卒業論文

# 文書画像中のグラフの理解に関する研究

東北大学 工学部 通信工学科４年
布田 寿康

# 目 次

# 図目次

# 第1章

# 序論

## 1.1 研究の背景

我々は日々、情報源の一つとして紙面に記述された文書を用いている。これら紙面に記述された大量の文書を保存、検索することは容易ではないが、それらの文書を電子化することで、多量の文書の保存や検索を計算機によって容易に実現することができる。そこで、近年紙面に記述、または描写された情報の電子化は、文書理解、図面認識などのさまざまな分野で試みられ、各分野で多くの成果を上げている。

しかし、文書画像中のグラフは、単に画像として電子化されるだけに留まっており、ユーザーが任意のレイアウトやスタイルに変更して出力することや、電子化されたグラフの再利用などが困難であるのが現状である。文書画像中のグラフを単に画像としてではなく、個々のデータとして認識することが可能になれば、場面に応じたスタイルなどの変更はもちろん、より大量のグラフの保存などを容易に実現することができる。このような理由から、ビジネスグラフを認識対象とした研究が横倉ら [1] や Lee ら [2] によってなされている。

横倉らの研究は、ビジネスグラフの中でも特に棒グラフを認識対象としたものであり、Lee らの研究は特に折れ線グラフを認識対象とした研究である。Lee らの認識手法は、グラフ画像中の折れ線を形成しているマーカーをパターン認識で区別し、そのマーカーの場所を軸のラベルと比較することで折れ線グラフを認識するというものである。しかし、現在、紙面に記述、または描写された折れ線グラフには、マーカーが表示されていないものも多く、それらのグラフに対しては Lee らの認識手法では考慮されていない。加えて、マーカーが表示されていないグラフにおいては折れ線を記述している線種が点線の場合も多く存在するが、このような折れ線グラフに対しても考慮されていない。また、図 1.1 のような折れ線グラフも文書中には多く存在しており、このようなグラフに対しても Lee らの認識手法では、考慮されていない。

## 1.2 研究の目的

本研究では、マーカーが表示されていない文書中の折れ線グラフを認識対象とし、グラフの認識を行うことを目的とする。

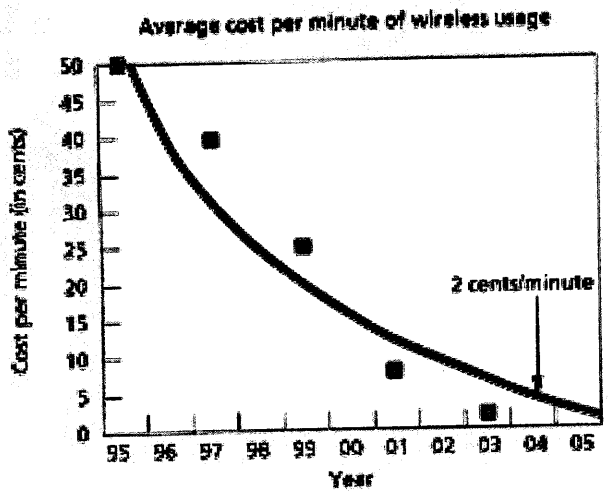



図 1.1: Lee らの認識手法で考慮されていないグラフの例

## 1.3 本論文の構成

本論文の構成は以下の通りである.

**第2章** 本研究における認識手法の流れと個々の処理の主旨について述べる。

**第3章** 入力画像をグラフ画像と文字画像とに分離する手法について述べる。

**第4章** 入力画像から分離されたグラフ画像に対する処理方法について述べる。

**第5章** 入力画像から分離された文字画像に対する処理方法、特に文字列の抽出方法について述べる。

**第6章** グラフ画像に対する処理結果と文字画像に対する処理結果を元に軸の読み替えの手法について述べる。

**第7章** 入力画像が白黒、つまり2値の場合についての処理方法について述べる。

**第8章** 認識実験の結果と考察について述べる。

**第9章** 結論である.

# 第2章

# グラフ認識手法

## 2.1 はじめに

本章では、グラフを認識する際の提案手法全体の流れについて説明する。

## 2.2 提案手法

図 2.1 に提案手法全体の流れを示した。
以下で、各過程の主旨を説明する。

### 2.2.1 色のクラスタリング、ノイズの除去

文書をスキャナーで計算機に取り込むときに、元画像に対する色の変化を完全に防ぐことは困難である。よって、計算機に取り込まれた画像においては、元画像では同色であったものでも、違った色として描写されていまう。このことは、直線の種類を色で判断しようというときの大きな妨げになる。そこで、画像を取り込んだ後、即認識に取り掛かるのではなく、入力画像の色をクラスタリングし、ある程度色の統一を図る必要がある。

また、無用なノイズは認識過程において正しい認識を妨げるので、この時点で除去する。

### 2.2.2 グラフ画像と文字画像への分離

前処理後の画像をグラフ画像と文字画像とに分離し、それぞれ異なった処理を実行する。
詳細な分離手法に関しては第 3 章で説明する。

### 2.2.3 グラフ画像と文字画像の統合

グラフ画像と文字画像の各認識結果を統合することによって最終的な出力データとする。具体的には、文字画像から得られた軸の情報を用いて、グラフ画像のデータを読み替えるという処理

