Matching techniques for mathematical symbol recognition

Scott MacLean
University of Waterloo

28 April, 2007

Contents

1 Introduction

Any application needing to interpret input captured via a pen-based interface must have a mechanism for determining what the user has written or drawn. In particular, the MathBrush project is an investigation into issues arising during the construction of pen-based interfaces for computer algebra systems and, as such, it requires a method for recognizing hand-drawn mathematics. The conversion of pen-based input into an internal representation of mathematical symbols is an important aspect of this recognition method.

Symbol recognition is a well-explored field and extensive literature exists [6]. We implemented five distinct recognition algorithms and performed experiments to determine which algorithms were best suited for recognition in the context of mathematical expressions. This report explains the requirements of mathematical symbol recognition, describes each algorithm and its implementation in terms of mechanism and complexity, and presents the results of our experiments along with analysis.

2 Mathematical symbol recognition

Symbol recognition in a mathematical context has some special requirements which do not apply to recognition of English text. Our particular requirements are as follows:

1. Since there is no finite set of valid mathematical expressions, our recognizer may not be dictionary-based.
2. Mathematics has a complex two-dimensional structure, including superscripts, subscripts, and symbols contained within bounding boxes of other symbols (eg. roots). Our recognizer must handle all permutations of this structure.
3. Such structure leads to wide variation in symbol sizes. For example, symbols in subscripts are generally smaller than symbols standing alone. Our recognizer must be insensitive to large variations in scale within a single expression.
4. We wish to allow users of MathBrush to include new symbols and add variations of existing symbols. Our recognizer must therefore be easily and quickly extendible.

To satisfy requirements (1) and (2), we apply isolated recognition techniques. That is, our recognition algorithms do not consider the context of the symbol being matched. The context is considered during the stroke grouping process described in [**?**]; however, this process is outside the scope of the present report.

To satisfy requirement (3), we normalize all model symbols and input, preserving the aspect ratio. In this way, symbols of varying size may be accurate matched with fixed-size models.

To satisfy requirement (4), we limit our attention to deterministic recognition techniques. Probabilistic techniques often require retraining after adding a new symbol.

3 Elastic matching

3.1 Overview

Elastic matching is a variant of "dynamic time warping", a technique used for matching in the signal processing field. Applied to pen input, this technique treats the sequence of x- and y-coordinates as two sampled signals and uses a distance metric to combine them. Our implementation follows [5].

Let the input stroke be the sequence of points $((x_1, y_1, \theta_1), (x_2, y_2, \theta_2), \ldots, (x_n, y_n, \theta_n))$ and the model stroke be the sequence $((u_1, v_1, \phi_1), (u_2, v_2, \phi_2), \ldots, (u_m, v_m, \phi_m))$, where θ_i is the angle of the vector tangent to the input stroke at (x_i, y_i), and ϕ_i is similarly defined for the model stroke.

We use the pointwise distance function

$$d(i, j) = \frac{360}{N}|y_i - v_j| + \min(|\theta_i - \phi_j|, 360 - |\theta_i - \phi_j|),$$

where N is the height of the input measured in stroke coordinates.

The algorithm finds a function M from input points to model points such that

$$\sum_{i=1}^{n} d(i, M(i))$$

is minimal over all M satisfying the following constraints:

1. $M(1) = 1$. (First point is matched to first point.)
2. $M(n) = m$. (Last point is matched to last point.)
3. If $M(i) = j$, then $M(i + 1) = j$ or $M(i + 1) = j + 1$ or $M(i + 1) = j + 2$. (No two consecutive model points may be skipped.)

Figure 1 illustrates such a matching.

In general, some points in a symbol convey more information about what the symbol represents than others. For example, the letters U and V have a similar shape but are distinguishable by the characteristic curve or point at the bottom. We extend the above matching process to include weights at each point and assign low weight to such disambiguating points. Our implementation of weighted matching is based on [4].

