1D−10

フリーハンド罫線抽出アルゴリズムと手書き表認識システム

山本吉伸　　安西祐一郎

慶應義塾大学

1 はじめに

手書きの表をオフライン認識する際、罫線は記述者の意図を理解するための重要な情報となる。罫線は、データベースに入力する際にはデータのまとまりを意味し、ワードプロセッサへの取り込みの際には文字列の割り付け位置を規定する。

しかしながらフリーハンドで書かれた罫線を抽出する一般的な手法は明らかではない。定規で引かれた直線のための（CADで用いられるような）手法はそのままでは適用できない。

本稿では、みなし直線を高速に抽出するアルゴリズムを述べる。さらにこのアルゴリズムを用いた手書き表認識システムの実装と評価について報告する。

2 バブルモデルアルゴリズム

2.1 バブルモデル

バブルモデルアルゴリズムとは、ある1つのバブルが近傍にあるバブルと連結しながら成長（融合）してゆくことによって直線をベクトル化する手法である(図1)。ここでバブルとは、縦方向もしくは横方向に連なった1列のドットの集合である。

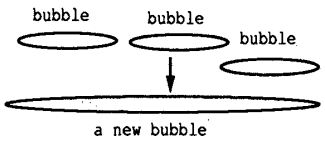

図1: バブルmodelによるベクトル化

バブルには縦方向(縦バブル)と横方向（横バブル）の2種類がある。表に使われる罫線は通常縦方向か横方向のみであるから、2種類のバブルについて考えればよい。処理開始時のバブル(以下プリミティブバブル＝PB）は、原画像から以下の手順で抽出される。(横バブルの場合)(1) 画像を左上から右方向にスキャンする。(2)PBとして設定した長さのドットの連なりがあれば、それをPBとする。(3) この処理を各行について実行する。

こうして抽出されたPBは、設定された範囲内にある同種類のバブルと融合する。つまり縦方向は縦方向のものだけで、横方向は横方向のものだけでバブルが融合しあう。

2つのバブルが融合した後の長さは、最長のものが保存される。位置に関しては2つのバブルの重心位置、すなわち長さの割合によって決定される。長く成長しているバブルは位置を変えにくく、短いバブルを併合してゆく。幅は融合に関係なく常に1である。これらの融合は疑似並列で実行される。

2.2 プリミティブバブル(PB)の決定法

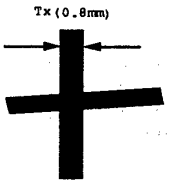

図2: PBの長さ決定手順

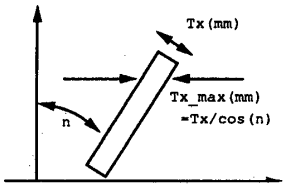

図3: n度傾いた時の縦罫線の最大幅

縦方向の線を無視し、横方向に伸びた直線のみを抽出するためには、画像データの中から横線だけを選択する条件でマッチングをとれば良い。図2は、縦と横に引かれた線の1部である。縦に引かれた線は現在傾いていないので、横PBの長さP_yが縦罫線の幅T_xに対して$P_y > T_x$であれば横バブルを生成しないで済む。

次に縦罫線がn度傾いている状況を考えてみる（図3)。このように傾いた状況の中でも縦罫線を無視するような横PBは

$$P_y > \frac{T_x}{\cos n}$$

という条件を満たしていれば良い。P_yを調節すればどの程度の傾きのずれに対応させるかを決定させることができる。イメージスキャナからの読み込み精度と、筆記具の太さを考慮してパラメータを決定する必要がある。

PBの長さの最大値は、入力される画像に依存する。すなわち、波打った手書き罫線を抽出するためには短い方が好ましいが、文字部分との分離や計算時間の面で劣ることになる。定規で引かれたような罫線を抽出する場合は、かなり長くPBをとることができる。

2.3 近傍範囲の設定法

バブルの抽出には図4に点線で示すような長方形の範囲(バブルの長さと比例定数a, bで決まる)を調べ、同種類のバブルが存在した時は融合を行なう。手書き罫線の傾きの許容範囲をこのパラメータで設定することができる。バブルが長くなると、融合範囲も広くなる。そこで1罫線あたり30ドット幅以上の探索をしないように上限を設定した。

Recognition Algorithm and System for Freehand-drawn Tables

Yoshinobu YAMAMOTO, Yuichiro ANZAI

Keio Univ.