を実行する。

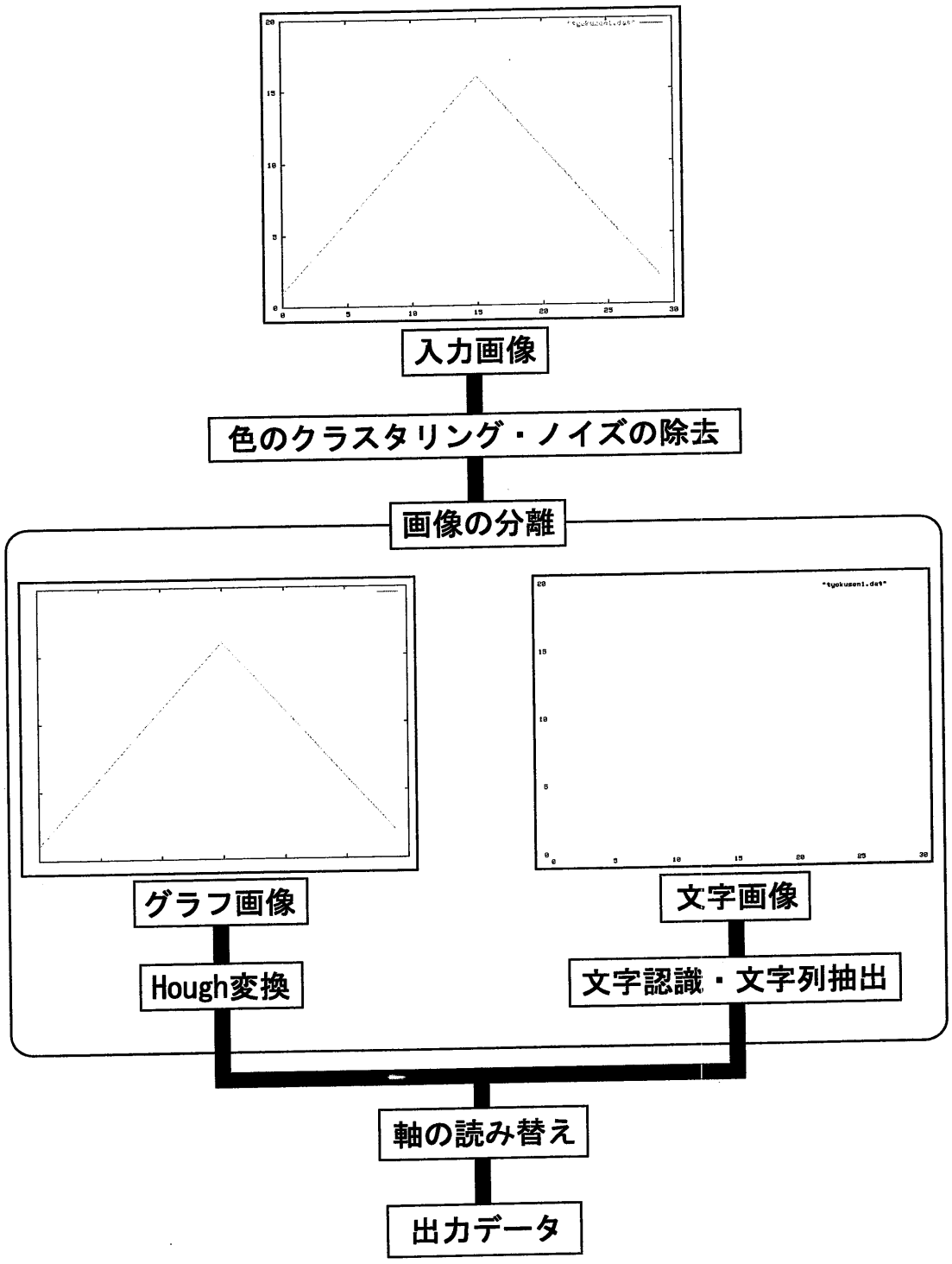



図 2.1: 本研究の認識手法の流れ

# 第3章

# 前処理

## 3.1 はじめに

本章では、入力画像の色のクラスタリング、ノイズの除去の方法について説明する。また、入力画像をグラフ画像と文字画像に変換する方法についても説明する。

## 3.2 色のクラスタリング

色をクラスタリングする手法としては、RGB をベクトルとみなしてマハラノビス距離などを利用することなどが考えられるが、今回は比較的単純な入力画像を念頭においているので、次のような方法で色のクラスタリングを実行した。

1 RGB それぞれにおいて、分布を調べる。

2 それぞれの分布において、分散が最大になる輝度を閾値とする。

3 閾値よりも大きな輝度のものは、輝度を最大に、閾値よりも小さな輝度のものは、輝度を最小にすることで、色を統一する。

上記の手法を用いることによって、最大で 8 色まで正しいクラスタリングができる。

## 3.3 ノイズの除去

入力画像に比べて極端に小さな画素の連結成分はノイズとして消去する。

本来、本手法の主軸となっている Hough 変換はノイズに対して頑健であるが、後の 4.3.4 で説明する手法をより有効に実行できるように、この時点である程度のノイズは除去しておく必要がある。

## 3.4 グラフ画像と文字画像の分離

入力画像の大きさと比較して、文字らしい画素の連結成分は文字として文字画像に分離する。

文字らしいとは、入力画像に対して比較的小さな画素の連結成分とした。

入力画像において、文字画像に分離されなかった比較的大きな画素の連結成分は、グラフ画像として使用する。

ここで、グラフ画像と文字画像に分離する理由は、後の 4.3.4 で説明する処理において余計な変換を避けることや、4.3.3 において誤認識を避けることなどが挙げられる。

画像分離の具体例は図 2.1 に示した。

# 第4章

# グラフ画像に対する処理

## 4.1 はじめに

本章では、グラフ画像からHough変換によって線分を抽出する過程、折れ線グラフの折れ点を検出する過程などグラフ画像に対する処理について説明する。なお、本章ではグラフ画像がカラーの場合について説明し、白黒の2値画像については7で説明する。

## 4.2 軸の検索

Hough変換を実行する領域をグラフ領域のみに限定するためグラフの軸の検索を第一に実行する。なお、グラフ領域とは軸に囲まれた領域のことを示す。

$x$軸の検索方法は以下の通りである。

**1** グラフ画像を横方向にスキャンし、白画素以外の画素の累積数をとる。

**2** 累積数が大きく、十分に離れている場所を$x$軸の場所であるとする。

図4.1に具体例を示す。

$y$軸に関しても同様の方法で検索することが可能である。

通常、スキャナーで取り込んだ画像の線幅は1ではないので、画素の累積数が最大の場所が2ヶ所以上検出される。そこで、軸の場所として検出するのは、より入力画像の端に近いものとする。

## 4.3 線分の抽出

### 4.3.1 Hough変換による直線の検出

Hough変換は、画像中の直線の抽出を目的とした変換である。

図4.2に示す直線は、$x$軸との角度$\theta$と原点との距離$r$を用いると以下の式で表される。

$$r = x\cos(\theta) + y\sin(\theta) \tag{4.1}$$

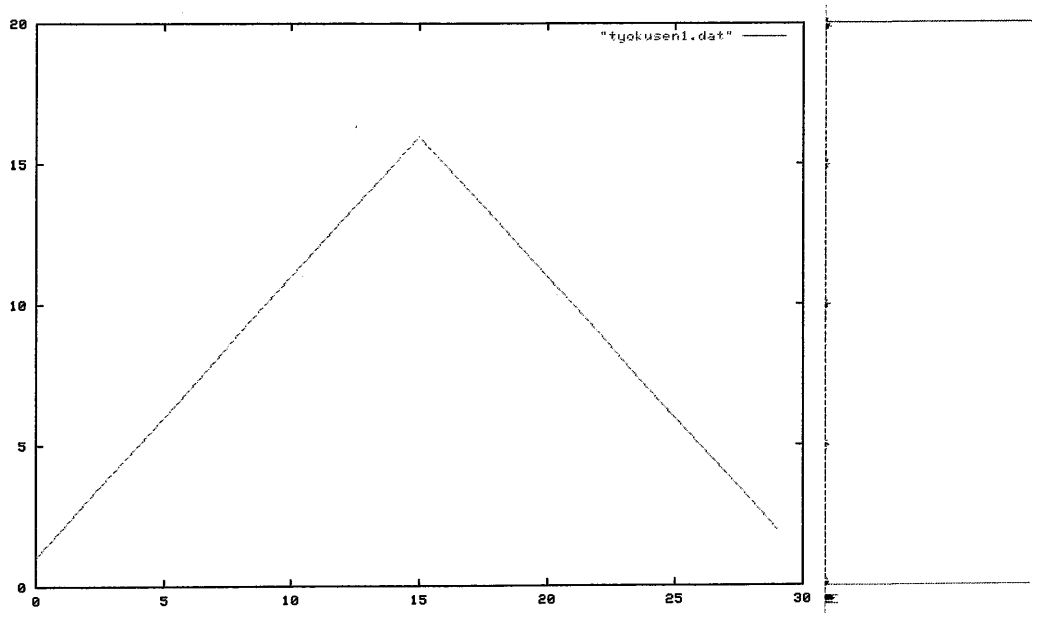



図 4.1: x 軸検索の例

通常、ある直線が $x-y$ 平面上に存在するときには、(4.1) 式において $r$ と $\theta$ が一意に決定されるが、逆に $r$ と $\theta$ が決まっていれば (4.1) 式から $x-y$ 平面上に存在する直線を一意に決定することができる。

そこで、画像中に存在する全ての画素 $(x_i, y_i)$ を (4.2) 式によって Hough 平面と呼ばれる $\theta-r$ 平面に投票し、投票数の多い $(\theta, r)$ を (4.1) 式に代入する。すると、元の $x-y$ 平面上に存在する直線が検出できる。

$$r = x_i \cos(\theta) + y_i \sin(\theta) \tag{4.2}$$

以上の操作の概念図を図 4.3 に示す。

本章では、カラーのグラフ画像を前提としているので、Hough 変換を行う画素の色を限定することによって、異なる種のグラフ要素と混同することを避けることができる。

また、この Hough 変換は画像中の全ての同色画素に対して行うため、検出対象の直線が実線である必要は無く、点線で描かれている直線に対しても正しい検出ができるという利点を持っている。