Weights are applied after the optimal match is determined; otherwise model points with low weights are matched with disproportionately many input points resulting in an inaccurate matching. Letting ω_i be the weight of model point i, the final match distance is

$$\sum_{i=1}^{n} \omega_{M(i)} d(i, M(i)).$$

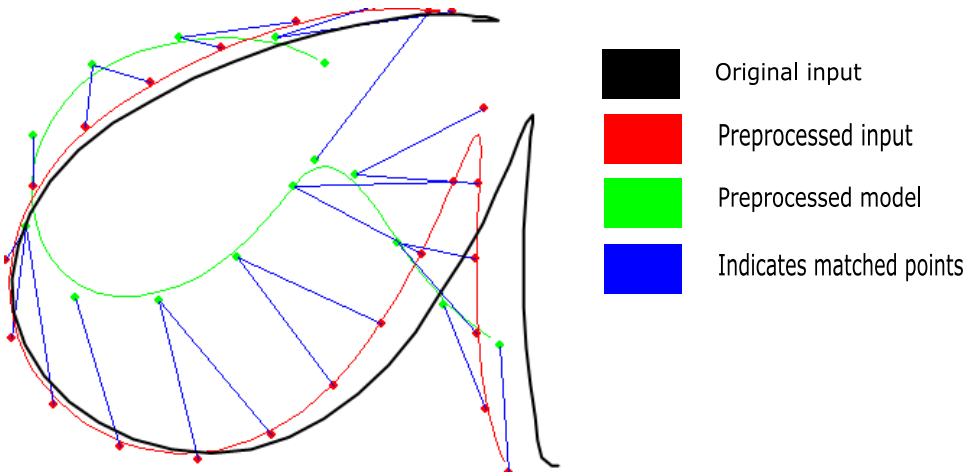

Figure 1: Elastic matching

Weights are determined using training data to measure the importance of each point in each database symbol. Each symbol is compared by unweighted elastic matching to every symbol in the training data. For any given symbol, two values are maintained for each point i: S_i is the average match distance observed at i when matching against symbols of the same class, and D_i is the average match distance observed at i when matching against symbols of a different class. After all data has been matched, point i is assigned weight $\frac{D_i}{S_i}$. Thus the farther a model point is from ambiguous points and the nearer it is to distinctive points, the higher the weight assigned to the point.

In our implementation, each model's weights are normalized to sum to 1. Without this step, symbols with many low weights are assigned artificially low match distances.

3.2 Algorithm

We apply a strokewise dynamic programming algorithm. Let $D[i, j]$ represent the minimal cumulative distance for matching points 1 through i of the input stroke to points 1 through j of the model. We define D by

$$
\begin{aligned}
D[1,1] &= d(1,1) \\
D[1,j] &= \infty \text{ if } j > 1 \\
D[i,1] &= d(i,1) + D[i-1,1] \text{ if } i > 1 \\
D[2,j] &= d(2,j) + \min\left\{D[1,j], D[1,j-1]\right\} \text{ if } j > 1 \\
D[2,2] &= d(2,2) + D[1,1] \\
D[2,3] &= d(2,3) + D[1,1] \\
D[2,j] &= \infty \text{ if } j > 3 \\
D[i,j] &= d(i,j) + \min\left\{D[i-1,j], D[i-1,j-1], D[i-1.j-2]\right\} \text{ if } i > 1, j > 2
\end{aligned}
$$

Here, the first two equations enforce constraint (1) above; the remainder enforce constraint

(3); constraint (2) is enforced by taking $D[n, m]$ to be the final match distance, where n and m are the number of points in the input and model strokes, respectively.

Let $n(s)$ be a function giving the number of points in stroke s. The following pseudocode describes an idealized version of algorithm for matching stroke groups, including weighting.

function ElasticMatchStroke ($s, m,$?weights)
// s is the input stroke; m is the model stroke; weights is an optional parameter
// D is cumulative distance described above;
// track$[i, j]$ records the model point matched with input point $i-1$ prior to matching model point j
// M is the matching found

$D[1,1] = d(1,1)$
track$[1,1] = 0$
for $j = 2$ **to** $n(m)$
    $D[1, j] = \infty$
    track$[1, j] = 0$
end for
for $i = 2$ **to** $n(s)$
    $D[i, 1] = d(i, 1) + D[i-1, 1]$
    track$[i, 1] = 1$
end for

$D[2,2] = d(2,2) + D[1,1]$
track$[2,2] = 1$
$D[3,2] = d(3,2) + D[1,1]$
track$[3,2] = 1$
for $j = 3$ **to** $n(m)$
    $D[2, j] = \infty$
    track$[2, j] = 0$
end for

for $i = 3$ **to** $n(s)$
    $D[i, 2] = d(i, 2) + \min_{j=1,2} \{D[i-1, j]\}$
    track$[i, 2] = \text{argmin}_{j=1,2} \{D[i-1, j]\}$
end for

for $i = 3$ **to** $n(s)$
    for $j = 3$ **to** $n(m)$
        $D[i, j] = d(i, j) + \min_{k=0,1,2} \{D[i-1, j-k]\}$
        track$[i, j] = \text{argmin}_{k=0,1,2} \{D[i-1, j-k]\}$
    end for
end for

