，各要素についてそれより大きな値をもつ要素までの最小距離を求める問題を考える．入力行列のサイズを $n \times n$ とするとき，各要素に対してより大きい値を持つ要素の中で最も近い要素を，近い順に近傍の要素を調べるという素朴なアルゴリズムが考えられるが，これでは $\mathrm{O}(n^4)$ 時間を必要とする．また，行列に含まれる要素が h 通りの値しか取らないときには，2 値行列に対する線形時間のアルゴリズムを h 回だけ繰り返すことにより $\mathrm{O}(hn^2)$ 時間のアルゴリズムが得られるが，繰り返し回数 h は n^2 になる場合があるので，最悪の場合 $\mathrm{O}(n^4)$ になりうる．これは行列の要素数の 2 乗に相当する．本論文では，この最悪時の計算複雑度を改善する．すなわち，一般の実数値行列が入力として与えられたとき，行列の各要素に対して，その値より大きな値を持つ要素までの最小距離を求める 2 乗より少ない計算時間の効率的なアルゴリズムを提案する．

1.3 本論文の流れ

本稿では，まず 2 章で一般化距離変換として，NLN 問題 (Nearest Larger Neighbors) を定義し，3 章で効率のよいアルゴリズムを提案する．次いで，4 章でより時間計算量の少ないアルゴリズムを得る方法について示す．

第2章　NLN問題

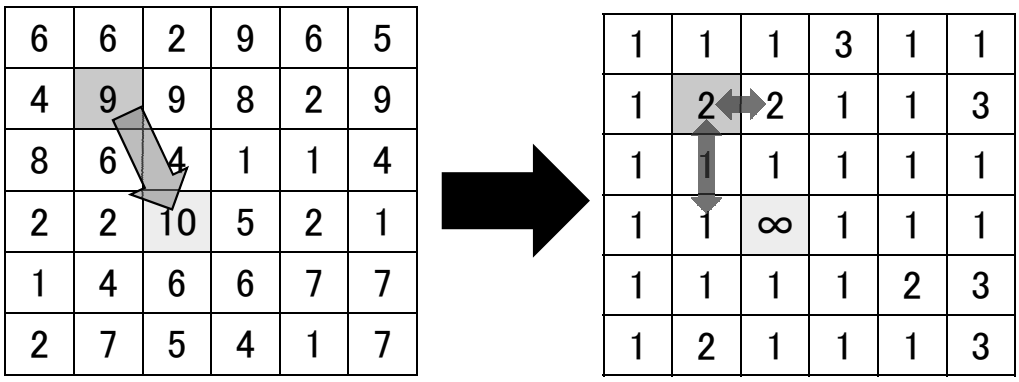
2.1　問題の定義

$n \times n$ 実数値行列 A が与えられたとき，各行列要素 (i,j) に対して，$A(i,j)$ より大きい値を持つ要素を (i,j) の優越要素として定義する．また，要素間の距離には，L_∞ 距離を用いることとする．L_∞ 距離を用いて多値画像の各要素に対して優越要素を求めることにより，特に与えられた画像に対して，対象図形の中心線を得るのに利用することができると考えられる．任意の要素 $(i,j),(i',j')$ の距離を $d((i,j),(i',j'))$ とすると，$d((i,j),(i',j'))$ は次のように書ける．

$$d((i,j),(i',j')) = \max\{|i-i'|,|j-j'|\}.$$

任意の要素 (i,j) に対して，最も近い優越要素 (i',j') までの距離 $D(i,j)$ を求める．すなわち，次のように定義される距離行列 D を求める．

$$D(i,j) = \begin{cases} \infty, & \text{A(i,j)が最大,} \\ \min\{d((i,j),(i',j')) \mid A(i',j') > A(i,j)\}, & \text{それ以外.} \end{cases}$$



(a) 入力の実数値行列 A　　(b) 行列 A に対する距離行列 D

図 2.1: 実数値行列に対する距離行列

例えば，図 2.1 のような実数値行列 A に対して，距離行列を求めることを考える．2 行 2 列の位置にある 9 という要素から見たとき，最も近い優越要素は 4 行 3 列にある 10 と

いう要素である．このとき，この二つの要素間の水平距離は 1 であり，垂直距離は 2 である．したがって L_∞ 距離では $d((2,2),(4,3)) = 2$ となり，$D(2,2)$ に 2 が書き込まれる．他の要素についても同様にして距離行列を求める．

2.2 自然な手法

与えられた行列の各要素に対して，L_∞ 距離の意味で最も近い優越要素までの距離を求める最も素朴なアルゴリズムは，行列の各要素 (i,j) の近傍要素を L_∞ 距離の昇順に順に調べて，最も近い優越要素を求めるというものである．図 2.2 のような行列の場合，行列のほぼ全体を調べることになるので，入力が $n \times n$ の行列の場合，各要素で $\mathrm{O}(n^2)$ の時間がかかるので計算時間は $\mathrm{O}(n^4)$ となる．このアルゴリズムの唯一の利点は, 作業領域が $\mathrm{O}(1)$ で済むという点である．特に，入力の行列は読み出し専用の配列として扱うことができる他，それ以外に作業配列を一切使わなくても各要素に対応する距離を求めることができるのは大きな利点である．

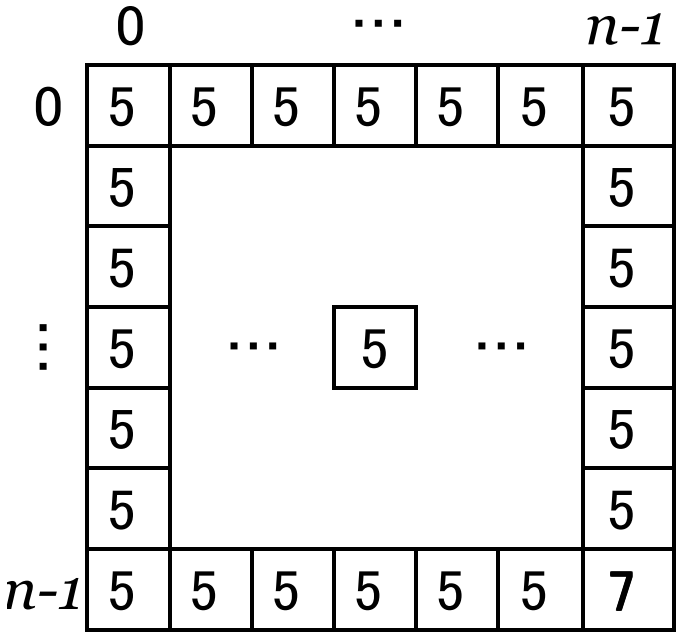


図 2.2: $\mathrm{O}(n^4)$ 時間かかる $n \times n$ 行列の例

第3章　優越要素の探索

L_∞ 距離における NLN 問題を解く $O(n^2\sqrt{n})$ 時間のアルゴリズムを提案する．まず，準備として双対変換と隠線除去問題について述べる [5].

3.1　準備

3.1.1　双対変換

平面上の点は x 座標値と y 座標値という二つのパラメータを持っている．また，平面上の垂直でない直線も傾きと y 切片という二つのパラメータによって決定することができる．ゆえに，平面上の点は平面上の直線と 1 対 1 に対応させることができる．このとき，点集合におけるある種の性質が直線集合に移したときに保存されるように変換することができる．そのような変換を総称して双対変換と呼ばれる．双対変換により変換された物体のことは元の物体の双対と呼ばれる．単純な双対変換として，平面上の点を直線に変換する図 3.1 のようなものが考えられる．平面上の点 $p = (p_x, p_y)$ の双対は，$p^* : y = p_x x - p_y$ を満たす直線として定義することができ，平面上の垂直でない直線 $l : y = mx + b$ の双対は，$l^* = (m, -b)$ として定義される．

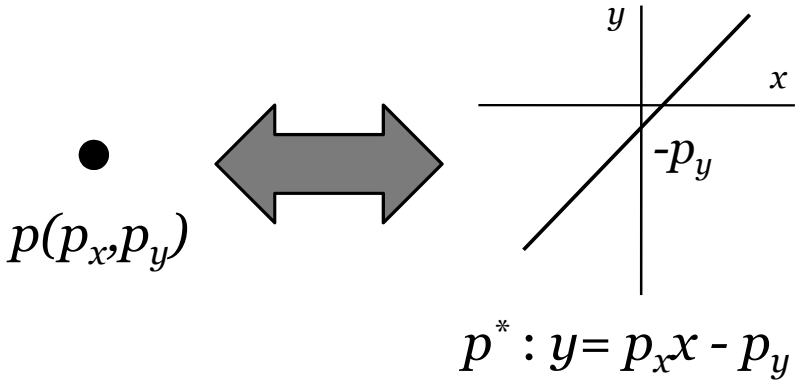


図 3.1: 点から直線への双対変換

双対変換では，主平面のある物体を双対平面上に変換するという言い方をする．このとき，主平面で成立した性質の中で双対平面でも成立する性質が存在する．たとえば，平面

上の点 $p = (p_x, p_y)$ を平面上の垂直でない直線 $l : y = mx + b$ に変換する双対変換を考えたとき，次のことが言える．

- p が l 上にあることと l^* が p^* 上にあることは同値である．つまり，
  $p \in l \Leftrightarrow l^* \in p^*$.
- p が l より上にあることと l^* が p^* より上にあることは同値である．つまり，
  $p_y > mp_x + b \Leftrightarrow -b > p_x m - p_y$.