但し、Hough 変換を利用するに当たっては以下の問題点に留意する必要がある。

**問題点 1** 計算機上では、なめらかな直線を描写できないため、本来一本であるはずの直線を Hough 変換しても複数の直線を検出してしまう場合がある。

**問題点 2** Hough 変換はあくまでも直線を検出するための変換であるため、折れ線グラフのような線分の集合を検出するためには工夫が必要である。

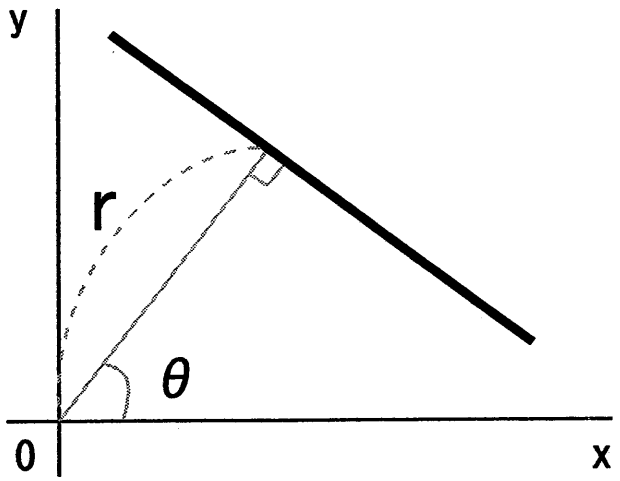



図 4.2: Hough 変換の原理

**問題点 3** Hough 平面上から検出する最低投票数の値によっては細かな線分を検出できない場合がある。

以上の問題点を考慮した認識手法を以下に示す。

### 4.3.2 検出直線の統合

本節では 4.3.1 で示した**問題点 1** を考慮した直線の統合方法を説明する。

(4.1) 式からもわかるように、Hough 平面上で近傍にある点 $(\theta, r)$ は変換前の $x-y$ 平面上では似通った直線である。このことを利用して、直線の統合を行う。

Hough 平面上のある点 $(\theta_i, r_i)$ を検出したときに、図 4.4 に示すような領域に入っているものは、元の $x-y$ 平面上では同一の直線を形成しているとみなす。ここで、統合後の直線の $\theta$ と $r$ を以下のように定義する。

$$\theta = \frac{\sum_{i=1}^{n} \theta_i}{n} \tag{4.3}$$

$$r = \frac{\sum_{i=1}^{n} r_i}{n} \tag{4.4}$$

$n$: 結合数

### 4.3.3 線分の検出

本節では 4.3.1 で示した**問題点 2** を考慮した線分の検出方法について説明する。

Hough 変換終了後に線分を検出する場合は、まず線分の折れ点を検出する必要がある。本研究の認識対象である折れ線グラフでは、線分の折れ点は Hough 変換によって検出された直線の交点

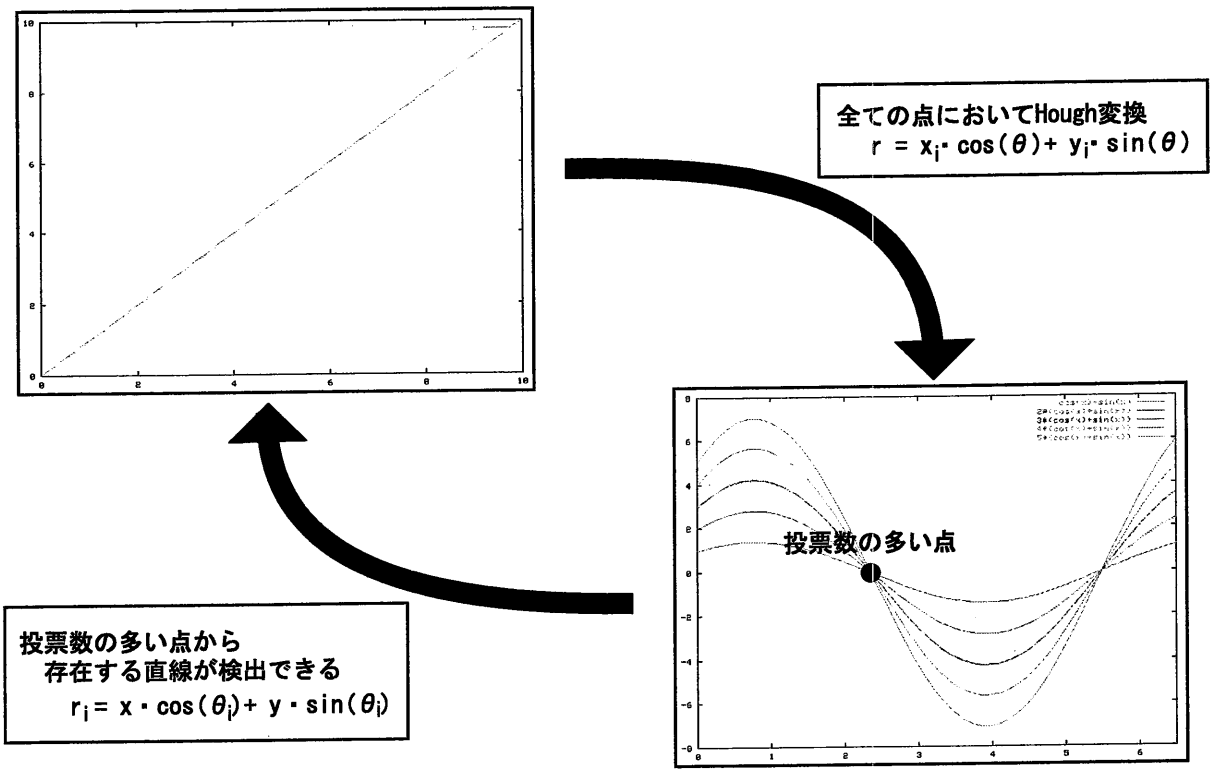



図 4.3: Hough 変換の概念図

となっている。そこで、Hough 変換によって検出された直線の全ての交点を求め、その交点に**検出直線と同色の画素が存在**したらその点を**線分の折れ点**とする。

その具体例を図 4.5 に示す。

本来、折れ線グラフでは線分の折れ点、つまり折れ線グラフにおける**折れ点**が検出された時点で認識過程としてはほぼ終了となるが、グラフ要素の画素色と同色の文字やノイズ、軸の目盛等がグラフ領域に存在していると、本来検出されるべきではない点が線分の折れ点として検出されてしまう。そこで、線分の折れ点を全て検出したところで、本当にその折れ点間に線分が存在しているかを確認する必要がある。

その確認の方法は以下に示す通りである。

**折れ点の間引き** $(x_{mid}, y_{mid})$ に検出直線と同色の画素が存在しなかった場合は、$(x_\alpha, y_\alpha)$ と $(x_\beta, y_\beta)$ を折れ点から削除する。但し、

$x_{mid} = \frac{x_\alpha + x_\beta}{2}$ , $y_{mid} = \frac{y_\alpha + y_\beta}{2}$

$(x_\alpha, y_\alpha), (x_\beta, y_\beta) \in$ 折れ点集合

とする。

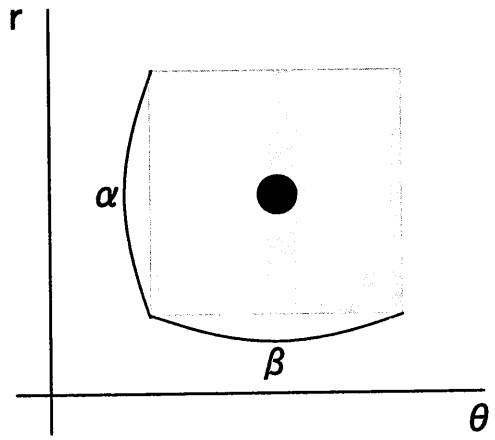



図 4.4: Hough 平面における直線の統合範囲

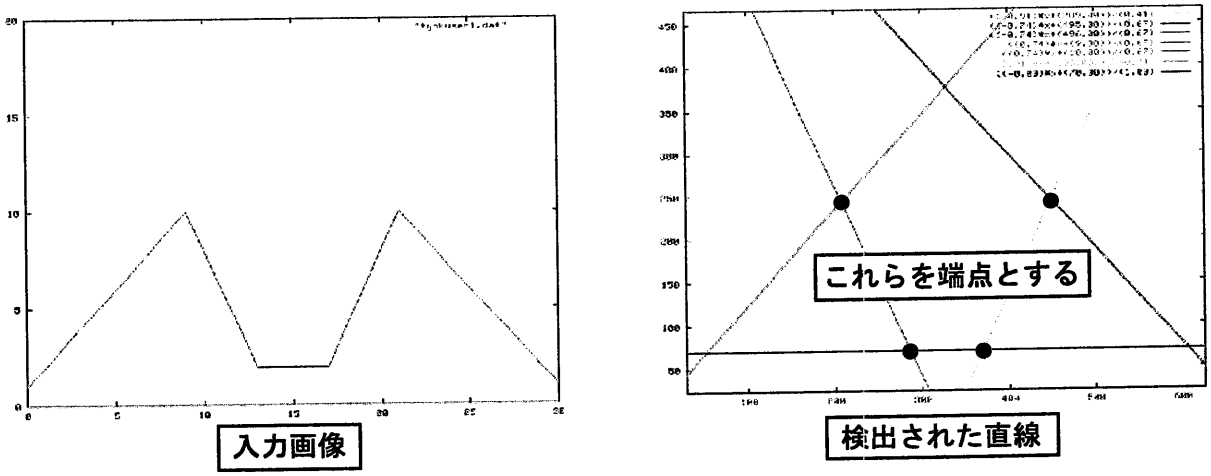



図 4.5: 線分の折れ点検出の例

### 4.3.4　細かい線分の検出

本節では、4.3.1 で示した**問題点 3** を考慮した線分の検出方法について説明する。

Hough 変換では、Hough 平面から検出する点の終了条件として、Hough 平面上の投票数を利用するのが一般的である。しかし、この方法を使用すると、無駄な直線を検出してしまったり、細かい線分が検出されなかったりする。これは、適切な終了条件が未知の場合は顕著に現れることである。