$M[n(s)] = n(m)$
for $i = n(s)$ **to** 2 **by** -1

```
        M[i − 1] = track[i, M[i]]
    end for

    cost := 0
    if (weights) then
        for i = 1 to n(s)
            cost = cost + weights[i] × d(i, M[i])
        end for
    else
        cost = D[n(s),n(m)] / n(s)
    end if
end function


function ElasticMatch (s_1, ... s_N, m_1, ... m_N)
    // s_i are input strokes; m_i are model strokes
    cost := 0
    for i = 1 to N
        cost = cost + ElasticMatchStroke (s_i, m_i)
    end for
    cost = cost / N
    return cost
end function
```

3.3 Complexity

ElasticMatchStroke fills in two tables of size $n(s) \times n(m)$ for each pair of corresponding strokes s and m. Each table element takes $O(1)$ time to compute. It is clear that filling these tables dominates the cost of the algorithm since both filling M and the possible weighted summation of the match cost require time linear in $n(s)$. ElasticMatch calls ElasticMatchStroke for N pairs of strokes, so the total runtime is $O(NSM)$, where $S = \max_{i=1,\ldots,N}\{n(s_i)\}$ and $M = \max_{i=1,\ldots,N}\{n(m_i)\}$.

The algorithm also requires roughly $O(SM)$ space.

4 Deformable template matching

4.1 Overview

Deformable template matching is a variation of the well-known Expectation Maximization algorithm adapted for symbol recognition. It applies deformations to model symbols so that they more closely resemble input symbols. Costs are associated with these deformations

as well as with other operations, yielding a total match cost between the model and input symbols.

Our implementation of this method is based on [3]. Each model point is initially assigned a circular normal distribution with uniform variance σ. Matching proceeds iteratively from $i = 0$ as follows either until convergence or some predetermined maximum number of iterations.

Let $S = ((x_1, y_1), \ldots, (x_n, y_n))$ be the input stroke and $M_0 = ((u_1, v_1), \ldots, (u_m, v_m)$ be the model stroke.

1. Compute the probability $\mathrm{P}(S|M_i)$ that the input symbol symbol is generated by the model. The *fit cost* is the negative log of this probability.
2. Determine the contribution

$$c_{i,j} = \frac{\mathrm{N}\left((x_i, y_i); (u_j, v_j), \sigma\right)}{\sum_{k=1}^{m} \mathrm{N}\left((x_i, y_i); (u_k, v_k), \sigma\right)}$$

   of each model point j to each input point i's probability of being generated.
3. Generate the next iteration's model M_{i+1} by moving each model point j to

$$\sum_{i=1}^{n} c_{i,j}(x_i, y_i);$$

   that is, the sum of all input points, weighted by the model point's contribution to generating them. This process draws each model point closer to input points it has high likelihood of generating. The *deformation cost* is the negative log of the probability $\mathrm{P}(M_{i+1}|M_i)$ that M_i generates the new model.
4. Find model points having no significant contribution to any input points. Assign a *whitespace cost* to these points.
5. Subdivide the model M_{i+1} and reduce σ. Repeat from (1).

Figure 2 illustrates this deformation process.

4.2 Algorithm

As with elastic matching, we break the symbol matching process into a number of stroke matches, tallying the total match cost. Let $n(s)$ be a function giving the number of points in stroke s. The pseudocode below describes our implementation. Note that this description assumes that $n(s) = n(m)$ initially; in practice we ensure this is the case by resampling the input so that it has the same number of points as the model.

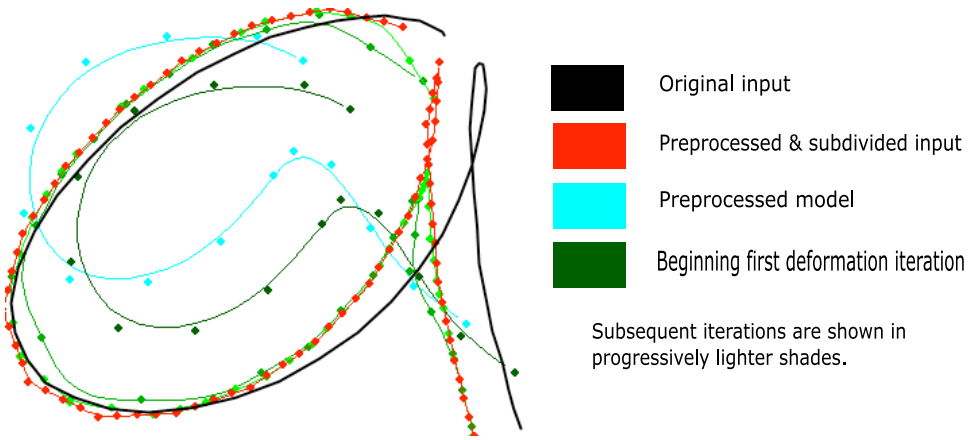

Figure 2: Deformable template matching

function DEFORMATIONMATCHSTROKE (s, m)