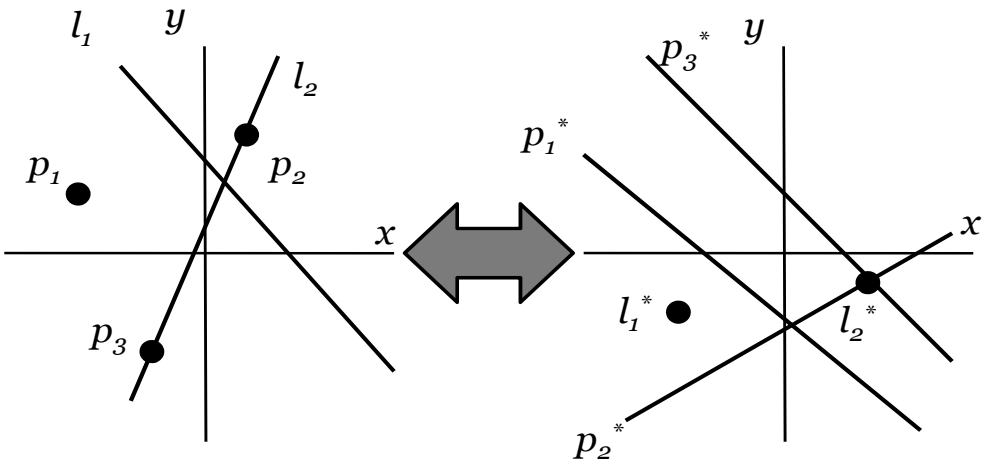


図 3.2: 双対変換で保存する性質

主平面で解くのが難しい問題であっても双対平面に変換することによって元の問題より解きやすくなることがある．そのため，双対変換によって元の問題を双対平面上での問題に変換し，双対平面上で解いた方法を主平面で同じように解くことで，主平面でも問題を解くことが可能となる．

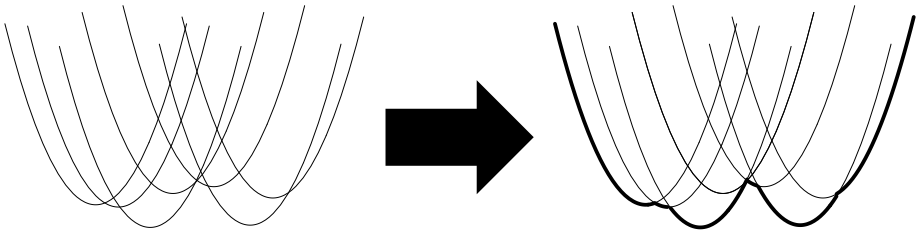
3.1.2 隠線除去問題

隠線除去問題とは，平面上に n 本の水平線分が与えられたとき，下方無限遠から見える部分を求める問題である．ここでは線分の集合でなく，特に放物線の集合に対して同じ問題を考える．すなわち，平面上に同じ形の n 本の放物線が与えられたとき，$y = -\infty$ から見える部分を求める．与えられた放物線がすべて同じ形であると仮定すると，任意の放物線 P_i は $P_i : y = (x - x_i)^2 + y_i$ という形で，一般性を失うことなく定義できる．このとき，任意の放物線 $y = (x - x_i)^2 + y_i$ と $y = (x - x_j)^2 + y_j$ に対して，下から見たとき放物線 $y = (x - x_i)^2 + y_i \rightarrow$ 放物線の交点 $\rightarrow$ 放物線 $y = (x - x_j)^2 + y_j$ の順になる．これは，どの二つの放物線を選んだとしても，放物線がすべて同じ形であることから必ず交点が一つだけであるためである (図 3.3).

(a) 同じ形の放物線　　(b) 違う形の放物線

図 3.3: 放物線の交点

そのため放物線を下方無限遠から見た時見える部分，すなわち放物線の集合に対する下側エンベロープに一つの放物線が複数の部分に別れて出てくることはない．ゆえに，下側エンベロープは放物線の名前の系列で表現することが可能である．図 3.4 のような下側エンベロープが求めるべき下方無限遠から見える部分である．



(a) 放物線の集合　　(b) 放物線の下側エンベロープ

図 3.4: 放物線の集合に対する下側エンベロープ

3.2 基本アルゴリズム

各行列要素 (i,j) から距離 k 以内の正方形領域における行列要素の最大値 $M_k(i,j)$ を求めるサブルーチンを用いて，各要素に対して最も近い優越要素を探索することを考える．具体的には (i,j) を中心として，左右及び上下に k 要素の帯状領域における行列要素の最大値を $H_k(i,j)$,$V_k(i,j)$ として求め，$H_k(i,j)$,$V_k(i,j)$ を用いて上記の正方形領域における最大値 $M_k(i,j)$ を計算する．$H_k(i,j)$,$V_k(i,j)$ 及び $M_k(i,j)$ は以下のようになる (図 3.5)．

$$H_k(i,j) = \max\{A(i,j') \mid |j-j'| \leq k\},$$

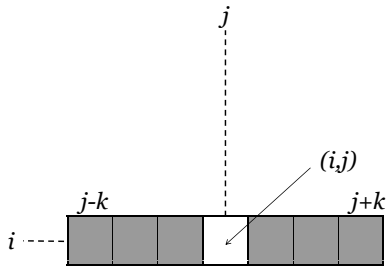
$$V_k(i,j) = \max\{A(i',j) \mid |i-i'| \leq k\},$$

$$M_k(i,j) = \max\{H_k(i-k,j), H_k(i+k,j), V_k(i,j-k), V_k(i,j+k)\}.$$

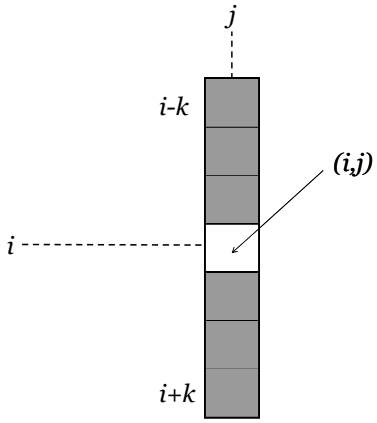
このとき，距離行列は次のようにして求めることができる．

$$D(i,j) = \min\{k \mid M_k(i,j) > A(i,j)\}.$$

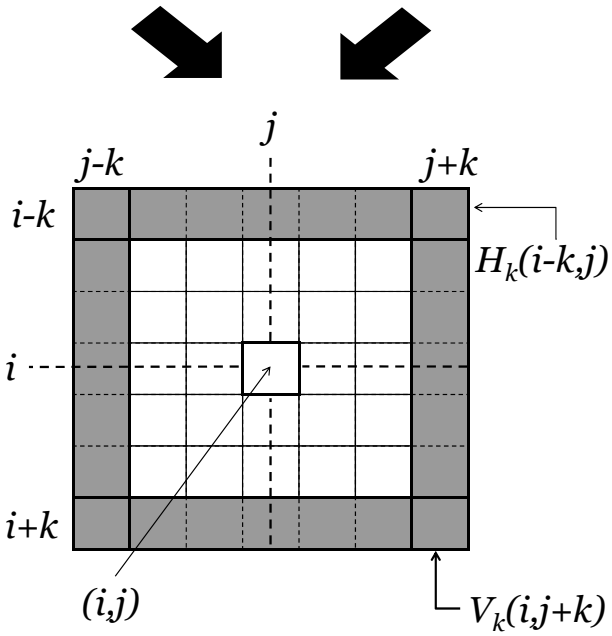
最初に $A(i,j)$ の値を超えるのに必要な正方形領域の大きさを考えることによって距離を定める．すなわち，$A(i,j)$ の値を最初に超えるのに必要な正方形領域の大きさとして最も近い優越要素までの距離を求めることができる．まず，$H_k(i,j)$,$V_k(i,j)$ にそれぞれの領域の最大値を記憶し，H_k と V_k を用いて帯領域の最大値を計算し，$M_k(i,j)$ を求める．$M_k(i,j)$ と $A(i,j)$ を比較して，$M_k(i,j)$ の方が大きい値だったら優越要素が距離 k にある要素の中に存在するので，$D(i,j)$ に k を書き込む．そうでなかったら，距離 k を変更して，この操作を優越要素が見つかるまで繰り返す．以上のアルゴリズムを Algorithm 1 に示す．



(a) この領域における行列要素の最大値が $H_k(i,j)$.



