Recognition of On-line Handwritten Mathematical Expressions Using a Minimum Spanning Tree Construction and Symbol Dominance

Ernesto Tapia and Raúl Rojas

Freie Universität Berlin, Institut für Informatik
Takustr. 9, D-14195 Berlin, Germany
{tapia, rojas}@inf.fu-berlin.de

Abstract. We present a structural analysis method for the recognition of on-line handwritten mathematical expressions based on a minimum spanning tree construction and symbol dominance. The method handles some layout irregularities frequently found in on-line handwritten formula recognition systems, like symbol overlapping and association of arguments of sum-like operators. It also handles arguments of operators with non-standard layouts, as well as tabular arrangements, like matrices.

1 Introduction

Recognizing mathematical formulae is an important pattern recognition problem, because mathematical expressions constitute an essential part of the notation in most scientific disciplines. In *off-line* recognition, handwritten or printed formulas are given in the form of images or bit-maps, a *static representation* of the data. In *on-line* recognition, computers with pen devices (graphic tablets, contact sensitive whiteboards, Tablet PCs) store the data as *digital ink*, a *dynamic representation* which is in essence a sequence of points with temporal information.

We follow a two-step approach to recognize on-line handwritten mathematical expressions [5, 1, 2]. The first step is to divide static or dynamic data into groups of strokes, which are interpreted as single objects. A list of objects and their *attributes* (location, size, etc.) is returned. The only missing attribute for an object is its identity, which is determined using a classifier. The second step is to apply some *structural analysis* technique to obtain a hierarchical structure of the expression which describes the mathematical relationships among the symbols.

One important problem in structural analysis is the irregular writing of users, which derives in layout problems affecting the recognition of the expression. This difficulties can be overcome if the writer works with an editor which allows immediate feedback and has undo-redo and visualization capabilities [7]. A very different situation occurs when the recognition of mathematical expressions is used as an auxiliary tool embedded in another system [4,8]. This limits the interaction between the user and the recognition engine: structural analysis has to be so flexible as possible. Zanibbi et al. [9] propose a system which handles horizontal layout irregularities using a data structure which exploits the left-to-right reading of mathematical expressions. Matsakis [6] developed a method for stroke grouping called minimum spanning tree constraint, which bases the structural analysis on the proximity of symbols. Combining both approaches, we developed a method, which handles symbol overlapping and argument association for operators, besides horizontal layout irregularities.

This manuscript describes the structural analysis method we developed and is organized as follows. Section 2 introduces the concepts we need for the rest of the text. In Sect. 3 we describe the construction of the minimum spanning tree based on symbol dominance. Section 4 concludes with some comments about this work.

2 Structural Analysis of Mathematical Expressions

As mentioned in the previous section, the raw data considered in on-line recognition are points with time information. A stroke is a sequence of points generated between pen-down and pen-up events. A symbol is a sequence of strokes constructed by applying some grouping heuristic or segmentation algorithm.

To simplify structural analysis of the expression, we consider as the *basic attributes* of a symbol s its label and its bounding box. The label is obtained by means of a classifier and the bounding box is defined by the minimum x and y coordinates (x_s, y_s) of all points in the symbol, its height H_s, and weight W_s.

Once raw symbols are endowed with attributes, we collect them in *ordered attributed lists*. The order of a symbol in a list is determined by its leftmost x coordinate, i.e. if the list L is formed by the symbols $(s_1, \ldots, s_k)$, it means that $x_{s_i} \leq x_{s_j}$ for $i < j$. The attributes of a list L are its label and its bounding box attributes (x_L, y_L), H_L and W_L. The label of a symbol list is obtained during the structural analysis process by the spatial and geometrical relations between symbols, as described in Sect. 2.1.

2.1 Symbol Regions and Symbol Attributes

Relations and operator dominance in mathematical notation are defined explicitly or implicitly by the position and relative size of symbols in an expression. The spatial regions *top-left*, *above*, *superscript*, *right*, *subscript*, *below*, *below-left* and *subexpression* are used to determine such relations. For example, the operands (numerator and denominator) of the horizontal bar (fraction operator) are expected to lie in the regions above and below of the horizontal bar. See Fig. 1.

By comparing symbol attributes, we can test whether or not a symbol belongs to a determined spatial region. The *superscript threshold* and the *subscript threshold* are numeric attributes used to delimit the regions around symbols. The *centroid* is a point attribute used to determinate the symbol's location.