// $s = ((x_1, y_1), \ldots, (x_n, y_n))$ is the input stroke
// $m = ((u_1, v_1), \ldots, (u_m, v_m))$ is the model stroke
// σ is the initial variance; ε is the convergence threshold
// T is the maximum allowed iterations
// $c[i, j]$ is the contribution of model point j for generating input point i

cost := 0
prevCost := 0
for $k = 1$ **to** T

fitCost := $-\frac{1}{n(s)} \sum_{i=1}^{n(s)} \frac{1}{n(m)} \sum_{j=1}^{n(m)} \log \mathrm{N}\left((x_i, y_i); (u_j, v_j); \sigma\right)$

for $i = 1$ **to** $n(s)$
for $j = 1$ **to** $n(m)$
$c[i, j] = \frac{\mathrm{N}((x_i, y_i); (u_j, v_j), \sigma))}{\sum_{k=1}^{m} \mathrm{N}((x_i, y_i); (u_k, v_k), \sigma))}$
end for
end for

for $j = 1$ **to** $n(m)$
$(u'_j, v'_j) = \sum_{i=1}^{n(s)} c[i, j] \times (x_i, y_i)$
end for
deformationCost := $-\frac{1}{n(m)} \sum_{i=1}^{n(m)} \frac{1}{n(m)} \sum_{j=1}^{n(m)} \log \mathrm{N}\left((u'_i, v'_i); (u_j, v_j); \sigma\right)$
for $j = 1$ **to** $n(m)$
$(u_j, v_j) = (u'_j, v'_j)$
end for

whitespaceCost := $-\frac{1}{n(m)} \sum_{j=1}^{n(m)} \frac{1}{n(s)} \sum_{i=1}^{n(s)} \log c[i, j]$

iterationCost := fitCost + deformationCost + whitespaceCost
cost = cost + iterationCost

```
        if (|iterationCost − prevCost| < ε) then
            break
        end if
        prevCost = iterationCost
        σ = σ/2
        s = SUBDIVIDE (s)
        m = SUBDIVIDE (m)
    end for

    cost = cost/k
    return cost
end function


function DEFORMATIONMATCH (s_1, ... s_N, m_1, ... m_N)
    // s_i are input strokes; m_i are model strokes
    cost := 0
    for i = 1 to N
        cost = cost + DEFORMATIONMATCHSTROKE (σ, s_i, m_i)
    end for
    cost = cost/N
    return cost
end function
```

4.3 Complexity

The time taken by DEFORMATIONMATCHSTROKE is dominated by the loops and summations over all input and model points. The body of each of these loops executes in $O(1)$ time, so DEFORMATIONMATCHSTROKE runs in time $O(Mn(s)n(m))$. DEFORMATIONMATCH therefore runs in time $O(NTSM)$, where $S = \max_{i=1,\ldots,N}\{n(s_i)\}$ and $M = \max_{i=1,\ldots,N}\{n(m_i)\}$.

The algorithm also requires roughly $O(SM)$ space.

5 Structural chain-code matching

5.1 Overview

Structural matching treats the recognition problem as a string matching problem. Our implementation is based on [1], although we use only the directional codes and deformation

ideas presented there and not the syntactic stroke grammar they introduce, as they found it led to ambiguity during matching.

In structural matching, each stroke is converted to a *chain code* by assigning each pair of consecutive points a *direction code* representing the stroke's direction between the points; the chain code is the ordered sequence of these values. We used eight direction codes. Figure 3 illustrates this conversion process.