(b) この領域における行列要素の最大値が $V_k(i,j)$.



(c) (i,j) 要素から L_∞ 距離が k に等しい要素から成る領域. この領域における行列要素の最大値が $M_k(i,j)$.

図 3.5: $H_k(i,j)$,$V_k(i,j)$ 及び $M_k(i,j)$ の領域

Algorithm 1 基本アルゴリズム

入力: $n \times n$ 実数値行列 A

出力: 距離行列 D

```
for each (i, j) ∈ A do {
    H_0(i, j) = V_0(i, j) = M_0(i, j) = A(i, j)
    D(i, j) = ∞
}
for k = 1 to n do {
    for each (i, j) ∈ A do {
        H_k(i, j) = max{H_{k-1}(i, j), A(i, j − k), A(i, j + k)}
        V_k(i, j) = max{V_{k-1}(i, j), A(i − k, j), A(i + k, j)}
        M_k(i, j) = max{H_k(i − k, j), H_k(i + k, j), V_k(i, j − k), V_k(i, j + k)}
        if M_k(i, j) > A(i, j) and D(i, j) = ∞ {
            D(i, j) = k
        }
    }
}
```

補題 1 Algorithm 1 は $O(n^3)$ 時間で，各要素に対して最も近い優越要素までの距離を求める．また，必要な作業領域は $O(n^2)$ である．

証明： 基本アルゴリズムにおいて，H_k, V_k, M_k は各要素 (i, j) に対して，帯状領域および正方形領域を任意の距離 k に対して計算している．また，$k = 1$ から始めており，優越要素が見つかったらすぐに終了しているため，求められた要素より距離が近い優越要素は存在しない．また，$k = 1$ から $k = n$ までのすべての H_k, V_k, M_k を記憶する必要はなく，実際に H_k, V_k, M_k を求めるのに必要なのは，$H_{k-1}, V_{k-1}, M_{k-1}$ だけである．なぜならば，H_k は，図 3.6 のように H_{k-1} の値から計算することができ，$k-1$ 未満の H は必要がない．同様にして V_k を求めるのに必要なのは V_{k-1} の値だけである．

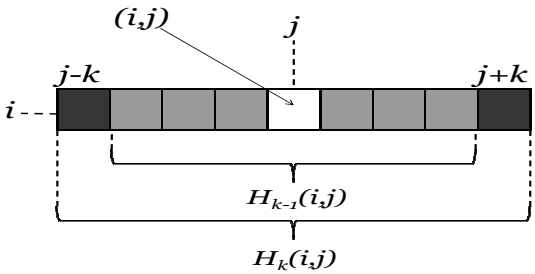


図 3.6: H_{k-1} による H_k の計算

したがって，必要な作業領域は $O(n^2)$ で十分である．■

3.3 提案するアルゴリズム

次に，基本アルゴリズムを応用して，アルゴリズムの時間計算量を $\mathrm{O}(n^3)$ から $\mathrm{O}(n^2\sqrt{n})$ に改善する方法について説明する．このアルゴリズムは，まず第 1 フェーズで各要素に対して距離が $\lceil\sqrt{n}\rceil$ 以内の近傍に優越要素があるかどうかを調べる．次に第 2 フェーズで近傍に優越要素がない要素に対して，優越要素を探索する．

3.3.1 近傍の探索

第 1 フェーズでは Algorithm 1 を利用して，各行列要素に対して優越要素を含むような正方形領域を考える．Algorithm 1 との違いとして，$k = 1$ から $k = n$ まで繰り返すのではなく，$k = 1$ から $k = \lceil\sqrt{n}\rceil$ まで繰り返して終了することとする．このとき，第 1 フェーズで優越要素が見つからない要素が存在する．そのような要素 (i, j) については (i, j) を中心として，$(2\lceil\sqrt{n}\rceil + 1) \times (2\lceil\sqrt{n}\rceil + 1)$ の正方形領域を R とすると，R 内に $A(i, j)$ より大きい要素が存在しなかったことが分かる．以下では，このように第 1 フェーズで優越要素が見つからなかった要素のことを局所最大要素と呼ぶことにする．第一フェーズの流れを図 3.7 に示す．ただし，優越要素 (i', j') は $k = 1$ から始めて，最初に見つかる優越要素であるとする．

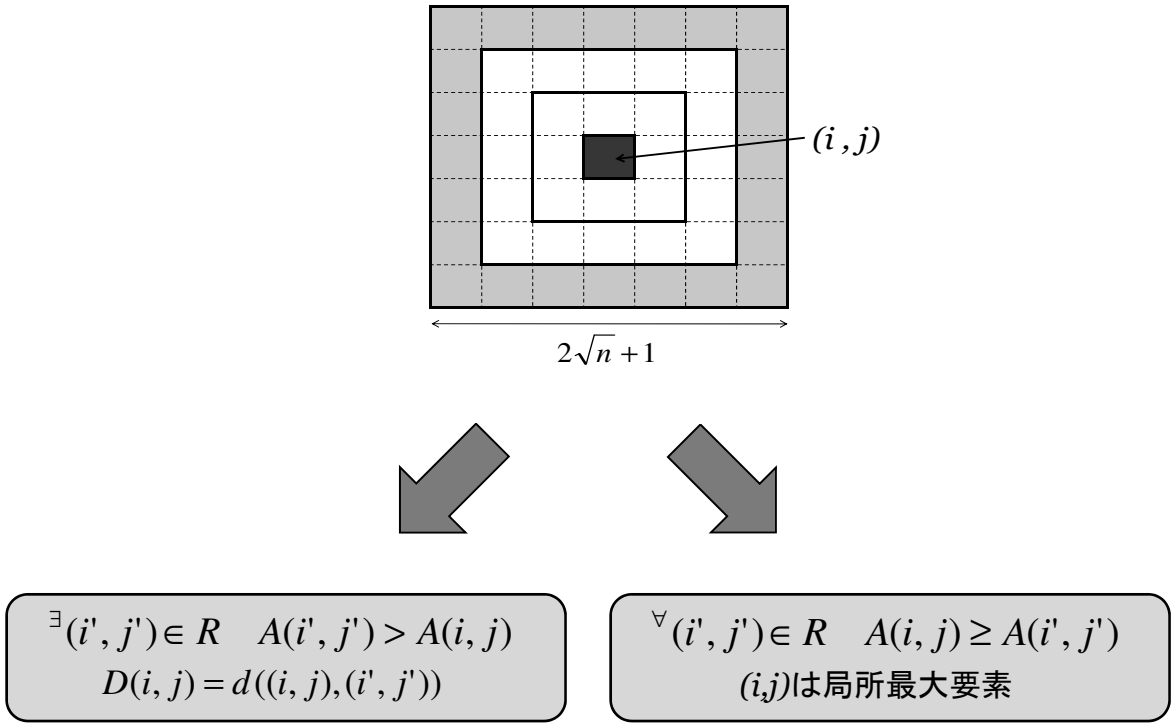


図 3.7: (i, j) の近傍の探索

3.3.2 局所最大要素に対する探索

第 2 フェーズでは，局所最大要素に対して優越要素を求める．まず，入力の $n \times n$ 行列 A を，$\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ の小領域（バケット）に分割する．各バケット内で行列 A における最大値を求め, そのバケットの値とすることで，新しい行列 B が次のように定義できる．

$$B(i_B, j_B) = \max\{A(i,j) \mid (i_B-1)\lceil\sqrt{n}\rceil \le i < i_B\lceil\sqrt{n}\rceil, (j_B-1)\lceil\sqrt{n}\rceil \le j < j_B\lceil\sqrt{n}\rceil\}.$$

このとき，局所最大要素は第一フェーズにおいて優越要素が見つかっていないことから，必ず各々のバケットの最大値となる．この $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ 行列 B に Algorithm 1 を適用する．$k = 1$ から $k = \lceil\sqrt{n}\rceil$ まで適用して，正確に値を求める．(i_B, j_B) のバケットに対して計算された距離が K であるとする．これは (i_B, j_B) から距離が K 未満のバケット内には $B(i_B, j_B)$ より大きな値を持つ要素は存在しないが，図 3.8 のように距離が K のバケット内に $B(i_B, j_B)$ より大きな値を持つ要素が存在することを示している．