そこで、どんなグラフ要素に対しても適切な終了条件を与える必要がある。

本手法では、Hough 変換の終了条件を「Hough 平面に写像する画素が無くなったとき」とした。具体的には、以下のように処理をする。

1 Hough 変換を実行する。

2 Hough 平面から投票数最大の点 $(\theta, r)$ を検出する。

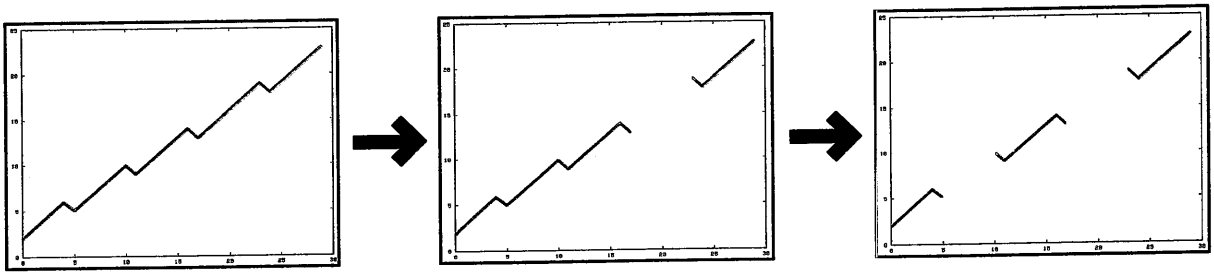


図 4.6: Hough 変換の対象の除去例

**3** グラフ画像から

$r = x\cos(\theta) + y\sin(\theta)$

の直線にのる点、つまり検出された Hough 平面上の点の投票に貢献した点、を除去する。

**4** Hough 変換すべき画素がグラフ画像に存在したら [1] に戻り、処理を続ける。Hough 変換すべき画素がグラフ画像に存在しない場合は、Hough 変換を終了する。

上記のように処理を実行することによって、一度直線の検出に貢献したグラフ画像の点は除去される。こうして二度以上直線の検出に貢献することを防ぐことで、無駄な直線が検出されることを避けることができる。

また、この手法を用いると処理が進むにつれて長い線分が除去されていくため、通常検出できない細かな線分も検出できるという利点もある。

その具体例を図 4.6 に示す。

## 4.4 線種が点線の場合

### 4.4.1 点線の場合の問題点

線種が点線の場合において問題となるのは、以下の 2 つである。

**問題点 1** 入力画像に対する前処理において点線を構成している点が、ノイズとして誤認識されてしまい入力画像から除去されてしまうこと。

**問題点 2** 入力画像をグラフ画像と文字画像に分離する過程で、点線を構成する点が文字として文字画像に分離されてしまうこと。

### 4.4.2 問題点の解決策

4.4.1 で示した問題点は、点線を構成する点が画素の連結成分として入力画像に対して比較的小さなものであることが大きな原因として生じている。

つまり、点線を構成する点を入力画像に対して比較的大きな画素の連結成分とすることができれば、先に示した 2 つの問題点を解決することができるということになる。

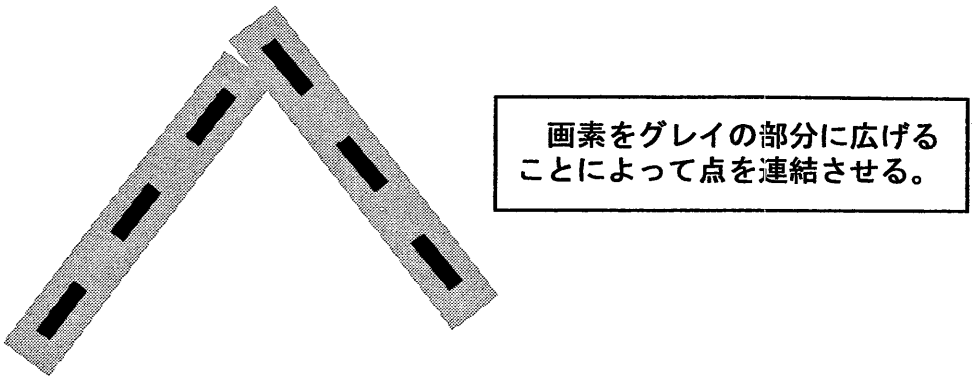



図 4.7: 画素拡張の概念図

そこで、点線を構成している点同士の間隔が他のグラフ要素間の間隔に対して非常に小さいことを利用し、入力画像の画素をまわりに広げることにより点同士を結合してやる。しかし、広げられた画素をそのまま処理の最後まで利用すると、グラフを構成する線幅が太くなってしまうため、画素を広げた大きさに比例して認識結果が悪くなってしまう。よって、画素を広げる処理を施した画像は、前処理におけるノイズの消去、入力画像のグラフ画像と文字画像への分離にのみ利用し、その後の処理の対象は、画素を広げる前の入力画像とする。こうすることによって、認識結果を変えることなく点線をグラフ画像に分離することが可能になる。

画素拡張を行う場合、拡張する幅は入力画像により適切な値が変化してくる。最終的には、点線を構成している点がノイズや文字領域に誤認識されない程度まで拡張されれば良い。よって、第8章で示すような程度の大きさの入力画像においては、画素の拡張幅は $2pixel$ 程度が適切な値となる。

この手法の概念図を図4.7に示す。

# 第5章

# 文字画像に対する処理

## 5.1 はじめに

本章では、文字列の抽出方法について説明する。

## 5.2 文字列の抽出

文字領域は入力画像の分離の時点で切り出されている。個々の文字領域については、既存の手法を使って通常の文字認識を実行する。

文字認識終了後、文字列として文字領域を結合しなければならないが、その方法は以下のようなものである。但し、$a, b$は定数である。

**1** 図5.1に示すような配置の文字列を結合するため、

$$x_{2_{min}} - x_{1_{max}} \leq a(x_{1_{max}} - x_{1_{min}}) \tag{5.1}$$

$$|y_{1_{min}} - y_{2_{min}}| \leq b \tag{5.2}$$

ならば、2つの文字領域を結合する。

**2** 図5.2に示すような配置の文字列を結合するため、

$$x_{2_{min}} - x_{1_{max}} \leq a(x_{1_{max}} - x_{1_{min}})$$

$$|y_{1_{max}} - y_{2_{max}}| \leq b \tag{5.3}$$

ならば、2つの文字領域を結合する。

**3** 図5.3に示すような配置の文字列を結合するため、

$$x_{2_{min}} - x_{1_{max}} \leq a(x_{1_{max}} - x_{1_{min}})$$

$$|\frac{y_{1_{min}} + y_{1_{max}}}{2} - \frac{y_{2_{min}} + y_{2_{max}}}{2}| \leq b \tag{5.4}$$