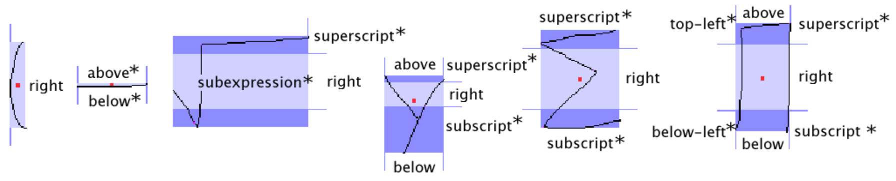


Fig. 1. Regions, thresholds and centroids of different symbol types. From left to right: non-scripted, horizontal bar, square root, scripted, sum-like and the product operator. The regions marked with an asterisk determine the range of different symbol types

To determine this symbol's attributes, we classify it as *ascendent*, *descendent* or *central*, as shown in Table 1. The reason for doing so becomes clear if we observe the layout differences in the subindex relation of the central symbol x_* and the descendent symbol y_*. The attributes for a symbol s are shown in Table 2. After obtaining symbol attributes we can determine which region symbols lie in. For example, given the symbols s and a we can define a boolean function to determine if a lies in the above region of s, as follows:

Table 1. The symbols used in our system

	scripted	superscripted	non-scripted	sum-like
ascendent	$b\ d\ \partial\ \Delta$	0 1 2 3 4 5 6 7 8 9		
descendent	y			
central	$a\ c\ x\ z\ )$	$e\ \pi\ \sqrt{\ }$	$+\ -\ *\ /\ \uparrow\ (\ \infty$	$\sum\ \int\ \prod$

Table 2. Attributes for different symbol types

	super threshold	sub threshold	centroid
ascendent	$y_s + 0.8H_s$	$y_s + 0.2H_s$	$(x + 0.5W_s, y + 0.33H_s)$
descendent	$y_s + 0.9H_s$	$y_s + 0.6H_s$	$(x + 0.5W_s, y + 0.66H_s)$
central	$y_s + 0.8H_s$	$y_s + 0.2H_s$	$(x + 0.5W_s, y_s + 0.5H_s)$

```
liesInAboveRegion($s, a$)
    1. Return getMinX($s$) $\leq$ getCentroidX($a$) $\leq$ getMaxX($s$) &&
       getSuperThreshold($s$) $\leq$ getCentroidY($a$).
```

For other regions we proceed in a similar way.

2.2 Symbol Dominance

The *range* of an operator is the expected location of its operands, see Fig. 1. Chang [3] defines dominance as follows. A symbol s *dominates* a symbol a, if a is in the range of s, and s is not in the range of a. We say that symbols *dominate* their arguments. Arguments have lower *precedence* than the dominant symbol.

We define `dominates`(s, a) as a boolean relation which depends on the set of *operator classes*

$$T = \{-, \sqrt{\ }, \text{scripted}, \text{superscripted}, \text{non-scripted}, \text{sum-like operators}\},$$

the spatial regions and symbol attributes of s and a. The symbol '$-$' represents the horizontal bar and '$\sqrt{\ }$' the square root. If `dominates`(s, a) is true, it means that s dominates a.

To clarify the concept of dominance, we give some examples. Consider the sum symbol in Fig. 2(a). It dominates the symbol '∞', because the last is in the range (superscript region) of the first and we do not expect any symbol lying on some of the regions of '∞'. Analogously, the constant 'e' in Fig. 2(b) dominates the symbols '$-$' and '$\int$'. The horizontal bar lies in the superscript region of 'e', but the last does not lie above or below the symbol '$-$'. Observe that 'e' lies in the range of the integral, but the dominance in this case is resolved by comparing their size. Figure 2(c) also shows a case where symbol dominance is not clear. We cannot determine which one of the fraction lines dominates the other, because both of them lie in the range of the other and have the same size. We can avoid the confusion here by taking as the dominant fraction bar the one with the greater centroid's y-coordinate. Observe that we added some extra conditions to the definition of Chang, namely comparison between symbol's sizes and attributes, to determine dominance and to resolve ambiguity.