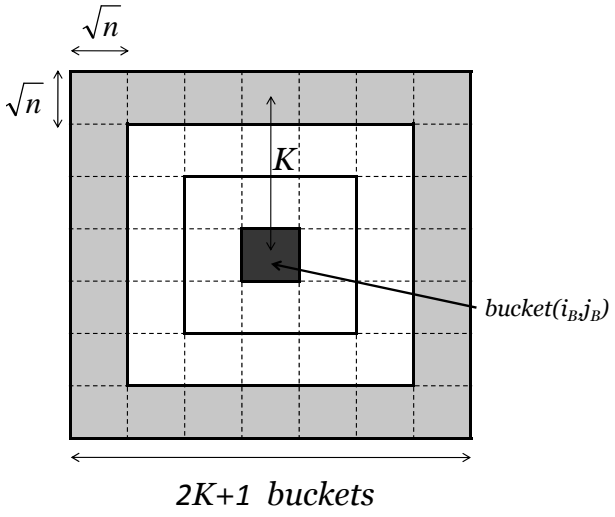


図 3.8: バケット (i_B, j_B) から距離 K にあるバケットの集合

次に，バケット (i_B, j_B) に含まれる局所最大要素に対して，最も近い優越要素までの距離を正確に求めることを考える．バケット (i_B, j_B) の最大値を与える要素と (i_B, j_B) から距離 K にあるバケット内の要素を比較する．バケットの幅が $\lceil\sqrt{n}\rceil$ であるため各バケットの要素数は $\mathrm{O}(n)$ であり，距離 K は最大で $\lceil\sqrt{n}\rceil$ になりうるので，(i_B, j_B) から距離 K にある帯領域に含まれるバケット内の要素数は $\mathrm{O}(n\sqrt{n})$ である．また，各バケット内にある最大値を与える要素は必ずしも一つであるとは限らず，最大で $\mathrm{O}(n)$ 個存在する．このことから素朴な方法で比較しようとすると 1 つのバケットあたり $\mathrm{O}(n^2\sqrt{n})$ 時間かかってしまう．そのため以下のような操作を行い，局所最大要素に対して優越要素を探索する．

まず，バケット (i_B, j_B) から距離 K にある帯領域を 3 種類の領域に分割する．バケット (i_B, j_B) の各要素 (i, j) と帯領域に含まれる任意の要素 (i', j') に対して，次のように 3 つの領域 R_1, R_2, R_3 を定める．

領域 R_1**.** L_∞ 距離が常に $|i-i'|$ で得られる要素の集合 (図 3.9).

$$^\forall(i,j) \in (i_B, j_B), ^\forall(i',j') \in R_1,\ |i-i'| \geq |j-j'|.$$

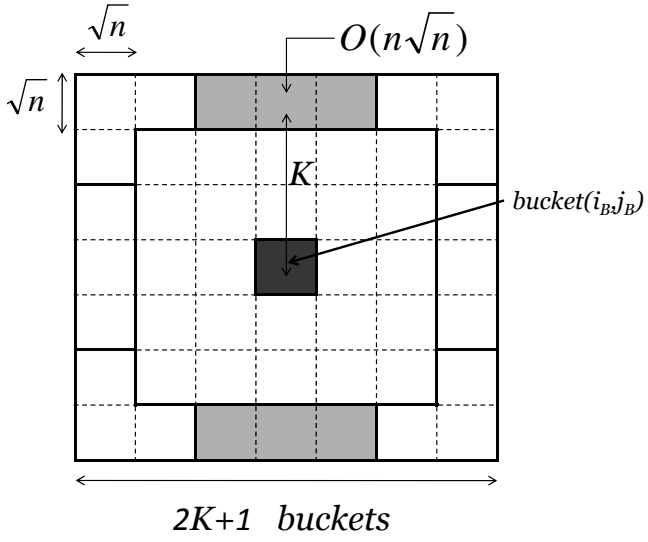


図 3.9: (i_B, j_B) に対する領域 R_1

領域 R_2**.** L_∞ 距離が常に $|j-j'|$ で得られる要素の集合 (図 3.10).

$$^\forall(i,j) \in (i_B, j_B), ^\forall(i',j') \in R_2,\ |j-j'| \geq |i-i'|.$$

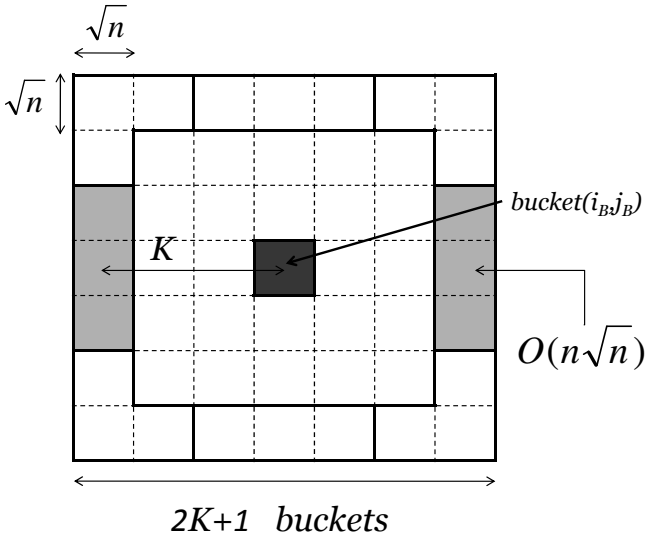


図 3.10: (i_B, j_B) に対する領域 R_2

領域 R_3**.** 要素ごとに L_∞ 距離が $|i-i'|$ もしくは $|j-j'|$ で変化する要素の集合 (図 3.11).

$$^\forall(i,j) \in (i_B, j_B), ^\forall (i',j') \in R_3,\ |i-i'| \geq |j-j'| \vee |j-j'| \geq |i-i'|.$$

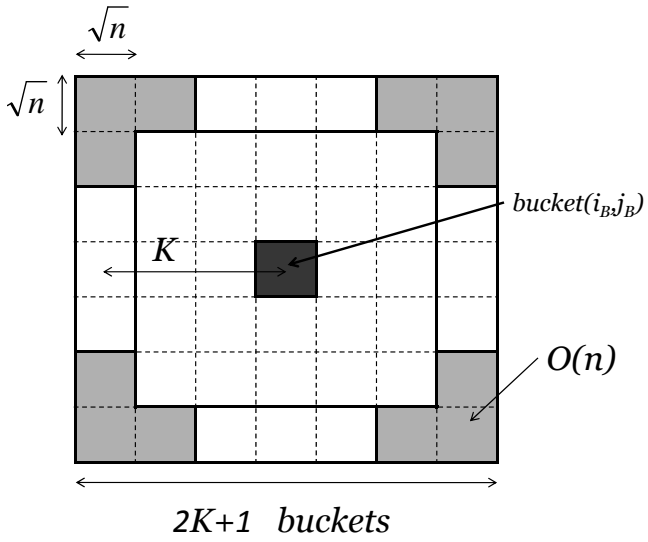


図 3.11: (i_B, j_B) に対する領域 R_3

バケットの幅は $\lceil\sqrt{n}\rceil$ であるので，領域 R_1,R_2 に含まれる要素数は $\mathrm{O}(n\sqrt{n})$ であり，領域 R_3 に含まれる要素数は $\mathrm{O}(n)$ である．

(i_B, j_B) に含まれる任意の要素 (i,j) に対して，領域 R_1 に含まれる各要素 (i',j') については，$B(i_B, j_B)$ と 1 度だけ比較し，$B(i_B, j_B)$ より値の大きい要素の中で $|i-i'|$ が最も小さい要素が領域 R_1 において最近の優越要素となる．領域 R_2 も同様にして，$|j-j'|$ が最も小さい要素が領域 R_2 において最近の優越要素となる．領域 R_1,R_2 の要素数は $\mathrm{O}(n\sqrt{n})$ なので比較回数は $\mathrm{O}(n\sqrt{n})$ 回である．

領域 R_3 に含まれる各要素 (i',j') については，$|i-i'|$ と $|j-j'|$ のどちらが大きいのかは要素同士によって異なるため，すべての要素間の距離を計算することを考える．しかし，R_3 に含まれる要素数は $\mathrm{O}(n)$ で，バケット (i_B, j_B) 内の要素数も $\mathrm{O}(n)$ であるため，任意の要素間を調べると各バケットに対して $\mathrm{O}(n^2)$ 時間かかってしまう．そのため効率よく優越要素を見つけるのに以下の操作を行う．

まず，バケット (i_B, j_B) を要素ごとに調べるのではなく，行ごとに調べていく．双対変換の考え方を用いて領域 R_3 の中で $B(i_B, j_B)$ より大きい値を持つ要素を，バケット (i_B, j_B) 内の要素からの垂直距離が要素同士の距離となるような折れ線に変換する (図 3.12).