ならば、2つの文字領域を結合する。

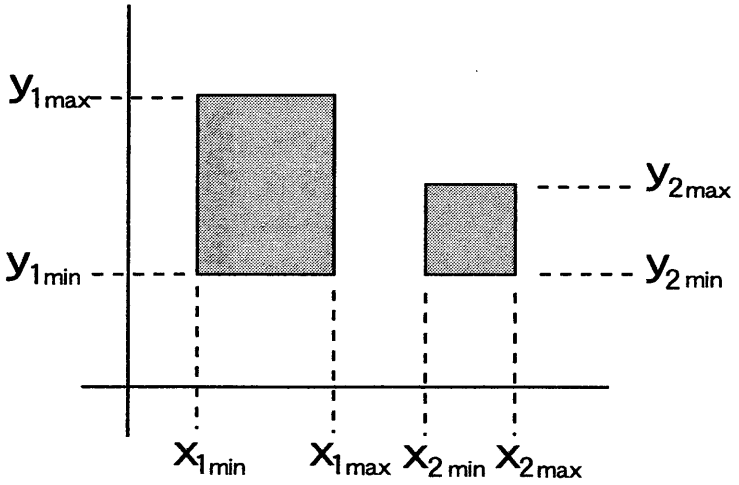



図 5.1: 文字領域の結合パターン 1

先に示した式を見ればわかるように、$a$は文字領域同士の間隔に、$b$は文字領域同士の縦方向へのずれに依存した定数である。よって、文字領域を文字列として結合する際に用いる定数$a, b$は、スキャナーで読み込んだ画像をどの程度拡大したものを入力画像にしているかによって変化する。通常、スキャナーで読み込んだ画像をそのまま入力画像としている場合、$a = 1.5$ , $b = 3$程度が適切な値となる。

個々の文字領域の文字認識は既に終了しているので、文字列抽出終了後、それぞれの文字列に対応する文字領域の認識結果を順に出力することによって、入力画像に記述されている文字列を再現することが可能である。

また、個々の文字列が存在する位置により、その文字列が表している情報、つまり所属するグラフ要素のクラスを以下のように定義する。

**x 軸の数値** 検出された原点を通るy座標よりも左にあり、検出された原点を通るx座標よりも上に存在する数値。

**y 軸の数値** 検出された原点を通るy座標よりも右にあり、検出された原点を通るx座標よりも下に存在する数値。

**凡例の文字列** 検出された4本の軸に囲まれた領域（グラフ領域）に存在する文字列。

具体例を図5.4に示す。

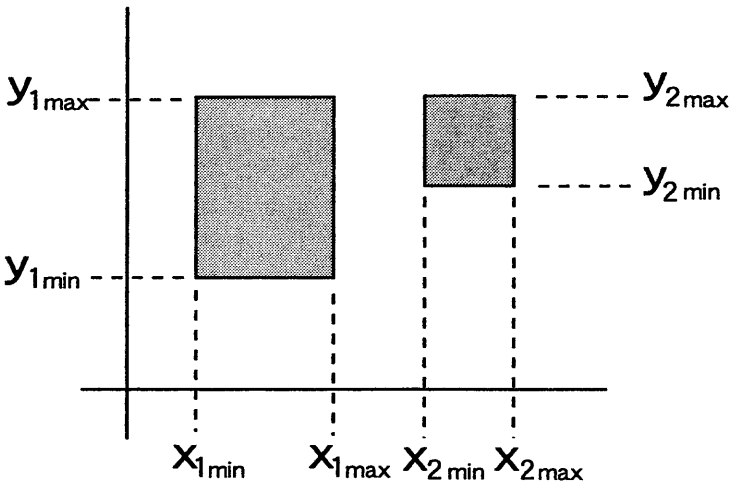



図 5.2: 文字領域の結合パターン 2

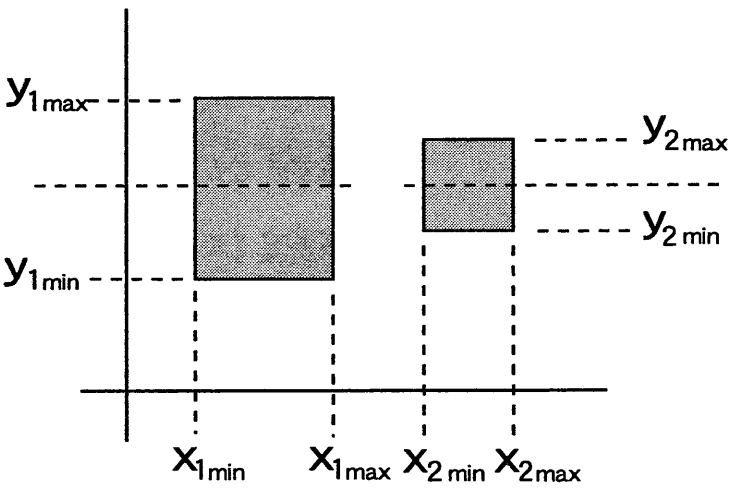



図 5.3: 文字領域の結合パターン 3

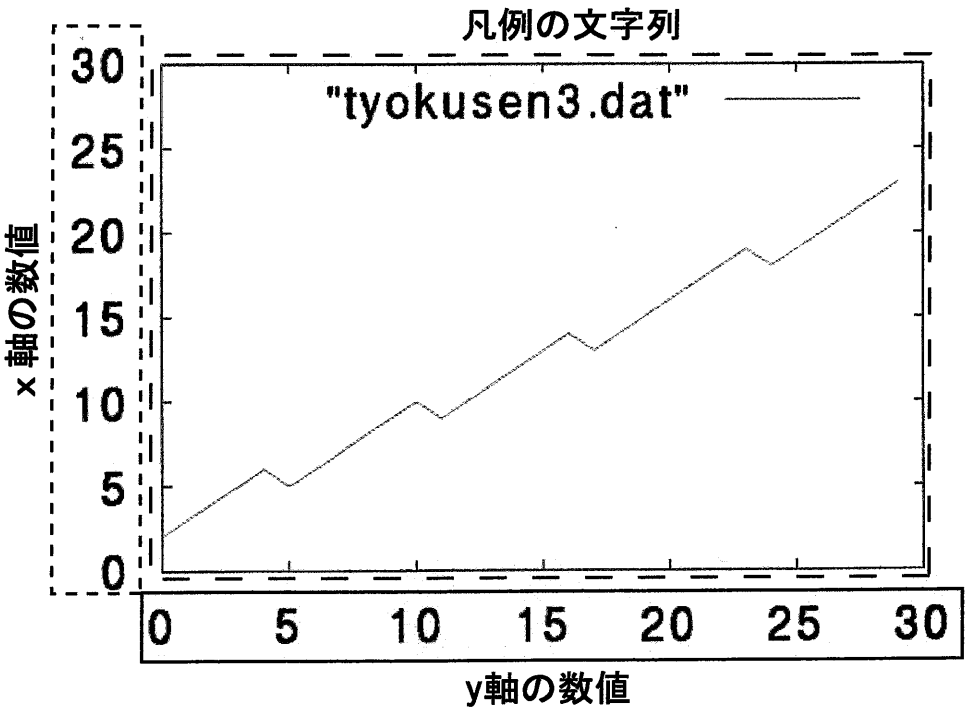



図 5.4: 文字列のグラフ要素のクラス

# 第6章

# グラフ画像と文字画像の統合

## 6.1 はじめに

本章では、第4章で述べたグラフ画像に対する処理結果と第5章で述べた文字画像に対する処理結果を統合して最終的なデータとして出力する方法について説明する。

## 6.2 軸の数値の読み替え

第4章で述べた処理の結果であるグラフの折れ点情報、第5章で述べた処理の結果である文字列の認識は既に終了しているので、最終的な処理としてグラフの折れ点情報の抽出文字列（数値）による軸の値の読み替えをして、最終的に出力されるグラフの折れ点情報とする。