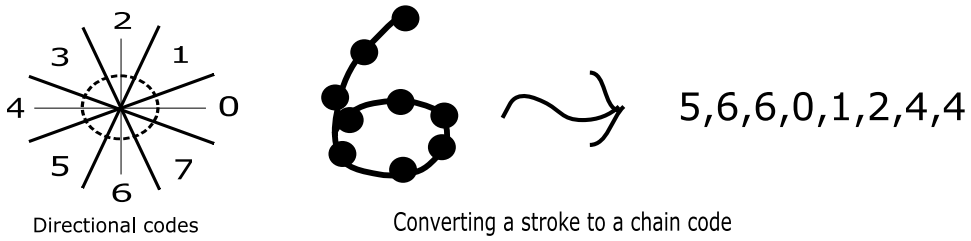

Figure 3: Chain coding process

We assume for simplicity that the input and model strokes have the same number of points; in practice, as with deformable template matching, input strokes are resampled prior to matching. During matching, we allow for slight rotation of symbols as well as extra or missing data at the ends of strokes by adjusting the chain codes. Letting R be the maximum allowable rotation measured in direction codes and S the maximum allowable number of extra or missing points, the following formula gives the match score for two chain codes $a_1 a_2 \ldots a_k$ and $b_1 b_2 \ldots b_k$:

$$\min_{s \in \{-S,-S+1,\ldots,S\}} \left\{ \min_{p \in \{-P,-P+1,\ldots,P\}} \sum_{i=1}^{k-|s|} \min\left(|a_{i+s} + p - b_i|, 8 - |a_{i+s} + p - b_i| \right) \right\}$$

In our implementation, P and S are both 1.

Unlike other algorithms, chain code matching does not match only a single stroke at a time. Since only intra-stroke directions are measured, we insert invisible strokes into the input to capture the direction traveled in between pen-up and pen-down. The distance traveled is not captured, however; this is a limitation of the chain coding approach.

5.2 Algorithm

Let $n(s)$ be a function giving the number of points in stroke s. Let $x(s,i)$, $y(s,i)$ be functions giving the x- and y-coordinate of point i in stroke s, and let $\theta(s,i)$ be a function giving the angle of the line segment between points i and $i+1$ of stroke s. The pseudocode below describes our implementation:

function ExtractChainCode $(s_1, \ldots s_N)$
    // s_i are strokes
    // D is the number of direction codes
    chainCode := ()
    for $i = 1$ **to** N
        for $j = 1$ **to** $n(s_i) - 1$
            direction $= \left\lfloor \frac{\theta(s_i, j)}{D} \right\rfloor$
            if (direction < 0) **then**
                direction = direction $+ D$
            end if
            chainCode = (chainCode, direction)
        end for

        if $(i < N)$ **then**
            $\Delta x = x(s_{i+1}, 1)) - x(s_i, n(s_i))$
            $\Delta y = y(s_{i+1}, 1)) - y(s_i, n(s_i))$
            direction $= \left\lfloor \frac{\arctan \frac{\Delta y}{\Delta x}}{\frac{2\pi}{D}} \right\rfloor$
            if (direction < 0) **then**
                direction = direction $+ D$
            end if
            chainCode = (chainCode, direction)
        end if
    end for

    return chainCode
end function

function ChainCodeMatch $(s_1, \ldots s_N, m_1, \ldots m_N)$
    // s_i are input strokes; m_i are model strokes
    // R is the maximum allowable rotation
    // S is the maximum number of extra or missing points
    input := ExtractChainCode $(s_1, \ldots, s_N)$
    model := ExtractChainCode $(m_1, \ldots, m_N)$
    k := Length (input)
    cost := ∞
    for $s = -S$ **to** S
        shiftedCost := ∞
        for $r = -R$ **to** R
            rotatedCost := $\sum_{i=1}^{k-|s|} \min \left(\left| \text{model}[i+s] + p - \text{input}[i] \right|, 8 - \left| \text{model}[i+s] + p - \text{input}[i] \right| \right)$
            shiftedCost = min (shiftedCost, rotatedCost)
        end for
        cost = min (cost, shiftedCost)
    end for
    return cost
end function

5.3 Complexity

Let $P = \sum_{i=1}^{N} n(s_i)$ be the total number of points in the input (and model, due to resampling). Chain code extraction requires $O(1)$ operations per point, totaling $O(P)$. Matching chain codes takes $O(SRP)$ time; this step clearly dominates the cost. Note that in practice S and R are fixed to 1, so this runtime is essentially linear in the number of input points.

The algorithm also requires $O(P)$ space.

6 Stroke feature matching

6.1 Overview

Feature-based matching is a general matching approach. It extracts salient features from its input to create feature vectors which can be compared by a vector norm. Our implementation extracts the following features for each stroke:

- topmost coordinate,
- leftmost coordinate,
- stroke width and height,
- first and last points, and
- total arclength.

Other features we have experimented with include displacement between first and last points, width-to-height ratio of bounding box, and estimated pen acceleration integrated along the stroke.

To match symbols, the algorithm extracts feature vectors from each pair of corresponding input and model strokes, then takes the 1-norm of their difference. The average of these norms over all pairs of strokes is returned as the match score. The algorithm is very simple, so pseudocode is omitted.