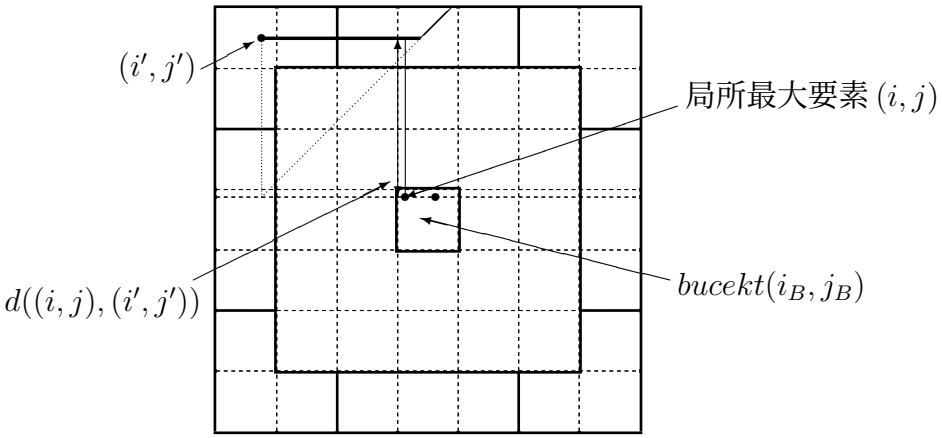


図 3.12: 要素と対応する折れ線

バケット (i_B, j_B) 内の i 行に対して，$i' \leq i$,$j' \leq j$ を満たす R_3 内の要素 (i', j') は (i', j') と $(i', j' + |i - i'|)$ 間の水平線分と $(i', j' + |i - i'|)$ から 45 °の半直線からなる図 3.13 のような折れ線に変換される．R_3 内の他の要素については回転させれば同じように変換することが可能である．このため，以下では一般性を失うことなく R_3 に含まれる要素 (i', j') が $i' < i$ かつ $j' < j$ であることを仮定する．

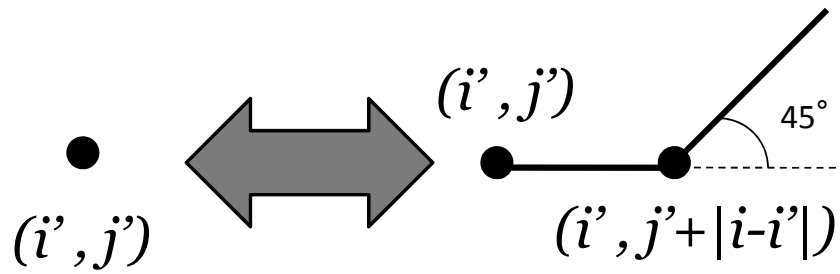
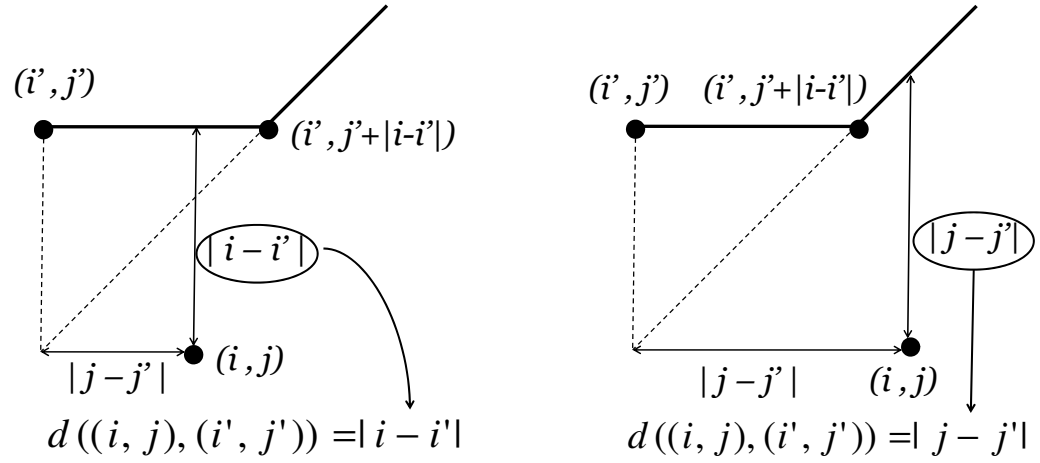


図 3.13: i 行に対して (i', j') から変換される折れ線

このとき，次の補題が成り立つ．

補題 2 バケット (i_B, j_B) 内の要素から鉛直線を伸ばしたとき，最初に交差する折れ線と対応する優越要素が最近の優越要素となる．

証明： 領域 R_3 内の各優越要素 (i', j') を折れ線に変換したとき，この折れ線は図 3.14 のように，$|i - i'| > |j - j'|$ ならば (i, j) からの鉛直線は折れ線の水平部分と交差し，その時の折れ線と (i, j) との垂直距離は $|i - i'|$ である．また，$|i - i'| < |j - j'|$ ならば，半直線の部分と交差し，その時の折れ線と (i, j) との垂直距離は $|j - j'|$ である．すなわち，いずれの場合も折れ線と (i, j) の垂直距離は (i', j') と (i, j) の距離に等しくなる．



(a) $|i - i'| > |j - j'|$ の場合　　(b) $|i - i'| < |j - j'|$ の場合

図 3.14: 要素と折れ線の距離

ゆえに，(i', j') と (i, j) の距離は必ず対応する折れ線と (i, j) との垂直距離と等しくなっているため，最近の優越要素 (i', j') を求めるには (i, j) との垂直距離が一番小さい折れ線を求めればよい．したがって，垂直距離が一番小さい折れ線は，(i, j) から鉛直線を伸ばしたとき最初に交差する折れ線であるので，そのような折れ線と対応する領域 R_3 内の優越要素が最近の優越要素となる．■

最初に交差する折れ線を求めることは折れ線に対する隠線除去問題を解くことにあたる．隠線除去問題を解くことにより，図 3.15 のように折れ線の下側エンベロープだけを求める．

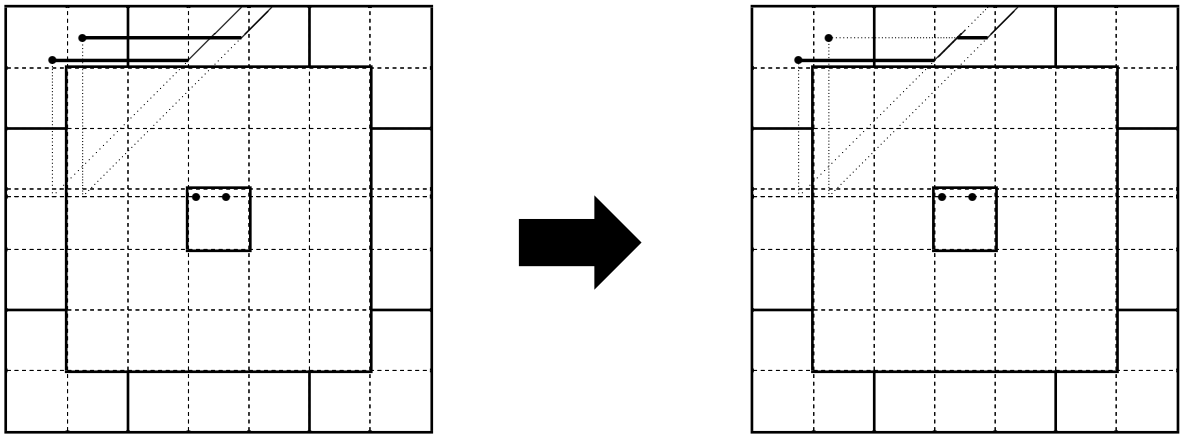


図 3.15: 折れ線の下側エンベロープ

領域 R_3 の要素 (i', j') について下側エンベロープを求めるとき，i' 行にある要素に対応する折れ線で下側エンベロープに出るのは，j' の値が最も大きい要素だけである．したがって，各行における優越要素の中で，j' の値が最も大きい要素についてだけ折れ線を考えればよい．R_3 の一番下の行を j' の降順に走査する．このとき，優越要素が見つかったら折れ線を作り，行を上に移動する．また，折れ線を作ったとき，下の行のエンベロープと交差したら図 3.16 のように，新しくエンベロープを更新する．このようにして下側エンベロープを構成する．

