アルゴリズム 41－4
（1994．9．21）

ユークリッド距離変換アルゴリズム

平田富夫　加藤敏洋

名古屋大学工学部

距離変換とは、入力として与えられた2値画像の各画素についてそこから最も近い0画素への距離を求める処理で、ディジタル画像処理における基本的な処理である。我々は先に$O(N^2)$時間で厳密なユークリッド距離変換を行なうアルゴリズムを与えた。ただし、画像のサイズを$N \times N$とする。本論文ではこのアルゴリズムが有効な距離のクラスを与える。このクラスは4近傍距離、8近傍距離、それに8角形距離やチャムファー距離など画像処理に現れるほとんどの距離を含む。

An Algorithm for Euclidean Distance Transformation

Tomio HIRATA and Toshihiro Kato

Faculty of Engineering, Nagoya University
Chikusaku, Nagoya, 464-01 Japan

Distance transforms of binary images contain, for each pixel, the distance between that pixel and the pixel of value 0 closest to it. They are useful for a variety of machine vision applications. In this paper we show that the algorithm for a Euclidean distance transform that we have previously proposed is also available for a wide class of distances. The class contains most distances that appear in machine vision applications such as cityblock, chessboad, octagonal and chamfer distances.

1 はじめに

2値画像の距離変換はディジタル画像処理における基本的な処理である。距離変換とは、入力として与えられた2値画像の各画素についてそこから最も近い0画素への距離を求める処理で、スケルトン画像と組み合わせて図形の形状を抽出する処理やパターンマッチング、最近接点補間、それにモフォロジカル変換などで用いられる [5, 10]。

2値画像の距離変換の定義は Rosenfeld[7] らによって与えられた。そこでは距離として4近傍または8近傍による最小経路の長さ、すなわち4近傍距離または8近傍距離を用いている。ユークリッド距離ではなく、4近傍距離や8近傍距離が用いられたのは、3×3 近傍に基づく局所処理のくり返しによって距離変換が容易に求まるからである。

4近傍距離や8近傍距離を用いた場合、1点からの距離が四角形状に広がるため、ユークリッド距離からのずれが大きい。そこで、4近傍と8近傍を交互に用いて八角形状に広げる8角形距離や、3×3 以上の近傍系を用いてユークリッド距離へのより精密な近似が図られてきた [2, 4]。しかし、いずれも近似値しか求まらない画素が生じる場合があり、その意味で厳密なユークリッド距離は与えない。また、近傍を広げたことにより処理が複雑化するなどの問題も起こってきた。

一方、各画素に最も近い0画素との距離値を求めるだけでなく、最も近い0画素の座標値も同時に伝搬することにより、厳密なユークリッド距離を8近傍による演算で算出する方法が山田 [11] により提案されている。この方法によれば N^2 台のプロセッサを用いて $O(N)$ 時間で厳密なユークリッド距離変換を行なう並列アルゴリズムが得られ、逐次アルゴリズムでは $O(N^3)$ 時間となる。ただし、画像のサイズを $N\times N$ とする。Ragnemalm[6] は山田の方法を等高線スキャンと組み合わせて $O(N^2)$ 時間のアルゴリズムを得ている。等高線スキャンによる方法では、各0画素から同時に波を発生させ、その伝搬を追跡することで距離変換を行なう。また、座標値の伝搬を行方向と列方向の処理に分解してハードウェア化と並列処理に適した形の実現法も提案されている [5]。しかしこれらはいずれも、入力画像と同じ大きさの作業配列(座標値配列)を必要とし、記憶容量の制約の厳しい場合には適さない。近年、斉藤, 鳥脇 [8, 9] らは、入力画像と同じ大きさの作業配列を使うことなく厳密なユークリッド距離変換を行なうアルゴリズムを与えた。このアルゴリズムは入力画像によって時間計算量が変化し、とくに斜め線の入った画像の場合には、その計算量は $O(N^3)$ となる。つまり、入力画像によっては、4近傍距離変換や8近傍距離変換の場合の計算量 $O(N^2)$ より効率が落ちる場合がある。