6.2 Complexity

Each of the extracted features can be computed in constant time with the exception of arclength, which requires $O(n(s))$ time, where s is a stroke and $n(s)$ is a function giving the

number of points in s. Taking a vector norm is a constant-time operation since the size of feature vectors is fixed. The algorithm's runtime is hence $O(NS)$, where N is the number of strokes in the input and S is the maximum number of points in any input or model stroke. The algorithm requires only $O(1)$ space.

7 Direction element method

7.1 Overview

The direction element method is a feature-based approach that focuses on stroke direction information. To extract the feature vector from a symbol, the symbol's bounding box is subdivided into an $n \times m$ grid of cells; each cell is associated with a d-element feature vector representing d directions $\phi_1, \ldots, \phi_d$. For each pair of consecutive points in the symbol, the angle θ of the line segment connecting the points lies between ϕ_i and ϕ_{i+1} for some i. Let $\theta_i = \theta - \phi_i$ and $\theta_{i+1} = \phi_{i+1} - \theta$. The contributions of that line segment to the directions ϕ_i and ϕ_{i+1}, respectively, are $\frac{L\theta_{i+1}}{\theta_i + \theta_{i+1}}$ and $\frac{L\theta_i}{\theta_i + \theta_{i+1}}$, where L is the length of the line segment.

This scheme gives larger weight to long line segments and accurately discretizes the input's angular components by applying the contribution weights. By summing all contributions over all pairs of consecutive points in the input, the algorithm computes a nmd-element feature vector. The vectors of input and model symbols are compared with a norm to obtain a match score. Our implementation follows the approach of [2] but adapts the grid size and number of directions based on the shape of the input symbols. We use a default of 10 directions.

7.2 Algorithm

Let $n(s)$ be a function giving the number of points in stroke s. Let $x(s, i)$, $y(s, i)$ be functions giving the x- and y-coordinate of point i in stroke s, and let $\theta(s, i)$ be a function giving the angle of the line segment between points i and $i + 1$ of stroke s. The pseudocode below describes our implementation. We assume here for simplicity that stroke coordinates are normalized to the range $[0, 1]$.

function DIRECTIONELEMENTFEATURES (xcells, ycells, directions, $s_1, \ldots, s_N$)
    // $F[i, j, k]$ is the feature vector entry for grid cell (i, j), direction index k
    dSpan := $\frac{2\pi}{\text{directions}}$
    for $i = 1$ **to** N
        for $j = 1$ **to** $n(s_i) - 1$
            cellX := $\lfloor x(s_i, j) \times \text{xcells} \rfloor$
            cellY := $\lfloor y(s_i, j) \times \text{ycells} \rfloor$
            cellD := $\lfloor \frac{\theta(s_i, j)}{\text{dSpan}} \rfloor$

$L := \sqrt{\| \big(x(s_i, j), y(s_i, j)\big) - \big(x(s_i, j+1), y(s_i, j+1)\big) \|}$

$\Delta\theta_1 := \theta(s_i, j) - \text{dSpan} \times \text{cellD}$
$\Delta\theta_2 := \text{dSpace} \times \big(\text{cellD} + 1\big) - \theta(s_i, j)$
$C_1 := \frac{L\Delta\theta_2}{\Delta\theta_1 + \Delta\theta_2}$
$C_2 := \frac{L\Delta\theta_1}{\Delta\theta_1 + \Delta\theta_2}$

$F[\text{cellX}, \text{cellY}, \text{cellD}] = F[\text{cellX}, \text{cellY}, \text{cellD}] + C_1$
if (dCell = directions − 1) **then**
$F[\text{cellX}, \text{cellY}, 0] = F[\text{cellX}, \text{cellY}, 0] + C_2$
else
$F[\text{cellX}, \text{cellY}, \text{cellD} + 1] = F[\text{cellX}, \text{cellY}, \text{cellD} + 1] + C_1$
end if
end for
end for
return F
end function