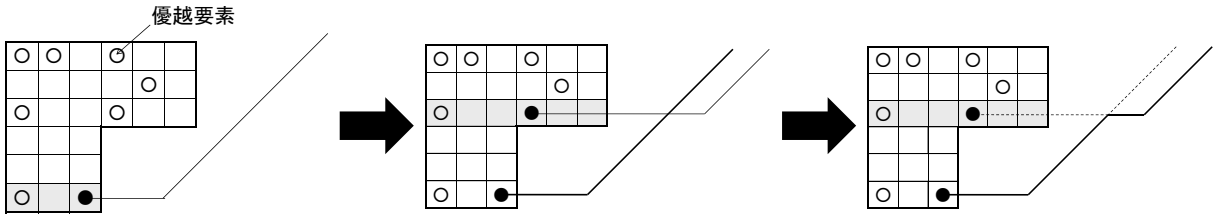
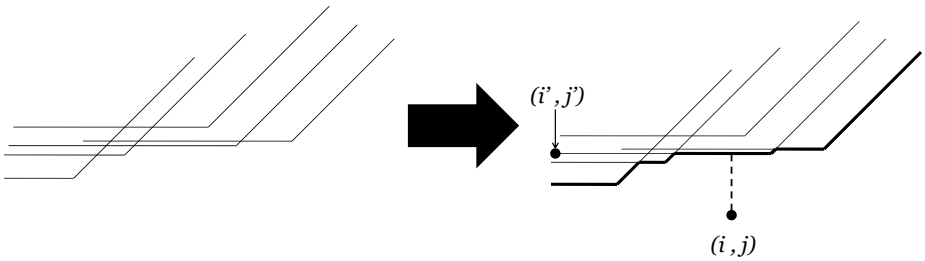


図 3.16: 下側エンベロープの構成

また，放物線と同様にすべての折れ線は同じ形をしているため下側エンベロープを求めたとき，折れ線の名前の系列で完全に表現できる．補題より各列において下側エンベロープを構成する折れ線と対応する要素が，最近の優越要素となる．図 3.17 では (i, j) に対して，(i', j') が最近の優越要素となる．



(a) 折れ線の集合　　(b) 折れ線に対する下側エンベロープ

図 3.17: エンベロープと最近の優越要素との関係

領域 R_3 に含まれる要素の数は $\mathrm{O}(n)$ なので，バケットの各行に対して，下側エンベロープは $\mathrm{O}(n)$ 時間で求められる．また，バケットの行数は $\sqrt{n}$ なので，各バケットに対して下側エンベロープを構成するのに，$\mathrm{O}(n\sqrt{n})$ 時間かかる．しかしながら，前の行のエンベロープを利用することによって計算することで，次の補題が成り立つ．

補題 3 下側エンベロープは各バケットに対して $O(n)$ 時間で求められる．

証明： 次の行に必要なエンベロープは図 3.18 のように水平方向にずらすだけでよい．なぜならば，行を移動したときバケット (i_B, j_B) 内の要素 (i, j) と領域 R_3 内の優越要素

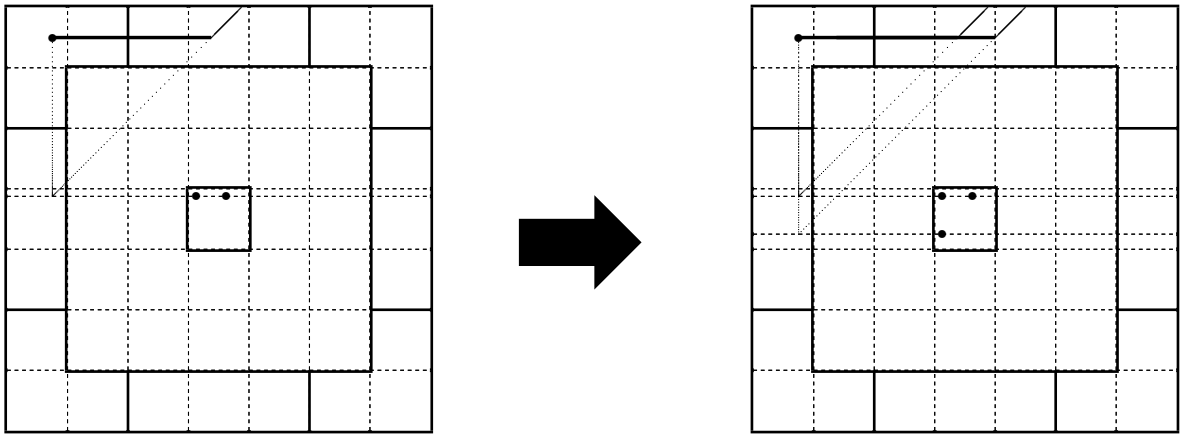
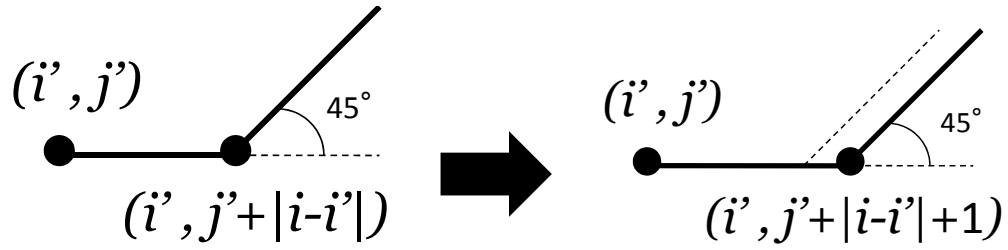


図 3.18: エンベロープの変更

(i', j') に対して，要素間の距離は垂直距離のみが変化するためである．つまり,(i_B, j_B) 内の i 行と比較していたのが $i+1$ 行に変わっただけである．このとき図 3.19 に示すように，i 行と比較したときの (i', j') と対応する折れ線は，(i', j') と $(i', j' + |i - i'|)$ 間の水平線分と $(i', j' + |i - i'|)$ から 45 °の半直線からなる．また，$i+1$ 行と比較したときは (i', j') と $(i', j' + |i - i'| + 1)$ 間の水平線分と $(i', j' + |i - i'| + 1)$ から 45 °の半直線からなる折れ線に変換される．



(a) i 行に対する折れ線　　(b) $i+1$ 行に対する折れ線

図 3.19: 行の変更による折れ線の変更

このことから行が変わったとき，折れ線の曲がる点が水平方向に移動するだけであることがわかる．各行の要素に対して，水平方向にずらした位置から鉛直線を伸ばすことで，任意の行に対して同じエンベロープで考えることができるため，各バケットに対してエンベロープを計算するのは最初の行に対してだけでよい．したがって，$\mathrm{O}(n)$ 時間でエンベロープを計算することができる．■

バケット (i_B, j_B) に対して下側エンベロープを求め，バケット内の各要素から鉛直線を引くことで最近の優越要素を調べる．このようにして，領域 R_3 に含まれる優越要素を調べる．各領域ごとの最近な優越要素の候補を比較することで，局所最大要素に対しての優越要素を求める．

基本アルゴリズムを利用した近傍の探索アルゴリズムを Algorithm 2 に，アルゴリズム全体を Algorithm 3 に示す．

Algorithm 2 近傍の探索アルゴリズム

BasicProcedure(N, M, A, D)

入力：$N \times N$ の実数値行列 A，近傍の範囲 M

出力：距離行列 D

初期化:

```
for (i, j) ∈ A do {
    H_0(i, j) = V_0(i, j) = M_0(i, j) = A(i, j)
    D(i, j) = ∞
}
```

近傍の探索:

```
for k = 1 to M do {
    for (i, j) ∈ A do {
        H_k(i, j), V_k(i, j), M_k(i, j) を計算.
        if M_k(i, j) > A(i, j) and D(i, j) = ∞ {
            D(i, j) = k
        }
    }
}
```

Algorithm 3 $\mathrm{O}(n^2\sqrt{n})$ 時間のアルゴリズム

(第 1 フェーズ)

入力：$n \times n$ 実数値行列 A

BasicProcedure$(n, \lceil\sqrt{n}\rceil, A, D)$ を計算する.

(第 2 フェーズ)

行列 B の定義:

for $i_B = 1$to $\lceil\sqrt{n}\rceil$ **do** {
  for $j_B = 1$to $\lceil\sqrt{n}\rceil$ **do** {
    $B(i_B, j_B) = \max\{A(i,j) | (i_B - 1)\lceil\sqrt{n}\rceil \leq i < i_B\lceil\sqrt{n}\rceil, (j_B - 1)\lceil\sqrt{n}\rceil \leq j < j_B\lceil\sqrt{n}\rceil\}$
  }
}

Basic Procedure$(\lceil\sqrt{n}\rceil, \lceil\sqrt{n}\rceil, B, D)$ を計算.