[3] では、上のような作業配列を必要とせず、しかもどのような入力画像に対しても $O(N^2)$ で厳密なユークリッド距離変換を行なうアルゴリズムを与えている。このアルゴリズムは、基本的には [5] や [8] のアルゴリズムと同様に、行方向と列方向の処理に分解して距離変換を行なうものである。これまでの距離変換アルゴリズムのほとんどは、近傍の画素の距離値を調べその最小値を用いて処理中の画素の距離値を求めており、0画素からの距離をスキャンとともに伝搬させていくというのが共通の手法であった。[3] で提案しているアルゴリズムは、列方向のスキャンは従来の方法と同じであるが、行方向のスキャンでは、N 個の関数の下側包絡線を求めることでその行の各画素に最も近い0画素をあらかじめ求めるという新しいアイデアを用いている。[6] のアルゴリズムにくらべ計算量は同じであるが、入力画像のスキャン方向が行方向と列方向の2種類のため単純でそのため並列計算に適している。すなわち、共有メモリをもつ N 台のプロセッサで $O(N)$ 時間で実行できる。また、3次元画像の距離変換にも拡張が容易で、その時間計算量は $O(N^3)$ である。

本論文では、まずこのアルゴリズムを紹介し、次にこのアルゴリズムが有効な距離のクラスを与える。このクラスはユークリッド距離の他に4近傍距離(L_1 距離)や8近傍距離(L_∞ 距離)、さらに8角形距離やチャムファー距離などのような画像処理で用いられるほとんどの距離を含んでいる。

以下、2章で本論文で必要となる定義を述べ、3章で [3] のユークリッド距離変換アルゴリズムを紹介する。4章では、このアルゴリズムが画像処理でよく用いられるその他の距離の場合にも(厳密な)距離変換を行なうことを示し、そのような距離のクラスを与える。5章はまとめである。

2 準備

サイズが N行 N列 (以下 $N \times N$と略す) の 2 次元ディジタル画像において、第 i 行第 j列の画素を (i,j) で表し、その濃度値が b_{ij} である画像を $\mathbf{B} = \{b_{ij}\}$ のように記述する。濃度値として 0 と 1 しかとらない画像を 2 値画像と呼び、値 0 および 1 の画素をそれぞれ 0 画素、1 画素と呼ぶ。

2 値画像 $\mathbf{B} = \{b_{ij}\}$ のユークリッド距離変換画像 $\mathbf{D} = \{d_{ij}\}$ を次式で定義する。

$$d_{ij} = \min_{1 \le p,q \le N} \{\sqrt{(i-p)^2 + (j-q)^2} \mid b_{pq} = 0\} \quad (1)$$

すなわち、$\mathbf{D}$ は d_{ij} が入力画像 $\mathbf{B}$ の画素 (i,j) から 0 画素までの最小ユークリッド距離であるような画像である。

$\mathbf{B}$ から $\mathbf{D}$ を求める処理を (ユークリッド) 距離変換という。また、距離変換を施して得られる画像 $\mathbf{D}$ 自体も ($\mathbf{B}$ の) 距離変換と呼ぶことがある。

画像処理ではユークリッド距離の他に次の重み付き 4 近傍距離がよく用いられる。xy 平面上の 2 点 $p_1 = (x_1, y_1), p_2 = (x_2, y_2)$ の距離 $d(p_1, p_2)$ を次式で定める。

$$d(p_1,p_2) = \begin{cases} w_0|x_1 - x_2| + w_1|y_1 - y_2| \\ \qquad (|x_1 - x_2 > |y_1 - y_2| \text{ のとき}) \\ w_1|x_1 - x_2| + w_0|y_1 - y_2| \\ \qquad (otherwise) \end{cases}$$

ここで、$w_0 \ge 0, w_1 \ge 0$ である。$w_0 = w_1 = 1$ のとき、この距離は 4 近傍距離 (cityblock 距離) と呼ばれ、$w_0 = 1, w_1 = 0$ のとき 8 近傍距離 (chessbord 距離) と呼ばれる。また、$w_0 = 1, w_1 = \sqrt{2} - 1$ のときチャムファー (chamfer) 距離 (または準ユークリッド距離) と呼ばれ、$w_0 = 1, w_1 = \frac{1}{\sqrt{2}} - 1 + \sqrt{\sqrt{2} - 1} \approx 0.351$ のとき最適チャムファー距離と呼ばれる [1]。