function DirectionElementMatch $(s_1, \ldots, s_N, m_1, \ldots, m_N)$
// s_i are input strokes; m_i are model strokes
// Directions is the base number of directions
// MinDirections is the minimum number of directions
// XCells is the base number of cells in the x-direction
// YCells is the base number of cells in the y-direction
// MaxCells is the maximum allowed number of cells in either direction
xcells := XCells
ycells := YCells
directions := Directions
bounds := BoundingBox $(s_1, \ldots, s_N)$
width := Width (bounds)
height := Height (bounds)
if (width > height) **then**
xcells = min $\big(\text{MaxCells}, \lfloor \text{ycells} \times \frac{\text{width}}{\text{height}} \rfloor\big)$
directions = max $\big(\text{MinDirections}, \lfloor \frac{\text{width}}{\text{height}} \rfloor - 1\big)$
else
ycells = min $\big(\text{MaxCells}, \lfloor \text{xcells} \times \frac{\text{height}}{\text{width}} \rfloor\big)$
directions = max $\big(\text{MinDirections}, \lfloor \frac{\text{height}}{\text{width}} \rfloor - 1\big)$
end if

input := DirectionElementFeatures (xcells, ycells, directions, $s_1, \ldots, s_N$)
model := DirectionElementFeatures (xcells, ycells, directions, $m_1, \ldots, m_N$)

$\text{cost} := \frac{\|\text{input} - \text{model}\|}{\sum_{i=1}^{N} n(s_i)}$

return cost
end function

7.3 Complexity

DirectionElementFeatures applies constant time operations to each pair of consecutive points in the input and model symbols. Letting $P = \sum_{i=1}^{N} n(s_i)$ and $Q = \sum_{i=1}^{N} n(m_i)$, this requires $O(P+Q)$ time. Computing the norm of the two feature vectors takes $O(\text{MaxCells}^2 \times \text{Directions})$ time. Determining the adaptive grid sizing takes $O(1)$ time. The total runtime is thus $O(P + Q + \text{MaxCells}^2 \times \text{Directions})$.

The algorithm also requires $O(\text{MaxCells}^2 \times \text{Directions})$ space to store feature vectors.

8 Experimental results

Matching algorithms were evaluated with two data sets. The first set, A, consists of 10 samples of each symbol written in isolation by a single user. This set was split into A1 and A2 so that one half could be used for training and the other for testing. Accuracy rates on A indicate the baseline performance of a matcher under near-ideal conditions.

The second set, B, consists of a number of mathematical expressions written by multiple writers. The training set for these tests is the symbol database used in practice, containing many symbols from set A as well as other isolated symbols collected from several users. Accuracy rates on B give a more realistic estimate of performance under real-world conditions.

The tables in Appendix A indicates the accuracy of the various matching techniques. The time reported is wall-clock time in seconds.

In practice, the set of symbols to match against is pruned to remove unlikely candidates. This process speeds up recognition dramatically and often improves accuracy as well. The tables in Appendix B present test results when pruning is enabled.

We can draw some clear conclusions from these results:

1. Pruning is a valuable technique. Not only does it does it make recognition three to four times faster, but in most cases it also improves accuracy.
2. Elastic matching is the only technique we implemented which comes close to the accuracy required for real-world use. The "Top 1" accuracy rates of all the other techniques are very poor and even the "Top 5" accuracy is generally worse than the "Top 1" rate of elastic matching.

3. There is virtually no difference in terms of either accuracy or time between weighted and unweighted elastic matching.

4. When pruning is disabled, the stroke feature matcher ranks second in accuracy and first in execution time.

The only surprising conclusion is (4). That so simple a comparison method would perform so well was unexpected. Since the stroke feature matcher is so fast, we use it in practice as our method of pruning the symbol set: any symbols with a sufficiently poor match score are discarded. (For this reason, the stroke feature matcher's accuracy results are identical both with and without pruning).

When pruning is enabled, most of the recognizers perform well in the sense of including the correct candidate in the top five, with the notable exception of the deformable template matcher. This matcher's poor performance is not unexpected, as as in [3] the matcher could not attain zero errors even on a hand-filtered database of only two symbols.

Elastic matching clearly outperforms the other matchers in terms of accuracy. In our system, we use unweighted elastic matching since it has similar accuracy to the weighted variant, but the weights require retraining when new symbols are added to the database to maintain their consistency.

9 Conclusions

We implemented five matching techniques for the purpose of mathematical symbol recognition and performed experiments measuring their accuracy and execution time. Algorithms implemented include: elastic matching (weighted and unweighted), deformable template matching, structural chain code matching, stroke feature matching, and the direction element method.

Elastic matching was found the be the most accurate technique, while stroke feature matching was the fastest. In our recognition system, we therefore stroke feature matching to prune the set of symbols to match against and elastic matching to perform actual symbol matching.

References

1. K-F. Chan and D-Y Yeung, Recognizing on-line handwritten alphanumeric characters through flexible structural matching, Pattern Recognition, 32(7), pp. 1099-1114 (1999).