優越要素の探索:

for $(i_B, j_B) \in B$ **do** {
  R_3 内の優越要素に対して下側エンベロープを形成.
  for $(i,j) \in (i_B, j_B)$ **do** {
    if $D(i,j) = \infty$ {
      R_1, R_2, R_3 における最近の優越要素を探索.
      $D(i,j) =$ 最近の優越要素との距離
    }
  }
}

定理 1 Algorithm 3 は $O(n^2\sqrt{n})$ 時間で，各要素に対して最も近い優越要素までの距離を求める．また，必要な作業領域は $O(n^2)$ である．

証明： まず，第 1 フェーズについて示す．第 1 フェーズは，基本アルゴリズムを距離が $\lceil\sqrt{n}\rceil$ 以内の範囲で実行しただけなので，$O(n^2\sqrt{n})$ 時間で計算することが可能である．

次に，第 2 フェーズについて示す．まず，各バケットは $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ なので，バケットの総数は $O(n)$ 個である．領域 R_1, R_2 について，各バケット (i_B, j_B) に対して領域 R_1, R_2 に含まれる $O(n\sqrt{n})$ 個の要素を 1 度だけ調べるため，全体で $O(n^2\sqrt{n})$ 回調べる．また，領域 R_3 について，補題 3 より，各バケット (i_B, j_B) に対して, 下側エンベロープは $O(n)$ 時間で求められる．また，バケットの各行に対して優越要素を計算するのに $O(\sqrt{n})$ 回調べる必要がある．各バケットには $\lceil\sqrt{n}\rceil$ 行あり，バケットは全部で $O(n)$ 個あることから，$O(n^2)$ 時間で領域 R_3 における最近の優越要素の候補を求めることができる．ただし，バケット (i_B, j_B) に対して，より大きい値を持つバケットの距離が K であったとき，そのバケットの集合に対してこの探索を行うが，バケット間の距離が近いからといって必ずしも，そのバケットに含まれる要素間の距離が近いわけではない．図 3.20 のように (i, j) はバケット (i_B, j_B) の要素であり，(i'', j'') は (i_B, j_B) から距離 K のバケットに含まれているが，距離 $K+1$ のバケットに含まれている (i', j') の方が距離が近くなっている．

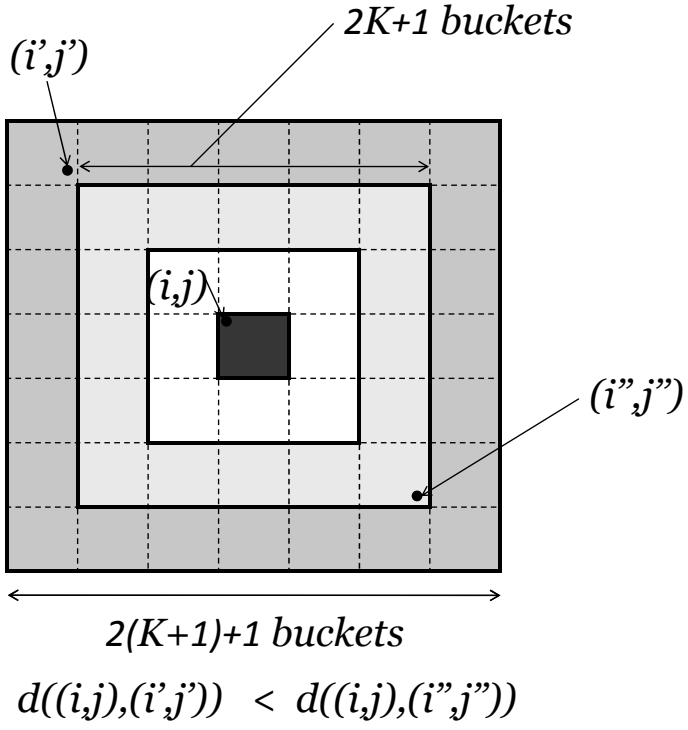


図 3.20: 距離 K のバケットに最近の優越要素が存在しない例

そのため，バケット (i_B, j_B) から距離 K にあるバケット及び距離 $K+1$ にあるバケットに含まれる要素を調べる．また，このとき優越要素を探すのにかかる時間は距離 K のバ

ケットだけを調べるのと比較しても定数倍しかかからない．したがって，領域 R_1, R_2, R_3 における最近な優越要素の候補を求めるのに必要な計算時間は $\mathrm{O}(n^2\sqrt{n})$ 時間である．また，各要素に対して，それぞれの領域における最近の優越要素の候補を比較するが，見つかる候補は各要素ごとに定数個であるため，$\mathrm{O}(n^2)$ 時間で最近の優越要素を求めることができる．ゆえに，各要素に対して，$\mathrm{O}(n^2\sqrt{n})$ 時間で最近の優越要素までの距離を求めることができる．

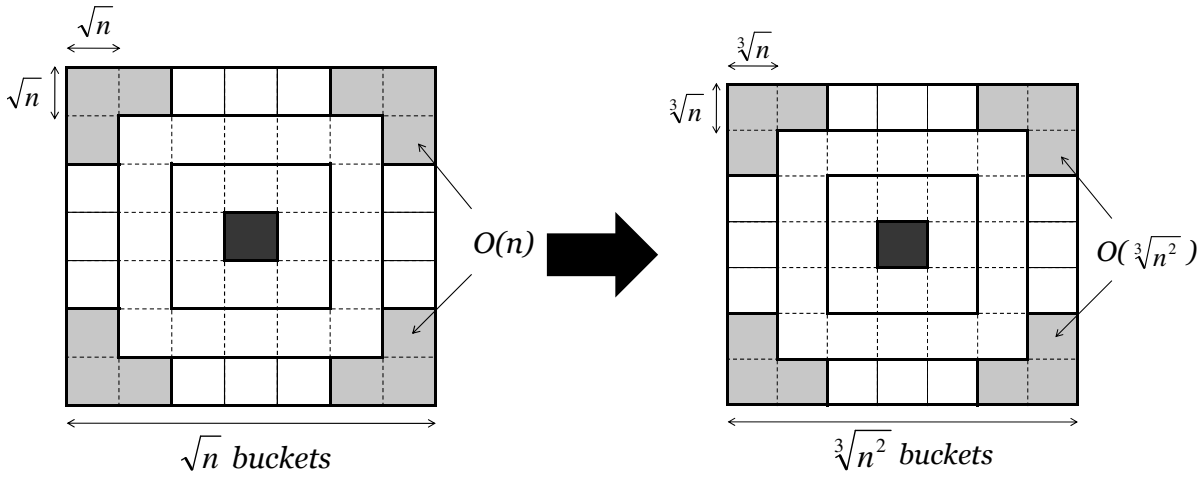
作業領域について，第 1 フェーズについては Algorithm 1 と変更がないため $\mathrm{O}(n^2)$ 必要である．第 2 フェーズについては対象のバケットの要素が局所最大要素であることを記憶するのに $\mathrm{O}(n)$，領域 R_3 に含まれる要素についてエンベロープを記憶するのに $\mathrm{O}(n)$ 必要である．したがって必要な作業領域は $\mathrm{O}(n^2)$ である．■

第4章　アルゴリズムの改善

Algorithm 3 では，行列を $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ のバケットに分割したが，その分割が最適な分割であるとは限らない．分割のサイズとアルゴリズムを変更することで，より少ない時間計算量で優越要素を求めることができる．

4.1　分割サイズの変更

Algorithm 3 では，行列を $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ のバケットに分割したが，バケットのサイズを $\lceil\sqrt[3]{n}\rceil \times \lceil\sqrt[3]{n}\rceil$ に変更する．これより，各バケットに含まれる要素数は $\mathrm{O}(\sqrt[3]{n^2})$ となり，バケットの総数は $\mathrm{O}(\sqrt[3]{n^4})$ となる．変更前後の分割を図 4.1 に示す．

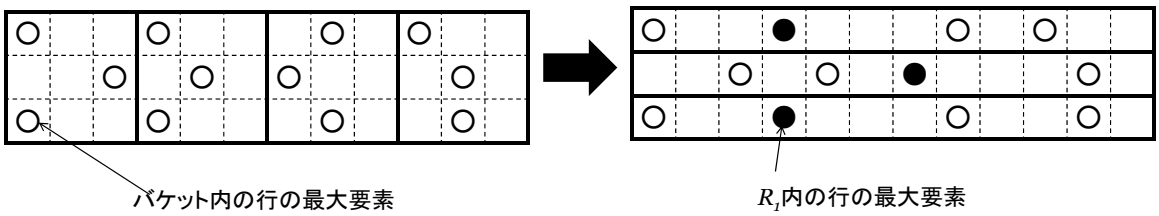


(a) $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ に分割　　(c) 分割変更後のバケット

図 4.1: 分割の変更

4.2 探索の効率化

バケットのサイズを変更して Algorithm 3 と同じように計算すると，第 1 フェーズを $O(n^2\sqrt[3]{n})$ 時間で計算できる．また，領域 R_3 における最近の優越要素を $O(n^2)$ 時間で求めることができる．しかしながら，領域 R_1,R_2 に含まれる要素に対して，すべての要素を調べているため，バケットのサイズを変更した後に同じように計算するとバケットごとに $O(n\sqrt[3]{n})$ 時間かかってしまう．そのため，各バケットについて行ごと及び列ごとの最大値をはじめに記憶することにより，その最大値と比較することで，計算を速くすることを考える．