実数 $p(1 \le p \le \infty)$ に対し L_p 距離は次のように定義される。

$$d(p_1,p_2) = (|x_1 - x_2|^p + |y_1 - y_2|^p)^{1/p}$$

したがって、ユークリッド距離は L_2 距離のことであり、4 近傍距離は L_1 距離、8 近傍距離は L_∞ 距離と一致する。

さらに、8 角形距離は次のように定義される。

$d(p_1,p_2) = \max\{|x_1 - x_2|, |y_1 - y_2|, \lfloor 2(|x_1 - x_2| + |y_1 - y_2| + 1)/3 \rfloor\}$

3 距離変換アルゴリズム

ここでは [3] で提案されたユークリッド距離変換アルゴリズムを紹介する。

3.1 基本アルゴリズム

まず、アルゴリズムの基本形を解説する。これは、[8] で提案されたアルゴリズムの 2 次元版である。$N \times N$ の入力 2 値画像 $\mathbf{B} = \{b_{ij}\}$ に対して次の 2 つの変換を順に実行する。

変換 1 $\mathbf{B} = \{b_{ij}\}$ に対し、各列 jごとに列方向のみを参照して距離変換を行い画像 $\mathbf{G} = \{g_{ij}\}$ を求める。すなわち、

$$g_{ij} = \min_{1 \le p \le N} \{|i - p| \mid b_{pj} = 0\} \quad (2)$$

である。ただし、0 を含まない列 jについては $g_{ij} = \infty (1 \le i \le N)$ とする。

変換 2 $\mathbf{G} = \{g_{ij}\}$ に対し、各行 i ごとに行方向のみを参照して画像 $\mathbf{D}' = \{d'_{ij}\}$ を求める。ただし、

$$d'_{ij} = \min_{1 \le q \le N} \{(j - q)^2 + g_{iq}^2\} \quad (3)$$

である。

このとき、次の補題が成り立つのは容易に分かる。

[補題 1] 上で求められた画像 $\mathbf{D}'$の各画素 $d'_{i,j}$の値は、$\mathbf{B}$ において画素 (i,j) からもっとも近い 0 画素までのユークリッド距離を 2 乗した値である。

変換 1 をディジタル画像上で実行するには、各列ごとに 2 回の走査を行なえばよい。よって、変換 1 の時間計算量は $O(N^2)$ である。また、変換 2 を実行するには各画素ごとに次の処理を行なえばよい。

$$d'_{ij} \leftarrow \min_{-g_{ij} \le l \le g_{ij}} \{g^2_{i(j+l)} + l^2\}$$

ただし、最小値の探索は lを順に変えることで行ない、また、画像内 ($1 \le j + l \le N$の範囲) のみで探索する。このように、変換 2 は行方向の全画素を走査する必要はなく、(i,j) から距離 g_{ij}の範囲内のみ走査すればよいので、その時間計算量は $O(\mathbf{G}$ の画素値の平均値$\times N^2)$ である。しかし、0 画素の数が少ない場合や画像中に大きな図形がいくつも存在する場合には $\mathbf{G}$ の画素値の平均値が $O(N)$ になるため、最悪時間計算量は $O(N^3)$ となる。

なお、[9] ではこの基本アルゴリズムの改良版を提案している。これは、変換2の行走査における最小値の探索方法を改良することにより高速化を図ったものであるが、入力画像によっては行の走査の際に戻りがひんぱんに生じ、最悪時間計算量はやはり $O(N^3)$ となる。

3.2 変換2のアルゴリズム

ここでは、上の基本アルゴリズムの変換2を効率良く実行する方法を与える。いま、i 行目の走査を考える。第 k 列に存在する0画素から (i,j) へのユークリッド距離の2乗の最小値を $f_k^i(j)=(j-k)^2+g_{ik}^2$ と表す。すると、$d'_{ij}=\min_{1\le k\le N} f_k^i(j)$ である。したがって、各行ごとに行なう処理は、関数の集合 $F_N^i=\{f_k^i(x)\mid 1\le k\le N\}$ の下側包絡線 $\min_{1\le k\le N} f_k^i(x)$ を求めることに帰着する。以下では、文脈から明らかなときは $f_k^i(x)$ と F_N^i をそれぞれ $f_k(x)$ と F_N と書く。

$F_N=\{f_k(x)\mid 1\le k\le N\}$ に対し、その下側包絡線を与える関数の集合を $L_{\mathrm{env}}(F_N)$ と記述する。すなわち、

$$L_{\mathrm{env}}(F_N)=\{f_{i_1}(x),f_{i_2}(x),\cdots,f_{i_m}(x)\}$$

$$\min_{1\le k\le N} f_k(x)=\begin{cases} f_{i_1}(x) & (x<x_{i_1i_2})\\ f_{i_2}(x) & (x_{i_1i_2}\le x<x_{i_2i_3})\\ \quad\vdots & \\ f_{i_m}(x) & (x_{i_{m-1}i_m}\le x)\end{cases}$$