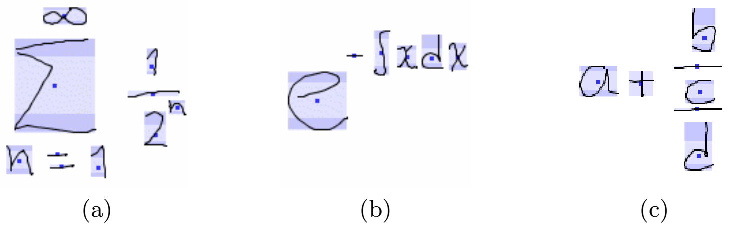


Fig. 2. Examples of expressions where (a) dominance is determined by the range, (b) dominance is determined by considering symbol sizes and (c) dominance between fraction lines is hard to determine

As we can see in this examples, dominance can be established by convention and can vary from an author to another. Different formulations of dominance define diverse *dialects* of mathematical notation.

2.3 Baseline Representation of Expressions

We describe mathematical notation as a hierarchical structure of nested baselines [9]. A *baseline* is a list which represents a horizontal arrangement of symbols in the expression. Each symbol has links to other baselines, which satisfy the spatial relations mentioned in Sect. 2.2, relative to it. For example, the expression $x_{ij} * y + \frac{a+b}{c}$ is determined by the baselines $(x, *, y, +, -)$, (i, j), $(a, +, b)$

and (c). The last two baselines satisfy the relations *above* and *below* relative to the horizontal bar, respectively.

This representation exploits the left-to-right reading of mathematical expressions. When reading an expression, one normally searches for the leftmost dominant symbol, then for the next leftmost dominant one and so on, until no more symbols are found. Given an ordered symbol list L, we can determine the leftmost dominant symbol in L through the function `getDominantSymbol`, which is defined as:

```
getDominantSymbol($L$)
    1. Let $n = \texttt{length}(L)$.
    2. If $n == 1$ return $s_1$.
    3. If $s_n$ dominates $s_{n-1}$, remove $s_{n-1}$ from $L$, in other case remove $s_n$.
    4. Return getDominantSymbol($L$).
```

Observe that this function uses the order of symbols in L.

In this way, given a list L, we construct its dominant baseline Db through function:

```
getDominantBaseline($Db, L$)
    1. If $Db$ is empty, then set $Db = \texttt{addSymbol}(Db, \texttt{getDominantSymbol}(L))$.
    2. Set $s = \texttt{getLastSymbol}(Db)$.
    3. Constructs a list $Hs = \texttt{getRightNeighbors}(s, L)$ of symbols in $L$ which
       are right horizontal neighbors of $s$.
    4. If $Hs$ is empty, return.
    5. Find the dominant symbol of the horizontal neighbors,
       $sd = \texttt{getDominantSymbol}(Hs)$.
    6. Set $Db = \texttt{addSymbol}(Db, sd)$.
    7. Use recursion: getDominantBaseline($Db, L$).
```

We take special care in the definition of the function `getRightNeighbors` (step 3), to handle irregular horizontal layouts.

Now, we are ready to construct the baseline tree of the mathematical expression described by the ordered symbols list L by finding recursively dominant baselines. This is done by the function `constructBaselineMST`:

```
constructBaselineMST($L$)
    1. If $L$ is empty, return.
    2. Set $Db = \emptyset$.
    3. getDominantBaseline($Db, L$).
    4. constructDominanceMST($Db, L$)
    5. For each symbol $s \in Db$, construct new symbols lists with its children
       obtained in the MST step, depending which spatial relations they satisfy
       and assign this lists to the corresponding links. The identity of this lists
       corresponds to the spatial relation they satisfy.
    6. Set $L = Db$.
    7. For each symbol $s \in Db$, use recursion applying constructBaselineMST
       to each of the child lists obtained in step 5.
```

Next section explains how this is done.

3 A Minimum Spanning Tree Construction and Symbol Dominance

The *minimum spanning tree* (*MST*) constraint method [6], considers the centers of the stroke bounding boxes as nodes of a completely connected weighted graph. Although this approach seems to be robust for stroke grouping (based on the minimization of a sum cost function), it lacks a robust method to carry out structural analysis. Compare Fig. 3(a) and Fig. 3(d).

Our experiments show that we can avoid many shortcomings in structural analysis if we use symbol dominance information to construct the MST. This method is described in the next three sections.