以下では、代表してy軸の数値の読み替えについて説明する。

**step1** まず抽出されたy軸の数値に対応する文字列の右方向を調べ、その位置にy軸の目盛りが存在しているかをチェックする。

目盛りが存在するとは以下の条件を満たしたときをいう。

$y_{mid} = \frac{y_{max}+y_{min}}{2}$ とし、

$y_{mid} \pm a$ 上に目盛りが存在する。

$a$ は任意の定数。

**step2** step1でy軸に目盛りが存在していると認識された場合、入力画像におけるその位置のy座標を抽出された文字列の数値に読み替える。

**step3** step1 ～ step2を抽出された全ての文字列について実行する。

**step4** 読み替えられた軸の座標間隔から、入力グラフのスケール形式を判断し、第4章で得られたグラフの折れ点情報を読み替える。

図6.1に具体的なy軸の読み替え例を示した。

以上の処理を実行することによって、入力画像における座標を読み替えることが可能になり、出力グラフの折れ点情報を得ることが可能になる。

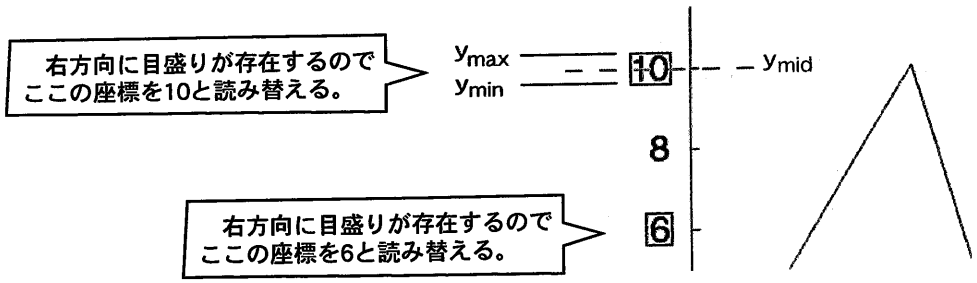



図 6.1: y 軸数値の読み替え例

また同様の手法によって、x 軸の数値読み替えに関しても処理することが可能である。但し x 軸の場合は、文字列の上方向の目盛りの存在を調べる必要がある。この場合の目盛りの存在条件は、以下のようになる。

$x_{mid} = \frac{x_{max}+x_{min}}{2}$ とし、
$x_{mid} \pm a$ 上に目盛りが存在する。
　$a$ は任意の定数。

# 第7章

# 入力画像が2値画像の場合

## 7.1 はじめに

本章では、入力画像が白黒、つまり2値の場合の認識手法について説明する。

## 7.2 認識手法

入力画像がカラーではない場合、グラフ認識をする過程でもっとも重要なことは、グラフ要素を識別することである。

入力画像がカラーの場合は通常、存在する折れ線が色で分けてあるため、認識過程ではその色を識別することによって存在する折れ線の数を知り、容易に折れ線の種類を識別することが可能であった。しかし、本章で扱う入力画像では色による折れ線の識別をすることが不可能であるため、存在する折れ線を構成する線分の形からのみ識別をする必要がある。

通常、2色のみで描かれたグラフというものは、折れ線を色で描き分けることが不可能なため、その描き分けを形によって行っている。具体的には、実線と点線のようにである。つまり、この形の違いが識別できれば、描かれている折れ線を識別することが可能になるということが言える。

識別の流れを以下に示す。

**step1** 入力画像の画素を広げることにより、点線を構成している点を連結させる。その後、グラフ画像と文字画像に入力画像を分離する。

**step2** 入力画像から分離したグラフ画像より軸を検出する。

**step3** 軸内をスキャンすることにより、グラフを構成している折れ線数を検出する。(図 7.1)

**step4** 画素を広げる以前の画像を用いて、軸内の画素に対して Hough 変換を実行し、直線の検出を行う。

**step5** 検出された直線の交点から折れ点を検出する。

**step6** 検出された端点間の直線に乗った画素のパターンからその直線が属する折れ線を識別する。(図 7.2)

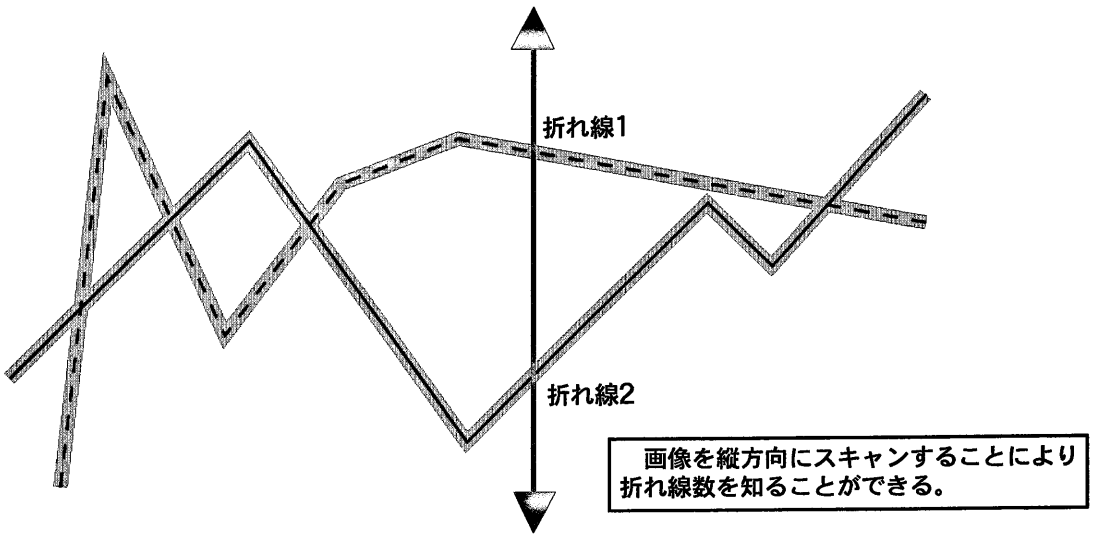



図 7.1: 折れ線数の検出

以上の処理により、グラフを構成している線分を識別することが可能になる。
折れ線の識別処理終了後は、カラーの入力画像のときと同様に処理を実行していく。
ここで、画素のパターンから直線が属する折れ線を識別する手法を以下に示す。