である。ただし、x_{ij} は関数 $f_i(x)$ と $f_j(x)$ の交点の x 座標である。次の補題2と補題3が成立する。

[補題2] $i<j<k$ で $f_i(x)=(x-i)^2+b_i$, $f_j(x)=(x-j)^2+b_j$, $f_k(x)=(x-k)^2+b_k$ の交点の x 座標を x_{ij},x_{jk},x_{ik} とすると、$x_{ij}\le x_{ik}\le x_{jk}$ または $x_{jk}\le x_{ik}\le x_{ij}$ である。

[補題3] $f_i(x)=(x-i)^2+b_i,\quad (i=1,2,\cdots,N)$ とする。$F_k=\{f_i(x)\mid 1\le i\le k\}$ の下側包絡線関数集合を $L_{\mathrm{env}}(F_k)=\{f_{i_1}(x),f_{i_2}(x),\cdots,f_{i_j}(x)\}(i_1<i_2<\cdots<i_j)$ とすると、$L_{\mathrm{env}}(F_{k+1})$ は次式で表される。

$$L_{\mathrm{env}}(F_{k+1})=\begin{cases} L_{\mathrm{env}}(F_k)\cup\{f_{k+1}(x)\} & (x_{i_{(j-1)}i_j}<x_{i_ji_{(k+1)}})\\ L_{\mathrm{env}}((L_{\mathrm{env}}(F_k)-\{f_{i_j}(x)\})\cup\{f_{k+1}(x)\}) & (x_{i_{(j-1)}i_j}\ge x_{i_ji_{(k+1)}})\end{cases}$$

[証明] $x_{i_{(j-1)}i_j}<x_{i_ji_{(k+1)}}$ のときは明らか。$x_{i_{(j-1)}i_j}\ge x_{i_ji_{(k+1)}}$ のとき補題2より、$x_{i_ji_{(k+1)}}\le x_{i_{(j-1)}i_{(k+1)}}\le x_{i_{(j-1)}i_j}$ である。よって、

$$\begin{aligned}
&f_{i_{(j-1)}}(x)\le f_{i_j}(x)\le f_{k+1}(x) && (x\le x_{i_ji_{(k+1)}})\\
&f_{i_{(j-1)}}(x)\le f_{k+1}(x)\le f_{i_j}(x) && (x_{i_ji_{(k+1)}}\le x\le x_{i_{(j-1)}i_{(k+1)}})\\
&f_{k+1}(x)\le f_{i_{(j-1)}}(x)\le f_{i_j}(x) && (x_{i_{(j-1)}i_{(k+1)}}\le x\le x_{i_{(j-1)}i_j})\\
&f_{k+1}(x)\le f_{i_j}(x)\le f_{i_{(j-1)}}(x) && (x_{i_{(j-1)}i_j}\le x)
\end{aligned}$$

となる。したがって、下側包絡線は、$\min_{1\le i\le k+1} f_i(x)=$

$$\begin{cases} f_{i_1}(x) & (x<x_{i_1i_2})\\ f_{i_2}(x) & (x_{i_1i_2}\le x<x_{i_2i_3})\\ \quad\vdots & \\ f_{i_{(j-1)}}(x) & (x_{i_{(j-2)}i_{(j-1)}}\le x\le x_{i_{(j-1)}i_{(k+1)}})\\ f_{k+1}(x) & (x_{i_ji_{(k+1)}}\le x)\end{cases}$$

である。これより、

$$L_{\mathrm{env}}(F_{k+1})=L_{\mathrm{env}}(L_{\mathrm{env}}(F_k)\cup\{f_{k+1}(x)\})$$

$$=L_{\mathrm{env}}((L_{\mathrm{env}}(F_k)-\{f_{i_j}(x)\})\cup\{f_{k+1}(x)\})$$

であることが分かる。□

補題3を用いて変換2を効率よく実行するアルゴリズムを次のように構成する。先にも述べたように、変換2の各行 i ごとに行なう処理は F_N の下側包絡線を求めることに帰着されるので、我々はまず $L_{\mathrm{env}}(F_N)$ を求める。$L_{\mathrm{env}}(F_2)=\{f_1,f_2\}$ である。(簡単のため $g_{i1}\ne\infty, g_{i2}\ne\infty$ と仮定する。) あとは補題3の漸化式より、$L_{\mathrm{env}}(F_3),L_{\mathrm{env}}(F_4),\cdots$ と順次求めていく。このためにスタックを用いる。$L_{\mathrm{env}}(F_N)$ がスタックに求まったら、スタック内の関数の値を各画素に代入する。以下このアルゴリズムを示す。