2. T. Kanahori et al. On-Line Recognition of Mathematical Expressions Using Automatic Rewriting Method. Advances in Multimodal Interfaces - ICMI2000, Lecture Notes in Computer Science 1948, Springer (2000) 394-401.

3. M. Revow, C. Williams and G. Hinton, Using generative models for handwritten digit recognition, IEEE Transactions Pattern Analysis and Machine Intelligence 18(6), pp. 592-606 (1996).

4. P. Scattolin, Recognition of Handwritten Numerals Using Elastic Matching. Master's thesis, Computer Science Department, Concordia University Montreal (1993).

5. C.C. Tappert. Cursive Script Recognition by Elastic Matching, IBM Journal of Research and Development 26(6) pp. 765-771 (1982).

6. C.C. Tappert, C.Y. Suen, and T. Wakahara, The state of the art in on-line handwriting recognition, IEEE Transactions Pattern Analysis and Machine Intelligence 12(8), pp. 787-808 (1990).

A Results without pruning

These tables contain results of testing with database pruning disabled.

Test set	Num. symbols	Top 1		Top 5		Time
		Raw count	%	Raw count	%	
A1	625	580	92.8	625	100	113.96
A2	625	576	92.16	624	99.84	95.55
B	724	642	88.67	712	98.34	310.72

Table 1: Elastic matching (unweighted)

Test set	Num. symbols	Top 1		Top 5		Time
		Raw count	%	Raw count	%	
A1	625	580	92.8	625	100	113.57
A2	625	576	92.16	624	99.84	103.24
B	724	638	88.67	717	98.34	314.14

Table 2: Elastic matching (weighted)

Test set	Num. symbols	Top 1		Top 5		Time
		Raw count	%	Raw count	%	
A1	625	143	22.88	363	58.08	278.99
A2	625	157	25.12	382	61.12	254.13
B	724	129	17.82	396	54.97	701.65

Table 3: Deformable template matching

Test set	Num. symbols	Top 1		Top 5		Time
		Raw count	%	Raw count	%	
A1	625	344	55.04	519	83.04	21.61
A2	625	344	55.04	518	82.88	18.25
B	724	383	52.9	645	89.09	91.65

Table 4: Chain code matching

Test set	Num. symbols	Top 1		Top 5		Time
		Raw count	%	Raw count	%	
A1	625	428	68.48	590	94.4	10.91
A2	625	426	68.16	587	93.92	8.81
B	724	401	55.39	609	84.12	41.30

Table 5: Stroke feature matching

Test set	Num. symbols	Top 1		Top 5		Time
		Raw count	%	Raw count	%	
A1	625	335	53.6	543	86.88	16.76
A2	625	309	49.44	540	86.4	14.17
B	724	338	46.69	604	83.43	63.14

Table 6: Direction element method

B Results with pruning

These tables contain results of testing with database pruning enabled.

Test set	Num. symbols	Top 1		Top 5		Time
		Raw count	%	Raw count	%	
A1	625	580	92.8	625	100	34.06
A2	625	576	92.16	624	99.84	27.82
B	724	643	88.81	712	98.34	97.4

Table 7: Elastic matching (unweighted)

Test set	Num. symbols	Top 1		Top 5		Time
		Raw count	%	Raw count	%	
A1	625	580	92.8	625	100	33.34
A2	625	566	90.56	624	99.84	28.49
B	724	641	88.54	715	98.76	99.18

Table 8: Elastic matching (weighted)

Test set	Num. symbols	Top 1		Top 5		Time
		Raw count	%	Raw count	%	
A1	625	253	40.48	567	90.72	81.9
A2	625	273	43.68	543	86.88	84.82
B	724	260	35.91	555	76.66	221.72

Table 9: Deformable template matching

Test set	Num. symbols	Top 1		Top 5		Time
		Raw count	%	Raw count	%	
A1	625	438	70.08	603	96.48	8.41
A2	625	448	71.68	610	97.6	6.66
B	724	540	74.59	698	96.41	25.54

Table 10: Chain code matching

Test set	Num. symbols	Top 1		Top 5		Time
		Raw count	%	Raw count	%	
A1	625	428	68.48	590	94.4	5.58
A2	625	426	68.16	588	93.92	4.11
B	724	401	55.39	608	84.12	16.29

Table 11: Stroke feature matching

Test set	Num. symbols	Top 1		Top 5		Time
		Raw count	%	Raw count	%	
A1	625	425	68	607	97.12	7.5
A2	625	427	68.32	602	96.32	5.87
B	724	433	59.81	681	94.32	22.36

Table 12: Direction element method