通常、点線は画素連結成分の組によって構成されている。よって、点線のパターンを識別するために、この連結成分の組のパターンを識別すれば良い。

**step1** 直線に乗った画素数を連結成分毎にカウントする。

**step2** step1 でカウントされた画素のパターンが、いずれの折れ線パターンに近いかを識別し、検出された直線が属する折れ点を識別する。

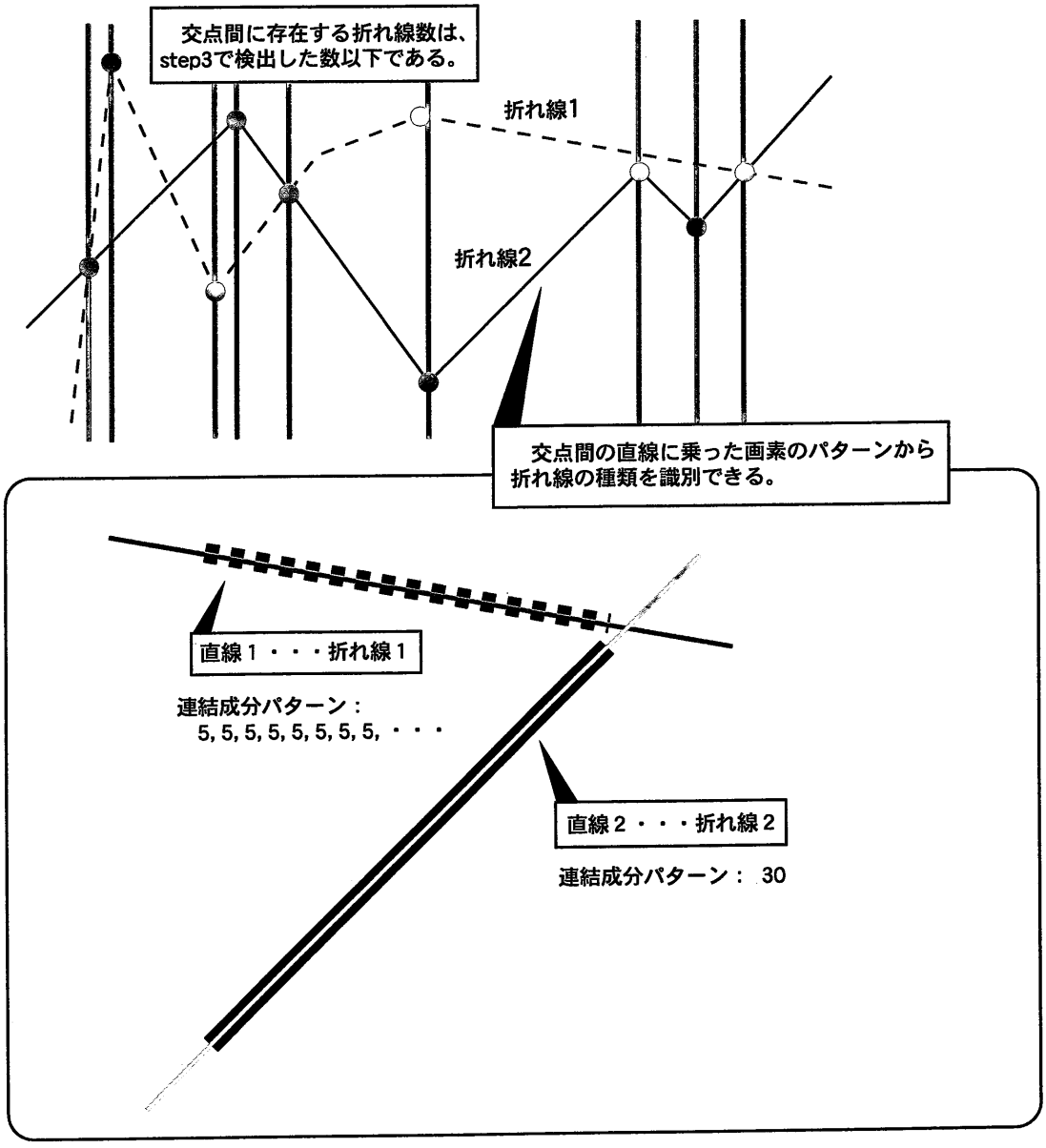



図 7.2: 直線上のパターンによる識別

# 第8章

# 認識実験

## 8.1 はじめに

本章では、Exeleによって作成したカラーグラフと実際に文書から切り抜いたカラーグラフを用いた認識実験について説明する。

## 8.2 認識実験

### 8.2.1 実験条件

Exeleによって作成したカラーの折れ線グラフと実際に文書から切り抜いたカラーの折れ線グラフをスキャナーによって取り込んだ画像（ppm画像）をテストサンプルとする。入力画像のサイズはテストサンプルによって異なるので、実験結果でサンプルの画像と共に示す。

また、認識処理における種々のパラメータの値を以下のようにした。但し、これらのパラメータの値は入力画像の大きさやグラフ領域の大きさに依存する値であるため、以下の値が常に適切であるわけではない。

**画素の拡張の大きさ:** 2pixel

**直線の統合範囲:** $\alpha = 20$ , $\beta = 6$

### 8.2.2 実験結果

図8.1～図8.8に実験結果の一例を示す。

ただし、入力画像3 (図8.5)に対しては軸の読み替え以前の認識結果、入力画像4 (図8.7)に対しては折れ線の識別状況を色で表示したものを示す。前者の理由は軸の読み替え処理で不具合が生じたためで、後者は折れ線の識別処理以降はカラーの入力画像に対しての処理と同様であるからである。前者で生じた不具合の理由については、考察で述べる。

実験結果より、本手法を用いた場合、ある程度の高い精度でカラーの折れ線グラフを認識できることがわかる。

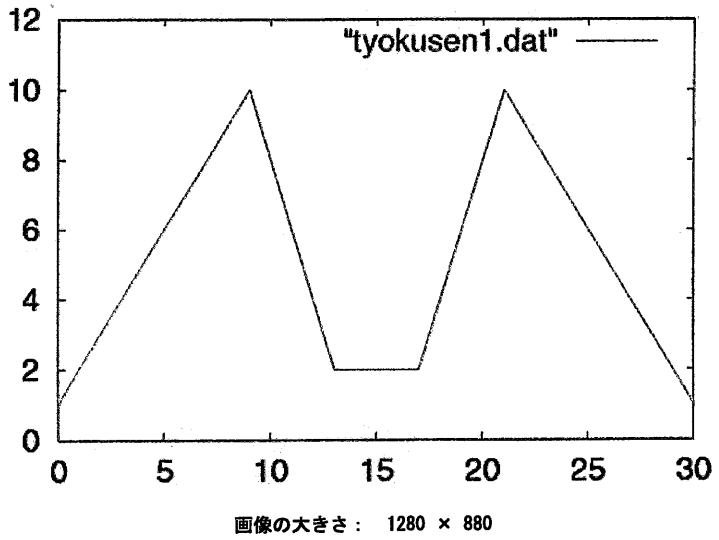



図 8.1: 入力画像 1

## 8.3　考察

本節では、入力画像 3 (図 8.5) における認識結果に対する考察を中心に誤認識の原因について考察する。

入力画像 3 (図 8.5) に対しては、前節で軸の読み替え以前の認識結果を示した。この理由が軸の読み替え処理における不具合が原因であることは、前節で述べた通りである。では、軸の読み替え処理における不具合とはなんであるのか。それは、文字列の配置に問題があったのではないかと考えられる。入力画像 3 (図 8.5) を見れば分かるように、このグラフでは軸に対するラベルが書かれている。このラベルに対して軸上の目盛りがマッチしたため、その座標をラベルに読み替えてしまったと考えられる。そのため、座標の読み替え処理が正常に実行されなかったのだろう。

また、入力画像 3 (図 8.5) とその認識結果 (図 8.6) を見比べると、折れ線の形状がしっかり認識されていないことがわかる。まず、入力画像では数本の線分によって描かれている折れ線部分が、結果では一本の線分で描かれてしまう理由であるが、これは第 4 章で述べた検出直線の統合範囲が原因であると考えられる。統合範囲を決定するパラメータの一つである $\beta$ の値が大きすぎたため、微妙に傾きの違う線分を同一の直線を形成している線分だと判断し統合してしまったのだろう。次に入力画像には存在しない折れ点が出力されている理由であるが、これは入力画像における線幅に大きく依存していると考えられる。入力画像において線幅が大きくなると、検出される直線が増加する。すると同時に検出される交点、つまり線分の端点が増加してしまうのである。しかも線幅が大きいため、端点が一直線上に乗っていなくても画素色が同一になってしまう。すると、端点の間引きによっても検出された端点を間引くことができずに、図 8.6 のような結果になってしまうものと考えられる。