[アルゴリズム T2]

1: 画像 **G** の各行 i について次の 2〜19 を繰り返す;
2: **begin**
3: 空のスタック s を用意する;
4: push$(s,1)$; { スタック s に $f_1(x)$ を push}
5: push$(s,2)$; { スタック s に $f_2(x)$ を push}
6: { スタック s の先頭の 2 要素を $t, t'(t' < t)$ とする }
7: **for** $j := 3$ **to** N **do**
8: **if** $g_{ij} \neq \infty$ **then**
9: **begin**
10: **while** $x_{tj} < x_{t't}$ **do** pop(s);
11: push(s,j) { スタック s に $f_j(x)$ を push}
12: **end**;
13: **while** $N < x_{t't}$ **do** pop(s);
14: **for** $j := N$ **downto** 1 **do**
15: **begin**
16: **if** $j < x_{t't}$ **then** pop(s);
17: $d_{ij} := f_t(j)$
18: **end**;
19: **end**

3.3 アルゴリズム T2 の計算量

アルゴリズム T2 は、画像の行ごとの処理に分解されている。各行ごとに、下側包絡線集合 $L_{\mathrm{env}}(F_N)$ を求め (ステップ 2〜ステップ 12)、下側包絡線関数の値を各画素に代入している (ステップ 13〜ステップ 18)。$L_{\mathrm{env}}(F_N)$ を求めるときの時間計算量について考察する。スタックに対して施される演算回数に着目すると、push はたかだか N回、pop もたかだか N回である。よって、各行ごとの処理の計算量は $O(N)$ で、アルゴリズム T2 の計算量は $O(N^2)$ となる。

変換 1 が $O(N^2)$ でできることと合わせて、$O(N^2)$ 時間でユークリッド距離変換を求めることができることになる。

3.4 3 次元への拡張と並列化

上で与えたアルゴリズムは、3 次元ディジタル画像のユークリッド距離変換にもすぐに拡張できる。入力の 3 次元ディジタル 2 値画像 $\mathbf{B} = \{b_{ijk}\}$ のサイズを $N \times N \times N$ とする。kを固定したときの 2 次元ディジタル画像 $\mathbf{B_k} = \{b_{ijk}\}$ のそれぞれに対し 2 次元ユークリッド距離変換を行なった結果の画像を $\mathbf{D_k} = \{h^k_{ij}\}$ とする。ここまでの処理は $O(N^2) \times N = O(N^3)$ 時間でできる。次に、i, jを固定してこれらの N枚の距離変換画像を k軸の方向にスキャンし、

$$d'_{ijk} = \min_{1 \leq q \leq N} \{(k-q)^2 + (h^q_{ij})^2\} \qquad (4)$$

を求める。これはアルゴリズム T2 を用いれば $O(N)$ 時間でできる。このスキャンをすべての $i, j (1 \leq i, j \leq N)$ について行なえば 3 次元画像のユークリッド距離変換が完了するので、このアルゴリズムの計算時間は $O(N^3)$ である。

このアルゴリズムはスキャンの方向が行方向と列方向 (3 次元の場合はさらに k軸方向) に固定しているため、等高線スキャンを用いた逐次アルゴリズムにくらべ並列処理に向いている。すなわち、列方向の N回のスキャンはそれぞれ独立しているので $p (1 \leq p \leq N)$ 台のプロセッサを用意して共有メモリに入力と結果の画像を書き込むことにすれば $O(N^2/p)$ 時間で実行できる。行方向も同様である。よって、2 次元画像の場合、$p (1 \leq p \leq N)$ 台のプロセッサで $O(N^2/p)$ 時間の並列アルゴリズムが構成でき、3 次元画像のとき $O(N^3/p)$ 時間の並列アルゴリズムが構成できる。