(a) 各バケットの行の最大値　　(b) R_1 における行の最大値の計算

図 4.2: 探索の効率化

バケットにおける行ごとの最大値を用いて 図 4.2 のようにして，R_1 における行の最大値を計算する．各行の最大値と局所最大要素の値を比較して，優越要素の存在する行が見つかったら，その行の各要素と比較して優越要素を探索する．また，R_2 の列についても同様に探索する．このようにして，各バケットに対して，$O(n)$ 時間で領域 R_1, R_2 における最近の優越要素を探索することができる．バケットの個数が $O(\sqrt[3]{n^4})$ であることから，$O(n^2\sqrt[3]{n})$ 時間で領域 R_1, R_2 の優越要素を探索することができる．変更後のアルゴリズムは Algorithm 4 のように記述できる．

Algorithm 4 $O(n^2\sqrt[3]{n})$ 時間のアルゴリズム

(第 1 フェーズ)

入力：$n \times n$ 実数値行列 A

BasicProcedure$(n, \lceil\sqrt[3]{n}\rceil, A, D)$ を計算する.

(第 2 フェーズ)

行列 B の定義:

for $i_B = 1$to $\lceil\sqrt[3]{n}\rceil$ **do** {
  for $j_B = 1$to $\lceil\sqrt[3]{n}\rceil$ **do** {
    $B(i_B, j_B) = \max\{A(i,j) | (i_B-1)\lceil\sqrt[3]{n}\rceil \leq i < i_B\lceil\sqrt[3]{n}\rceil, (j_B-1)\lceil\sqrt[3]{n}\rceil \leq j < j_B\lceil\sqrt[3]{n}\rceil\}$
  }
}

各バケットの行・列ごとの最大値を計算.

Basic Procedure$(\lceil\sqrt[3]{n}\rceil, \lceil\sqrt[3]{n}\rceil^2, B, D)$ を計算.

優越要素の探索:

for $(i_B, j_B) \in B$ **do** {
  R_3 内の優越要素に対して下側エンベロープを形成.
  領域 R_1 の各行の最大値を計算.
  領域 R_2 の各列の最大値を計算.
  for $(i,j) \in (i_B, j_B)$ **do** {
    if $D(i,j) = \infty$ {
      R_1, R_2, R_3 における最近の優越要素を探索.
      $D(i,j) =$ 最近の優越要素との距離
    }
  }
}

定理 2 Algorithm 4 は $O(n^2\sqrt[3]{n})$ 時間で，各要素に対して最も近い優越要素までの距離を求める．また，必要な作業領域は $O(n^2)$ である．

証明： 第 1 フェーズは基本アルゴリズムを，距離が $\lceil\sqrt[3]{n}\rceil$ までと変更されただけなので，$O(n^2\sqrt[3]{n})$ 時間で，計算することが可能である．

次に，第 2 フェーズについて示す．まず，各バケットは $\lceil\sqrt[3]{n}\rceil \times \lceil\sqrt[3]{n}\rceil$ なので，バケットの総数は $O(\sqrt[3]{n^4})$ 個である．

領域 R_1 について，領域 R_1 に含まれるバケットは $O(\sqrt[3]{n^2})$ 個あり，バケットの行ごとの最大値を用いて R_1 の各行の最大値を $O(n)$ 時間で求めることができる．次に，各バケット (i_B, j_B) に対して領域 R_1 の各行の最大値と比較する．$B(i_B, j_B)$ より大きい値を持つ要素が見つかった場合その行を調べ，優越要素を見つける．このとき，領域 R_1 の各行の最大値と比較するのに最大で $O(\sqrt[3]{n})$ 時間かかり，R_1 の行により大きい値を持つ要素が見つかったとき，行を調べるのに $O(\sqrt[3]{n})$ 回比較する．ゆえに，各バケットに対して，領域 R_1 において最も近い優越要素を $O(n)$ 時間で求めることができる．したがって，バケットの数が $O(\sqrt[3]{n^4})$ であることから，領域 R_1 において，最も近い優越要素を見つけるのにかかる時間は $O(n^2\sqrt[3]{n})$ 時間である．領域 R_2 についても同様にして，列ごとの最大値を用いることにより，$O(n^2\sqrt[3]{n})$ 時間で領域 R_2 において最も近い優越要素を求めることができる．

領域 R_3 について，各バケット (i_B, j_B) に対して領域 R_3 に含まれる要素は $O(\sqrt[3]{n^2})$ 個あるため，定理 1 と同様にして，下側エンベロープを計算する時間は $O(\sqrt[3]{n^2})$ 時間かかる．下側エンベロープは行ごとに求める必要があるが，補題 3 と同じように考えることにより，各行のエンベロープは $O(\sqrt[3]{n^2})$ 時間で求めることができる．また，バケットの各行に対して優越要素を計算するのに $O(\sqrt[3]{n})$ 回調べる必要がある．バケットには $\lceil\sqrt[3]{n}\rceil$ 行あるので，1 つのバケットあたりにかかる計算時間は $O(\sqrt[3]{n^2})$ 時間かかる．バケットは全部で $O(\sqrt[3]{n^4})$ 個あることから，$O(n^2)$ 時間で領域 R_3 における優越要素を求めることができる．したがって，領域 R_1, R_2, R_3 における最近な優越要素の候補を求めるのに必要な計算時間は $O(n^2\sqrt[3]{n})$ 時間である．また，各要素に対して，それぞれの領域における最近の優越要素の候補を比較するが，見つかる候補は各要素ごとに定数個であるため，$O(n^2)$ 時間で最近の優越要素を求めることができる．ゆえに，各要素に対して，$O(n^2\sqrt[3]{n})$ 時間で最近の優越要素までの距離を求めることができる．

作業領域について，第 1 フェーズについては Algorithm 1 と変更がないため $O(n^2)$ 必要である．第 2 フェーズについては対象のバケットの要素が局所最大要素であることを記憶するのに $O(\sqrt[3]{n^2})$，領域 R_3 に含まれる要素についてエンベロープを記憶するのに $O(\sqrt[3]{n^2})$ 必要である．また，領域 R_1, R_2 における行及び列ごとの最大値を記憶するのに，$O(n\sqrt[3]{n^2})$ 必要である．したがって必要な作業領域は $O(n^2)$ である．■

第5章　おわりに

本論文では，距離変換の一般化として，与えられた $n \times n$ 実数値行列に対して，L_∞ 距離において最も近い優越要素までの距離を求める問題を考え，$\mathrm{O}(n^2 \sqrt[3]{n})$ 時間で解くアルゴリズムを提案した．

今後の課題として，より少ない時間計算量で一般化距離変換を解くアルゴリズムを提案することがあげられる．作業領域についても，提案したアルゴリズムでは $\mathrm{O}(n^2)$ の作業領域を必要とするが，時間計算量を増やすことなく，より少ない作業領域で計算できるように改善することが考えられる．また，本論文では L_∞ 距離において最も近い優越要素までの距離を求めているが，マンハッタン距離，ユークリッド距離に対して最も近い優越要素までの距離を求めることが考えられる．

謝辞

本研究を行うにあたり，日頃より懇切丁寧な指導を賜りました浅野哲夫教授に心より感謝いたします．また，上原隆平准教授，清見礼助教，金沢高専の元木光雄准教授には，適切な御教示を頂き，厚く御礼申し上げます．浅野研究室，上原研究室の学生の皆様にも公私にわたり，様々な場面でお世話になりました．この場を借りて感謝いたします．

参考文献

[1] D. W. Paglieroni, Distance Transforms, *Computer Vision, Graphics and Image Processing: Graphical Models and Image Processing*, Vol. 54, pp. 56-74, 1992.

[2] 平野靖, 鳥脇純一郎, 距離変換を用いた図形の構造解析手段, *Medical Imaging Technology*, Vol. 20, No. 1, pp. 13-22, 2002.

[3] H. Breu, J. Gil, D. Kirkpatrick, and M. Werman, Linear Time Euclidean Distance Algorithms, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 17, No. 5, pp. 529-533, 1995.

[4] T. Hirata, A unified linear-time algorithm for computing distance maps, *Information Processing Letters*, Vol. 58, No. 3, pp. 129-133, 1996.

[5] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, 2nd edition, Springer Verlag, 2000.