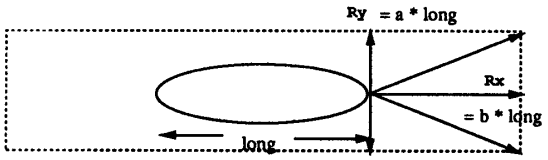

図 4: 探索範囲

2.4 バブルの融合

前節で定義された PB を並列に動作させる事によって、次第にベクトル化が進む。このアルゴリズムをまとめると次のようになる。(1)PB を抽出する。(2) 並列に各バブルを融合させる。(3) どのバブルにも変更がなくなれば終了する。そうでなければ (2) へ。(4) 罫線素片の長さを横軸に、素片の数を縦軸にしてグラフを作成する。すると、長くなった素片と短いままの素片がそれぞれ群を作っていることがわかる。そこで、短い方の罫線素片群を削除する。

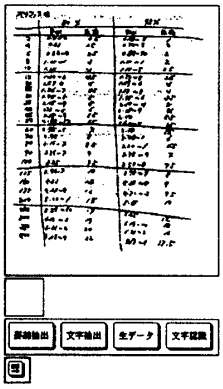

図 5: 入力データ

3 実装

このアルゴリズムを基本にして、手書き表認識システムを SONY NEWS(NWS-3800) 上に実装した。システムは以下のような手順を用いて清書した表を PostScript で出力する。

- 乱雑に書かれた表 (図 5) から、罫線素片を抽出する (図 6)。
- 抽出結果をユーザが評価し、もう 1 度やり直すかどうかを判定する。
- 方眼に罫線素片を重ね、罫線素片の含まれる矩形を塗りつぶす。補間が完全でなかった場合、方眼の目を荒くし補間されるまで繰り返す。この情報から罫線素片間の結線を行なう。
- 罫線素片の周辺 n ドットを罫線とみなし、データより削除 (図 7)。これにより、文字切り出しアルゴリズムの適用範囲を限定できる。この時、文字と罫線が隣接していた場合は多少文字が欠けることがある。
- 左上よりドットを調べ、文字を正方形の塊として切り出す[1]。図 7中、文字が黒くなっているのは切り出し済み文字。左下の小窓に文字の切り出し結果が表示される。
- 切り出した文字は Neural Network(back propagation) によって認識する。
- 罫線の抽出位置に直線をおき、認識した文字と罫線素片の位置関係に従って文字を再配置する。

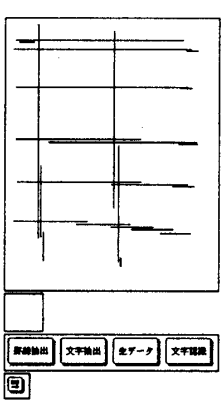

図 6: 罫線素片の抽出結果

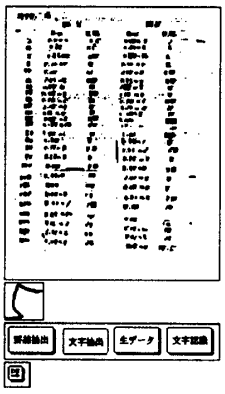

図 7: 罫線素片消去後、文字抽出の様子

4 評価および結論

機械工学科で実際に書かれた手書き表データ 20 枚を認識させたところ、異なる乱数の初期値で複数回試行することにより、すべてのデータについて罫線素片を抽出することができた。乱数の振り直しは平均 1.8 回必要であった。認識率をあげるためには表のフォーマットに合わせたパラメータの調整 (PB のサイズや融合の範囲等) が効果的であった。

また、極端に傾いた場合を除いて、罫線素片から罫線を再構成することができた。再構成された罫線によって、適切な文字列領域を切り出すことができる。

すでに触れた通り、現在の実装では罫線素片付近の文字は欠けてしまうため、文字の認識率は高いとはいえない。罫線素片から再構成した罫線の近辺をさらに探索して抽出精度を高めることも考えられる。

本アルゴリズムは、みなし罫線に対応できること、高速であること (Hough 変換による罫線素片抽出と比較して約 25 倍)、並列動作であること等が利点である。一方、垂直、水平方向の直線にしか有効でないこと、点線と実線の区別ができないこと、2 本線の認識が難しいこと等が欠点として考えられる。

参考文献

1. 岡本正之, 岡本浩之, ランレングス情報のみを用いた手書き図面の線認識, PRL85-93, pp.61–pp.68, 1985
2. 小島秀登, 木村茂, 小沢慎治, 線分構造抽出のための新しい手法の提案, PRU88-85, pp.5–pp.8, 1988
3. 清末悌之, 西村孝, 手書き線図形の自動浄書法に関する基礎検討, PRU87-73, pp.55–pp.62, 1987

[1] 文字がつながっていた場合については今後の課題である。