4 距離関数のクラス

前章で紹介したアルゴリズムは、処理を列方向 (変換 1) と行方向 (変換 2) に分解して距離変換を求めている。このような方法で正しく距離変換が求まるのは、ユークリッド距離の場合に限らない。実際、Paglieroni[5] は、2 点 $p_1 = (x_1, y_1), p_2 = (x_2, y_2)$ 間の距離を $d(p_1, p_2)$ とするとき、$d(p_1, p_2)$ が x 座標の差と y座標の差をのみに依存する関数で、かつ、それぞれの引数に関し単調増加であれば処理を各軸方向に分解する方法で正しく距離変換が求められることを示している。すなわち、

$d(p_1, p_2) = f(|x_1 - x_2|, |y_1 - y_2|)$ なる関数 f が存在し,

$\forall y\ |x_1| < |x_2| \Longrightarrow f(|x_1|, |y|) \leq f(|x_2|, |y|)$,

$\forall x\ |y_1| < |y_2| \Longrightarrow f(|x|, |y_1|) \leq f(|x|, |y_2|)$

である。

4 近傍距離や 8 近傍距離など画像処理の分野で用いられる距離はほとんど軸方向の処理に分解して距離変換を行なうことが可能である。

アルゴリズム T2 では、2 次関数の集合 $F_n = \{f_1(x), f_2(x), \cdots, f_n(x)\}$ の下側包絡線 $L_{\mathrm{env}}(F_n)$ と関数 $f_{n+1}(x)$ の交点を計算し $L_{\mathrm{env}}(F_{n+1})$ を求めている。このとき関数 $f_{n+1}(x)$ は他の関数 $f_k(x)$ $(k = 1, \cdots, n)$ のどれともたかだか 1 個しか交点を持たないことを利用している。4 近傍距離と 8 近傍距離の場合も同じ性質をもっているので、3 章のアルゴリズムによって（正確な）距離変換を $O(N^2)$ 時間で求めることができる。

具体的には、アルゴリズム T2 中の関数を次のように置き換えればよい。

ユークリッド距離変換 $\cdots$ $f_k^i(j) = (k-j)^2 + g_{ik}^2$

4 近傍距離変換 $\cdots$ $f_k^i(j) = |k-j| + g_{ik}$

8 近傍距離変換 $\cdots$ $f_k^i(j) = max(|k-j|, g_{ik})$

これらの関数について補題 2 と補題 3 が成立するのは容易に確かめられる。(4 近傍距離や 8 近傍距離の場合に、$f_i(x)$ と $f_j(x)$ が一部で重なる場合があるが、そのときは一方をわずかにずらして交点を求めればよい。) このように、この距離変換アルゴリズムは、画像処理にあらわれる他の距離関数に対しても有効である。以下ではそのような距離関数のクラスについて考察する。なお、ここでは距離はすべて距離公理を満たすものとする。

[定義] 直線上に並んだ 3 点 p_1, p_2, p_3 に対し、

$$d(p_1, p_2) + d(p_2, p_3) = d(p_1, p_3)$$

が成立するとき、その距離は直線等式を満たすという。

[補題 4] 重み付き 4 近傍距離、L_p 距離、それに 8 角形距離は、いずれも Paglieroni の条件と直線等式を満たす。

[補題 5] $p = (x_p, 0), p' = (x_{p'}, 0)$ を x 軸上の 2 点、$a = (x_a, y_a), b(x_b, y_b)(x_a < x_b)$ を x 軸上にない 2 点で $d(a,p') = d(b,p')$ であるとする。また、距離関数 $d(p_1, p_2)$ は Paglieroni の条件と直線等式を満たすとする。このとき、$x_{p'} < x_p$ なら $d(a,p) \geq d(b,p)$ であり、$x_{p'} > x_p$ なら $d(a,p) \leq d(b,p)$ である。