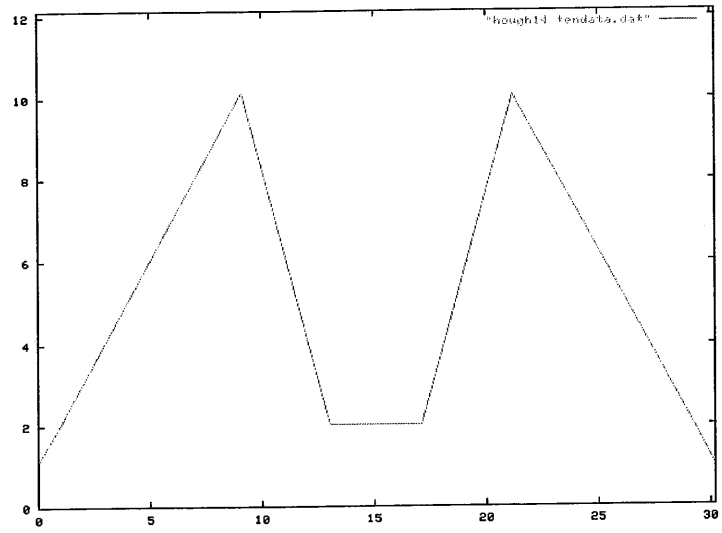



図 8.2: 入力画像 1 に対する認識結果

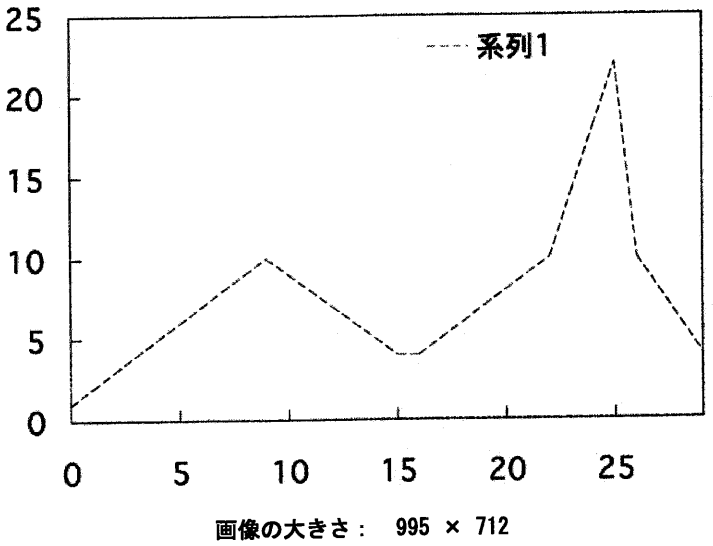



図 8.3: 入力画像 2

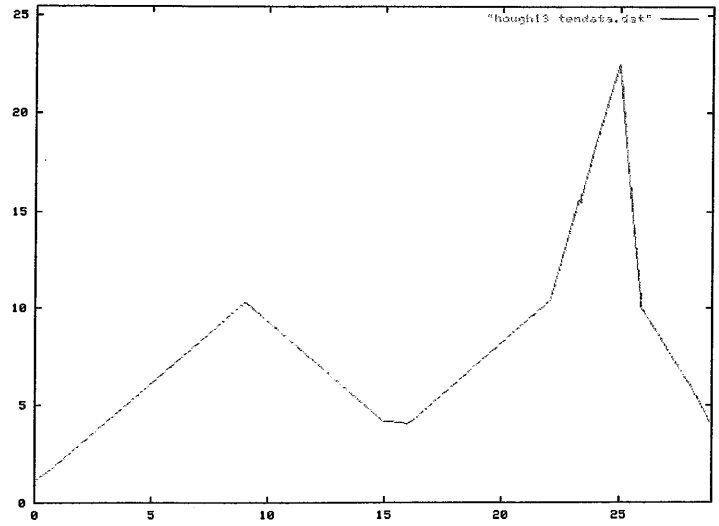
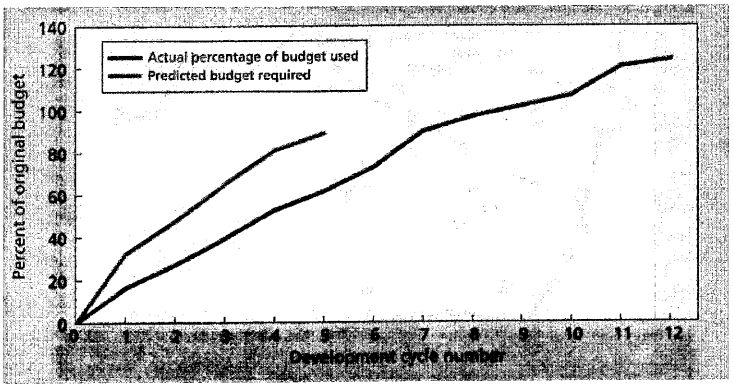



図 8.4: 入力画像 2 に対する認識結果




画像の大きさ：　1072 × 563

図 8.5: 入力画像 3

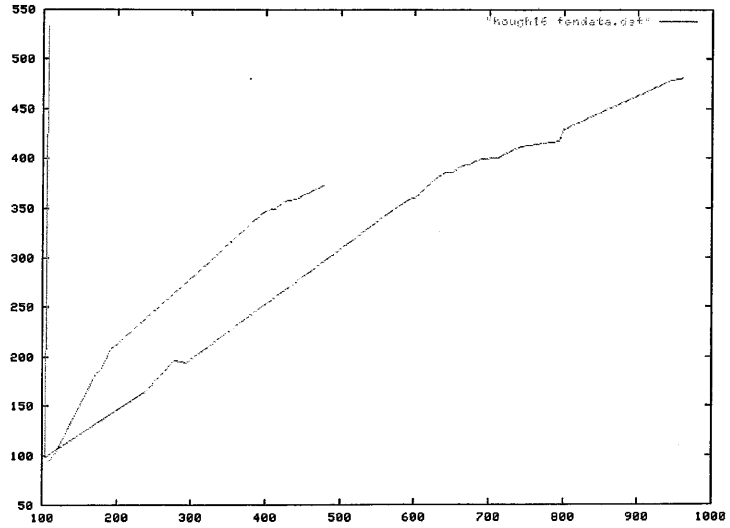



図 8.6: 入力画像 3 に対する認識結果

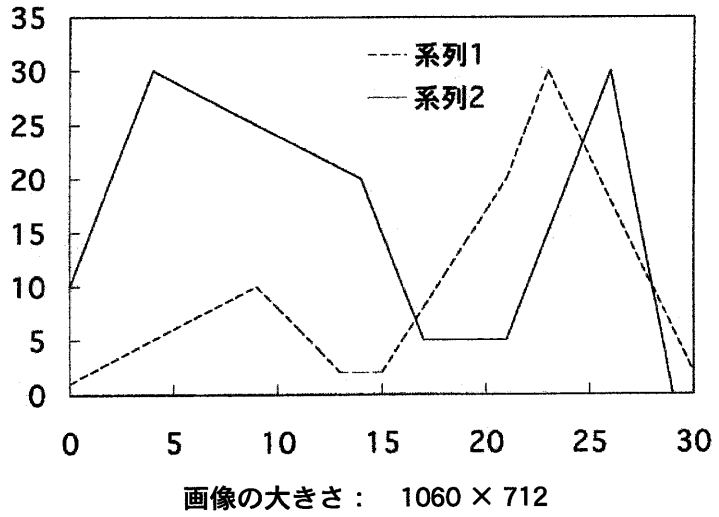



図 8.7: 入力画像 4

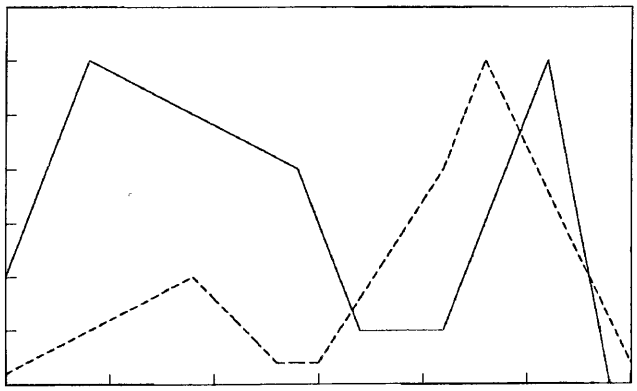


図 8.8: 入力画像 4 に対する認識結果

# 第9章

# 結論

## 9.1 本研究の成果

本研究では、Leeらの認識手法では考慮されていなかったマーカーの存在しない折れ線グラフ画像から適切な折れ点を検出し、個々のユーザーが任意に書き換えられるように折れ点のデータを出力する手法を提案した。また、提案手法を評価するための認識実験を行った。

その結果、ある程度の折れ線グラフについては正しく認識することを確認した。しかし、グラフを構成する線幅に依存した誤認識や文字列からの軸の読み替えにおける誤認識など、考慮すべき点もいくつか存在することが分かった。

## 9.2 今後の課題

今後は、任意の文書中の折れ線グラフ画像を入力することによって、より正確な折れ点データを出力できる手法を確立することが大きな課題として挙げられる。そのためには、以下の点について考慮、または再検討する必要がある。

- 前処理における色のクラスタリング手法を再検討し、任意の入力画像に対してより頑健な手法の検討。
- グラフを構成する折れ線の線幅を考慮した直線の統合手法と折れ点の検出手法の検討。
- 入力画像の大きさやグラフ画像の大きさから適切なパラメータの値を算出する手法の検討。
- 文字領域の誤認識や読み替え対象の誤認識に対して、誤認識を吸収できるような頑健な手法の検討。
- グラフ画像中の文字列をグラフ認識へ利用する検討。例えば、折れ点の座標を示している文字列など。

# 謝辞

本研究を進めるにあたり、多大な御指導とともに本研究の機会を与えてくださった東北大学工学部教授　阿曽弘具先生に深く感謝致します。

また、御指導、御意見を賜った東北大学工学部助教授　大町真一郎先生に心より感謝致します。

最後に、本研究に対し多岐にわたる御助言、御協力をいただき大変お世話になった阿曽研究室のみなさまに心より感謝致します。

# 参考文献

[1] N.Yokokura and T.Watanabe ：レイアウト構造知識を用いた棒グラフの認識, **情報処理学会論文誌** , vol.40 , No.7 , pp.2954 - 2966 (1999).

[2] Lee , M.H. Babaguchi , N. and Kitahashi , T. : Symbolization and Presentation of Graph Images for Intelligent Communication of Document Images , *Proc. ACCV'95* , Vol.3 , pp.680 - 684 (1995).