[証明] まず、$y_a > 0, y_b > 0$ または $y_a < 0, y_b < 0$ の場合を考える。$x_{p'} < x_p$ とする。a も b も x 軸の同じ側にあるので、線分 ap と線分 bp' の交点を c とする。(このような交点がないときは、$|y_a| > |y_b|$ かつ $|x_p - x_a| > |x_p - x_b|$ であり、Paglieroni の単調性の条件より $d(a,p) \geq d(b,p)$ である。) このとき、$d(a,c) \geq d(b,c)$ である。なぜなら、$d(a,c) < d(b,c)$ とすると、

$$d(a,c)+d(c,p') < d(b,c)+d(c,p') = d(b,p') = d(a,p')$$

となり、距離公理 (3 角不等式) に反するからである。もういちど 3 角不等式を用いて、

$$d(b,p) \leq d(b,c) + d(c,p) \leq d(a,c) + d(c,p) = d(a,p)$$

となり証明ができた。$x_{p'} > x_p$ の場合も同様にして証明できる。

$y_a > 0, y_b < 0$ の場合は、$b' = (x_b, -y_b)$ なる点を考えると、Paglieroni の条件 ($d(p_1,p_2) = f(|x_1 - x_2|, |y_1 - y_2|)$) より $d(b',p') = d(b,p'), d(b'p) = d(b,p)$ である。よって、上の議論によりこの場合も証明ができる。□

補題 5 は、変換 2 で下側包絡線を求める関数 $f_i(x)$ のそれぞれが、他とたかだか 1 箇所でしか交わらないことを保証する。つまり、a と b の位置に 0 画素があったとき、i 行目のスキャンで、$f_{x_a}(x)$ と $f_{x_b}(x)$ の交わりの x 座標が p' に対応する。($f_{x_a}(x)$ と $f_{x_b}(x)$ が一部で重なる場合は一方をわずかにずらすものとする。) 以上をまとめて次の定理を得る。

[定理] 距離が Paglieroni の条件と直線等式を満たすならば、その距離に関する距離変換は $O(N^2)$ で求めることができる。

この定理の条件を満たす距離のクラスは広く、上で挙げたユークリッド距離、4 近傍距離、8 近傍距離はもちろん、その他にチャムファー (chamfer) 距離や 8 角形距離などをすべて含む。

5 まとめ

本論文では先に提案した $N \times N$ ディジタル 2 値画像のユークリッド距離変換を $O(N^2)$ で実行するアルゴリズムが、画像処理でよく用いられるほとんどの距離に対して有効であることを示した。

本文中ではふれなかったが距離変換の逆変換として定義されている逆距離変換にもこのアルゴリズムのアイデアを用いることできる。逆距離変換はスケルトンから元の 2 値画像を復元する際に必要な処理であり、スケルトンとその上の距離値を入力とし、2 値画像を出力する。本論文が与えた距離のクラスについては、逆距離変換も効率よく実行できることになる。

謝辞

貴重なコメントをいただきました名古屋大学 鳥脇純一郎先生と斉藤豊文博士に感謝致します。

参考文献

[1] Borgefors, G. :Distance transformations in digital images, Computer Graphics & Image Processing, 34, pp. 344-371(1986).

[2] Danielsson, P. E.: Euclidean distance mapping, Computer Graphics & Image Processing, 14, pp. 227-248(1980).

[3] 加藤、斉藤、平田:ユークリッド距離変換アルゴリズムの改良, 電気関係学会東海支部連合大会講演論文集, page 369(1993).

[4] Leymarie, F. and M. D. Levine: Fast raster scan distance propagation on the discrete rectangular lattice, Computer Graphics & Image Processing: Image Understanding, 55, pp. 84-94(1992).

[5] Paglieroni, D. W.: Distance transforms: properties and machine vision applications, Computer Graphics & Image Processing: GMIP, 54, pp. 56-74(1992).

[6] Ragnemalm, I:Neighborhoods for distance transformations using ordered propagation, Computer Graphics & Image Processing: Image Understanding, 56, pp. 399-409(1992).

[7] Rosenfeld, A. and J. Pfalts:Sequential operations in digital picture processing, J. Assoc. Comput. Mach., 13, pp. 471-494(1966).

[8] 斉藤、鳥脇:3次元ディジタル画像に対するユークリッド距離変換, 電子情報通信学会論文誌, J76-D-II, pp.445-453(1993).

[9] Saito, T. and J. Toriwaki:Fast algorithms for n-dimensional Euclidean distance transformatin",Proc. of 8th Scandinavian Conference on Image Analysis, Vol. II, pp.747-754(1993).

[10] Serra, J:Introduction to mathematical morphology, Computer Graphics & Image Processing, 35, pp. 283-305(1986).

[11] Yamada, H.:Complete Euclidean distance transformation by parallel operation, Proc. of 7th Int. Conf. on Pattern Recognition, Montreal, pp. 69-71(1984).