3.1 Weight Calculation Based on Attractor Points

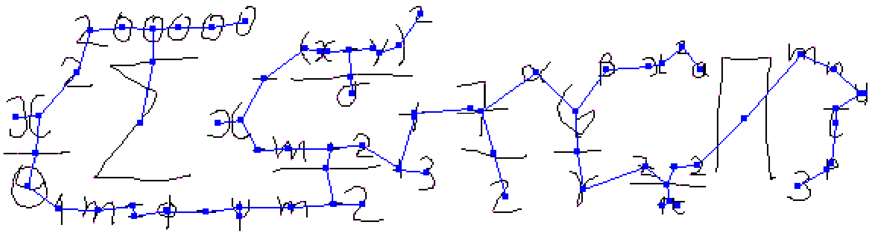
We consider the previously recognized symbols as the nodes of a totally connected weighted graph. Then, we use Prim's algorithm to construct its MST: a new edge (s_t, s_n) is added to the MST if its corresponding weight $w(s_t, s_n)$ is the minimum of all edges, where s_t belongs to the MST and s_n does not belong to the MST. The function $\texttt{constructMST}(Db, L)$ constructs the MST of the symbol list L. This is done by initializing the MST to $\{(s_1, s_2), \ldots, (s_{k-1}, s_k)\}$ where the symbols s_i belong to the dominant baseline $Db = (s_1, \ldots, s_k)$. In our method, the crucial step is the weight calculation of edges, where we use symbol dominance.

If $\texttt{dominates}(s_t, s_n)$ is true, the weight $w(s_t, s_n)$ is the minimum distance between *attractor points* of symbols s_t and s_n. If $\texttt{dominates}(s_t, s_n)$ is false, the weight corresponds to the distance between the centroids of s_t and s_n. Finally, if the relation $\texttt{right}(s_t, s_n)$ or $\texttt{right}(s_n, s_t)$ is true, the weight is the minimum distance among their corresponding black points as shown in the second a of Fig. 4. Figure 3(b) shows the first recursion step of `constructMST` without using dominance analysis.

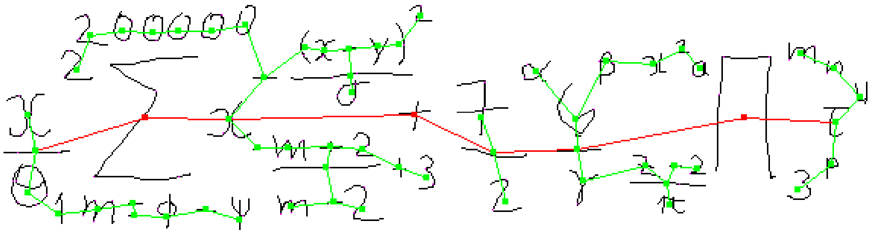
The attractor points are located in the boundary of the symbol bounding box. The number of such points depends on the operator class. Figure 4 shows the attractor points corresponding to the different symbol classes when the first a and the integral (sum-like operators and square root) are dominated by x (scripted) and 2 (superscripted), the second a is dominated by + (non-scripted) and when the third a is dominated by sum-like operators and the horizontal bar.

3.2 Arguments to Special Operators

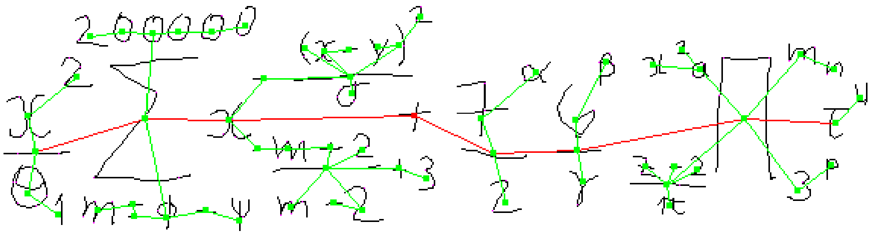
The range of symbols in the dominant baseline is not limited by the threshold attributes alone but also by "neighbor" operators [9]. Figure 5 shows how attractor points and symbol dominance help to define dominance regions. To draw the regions, we proceed as follows. Firstly, we take a pixel from the whole region and translate the symbol 'a', leftmost symbol in the region, such that its centroid and the pixel coincide. Secondly, we associate each symbol in the



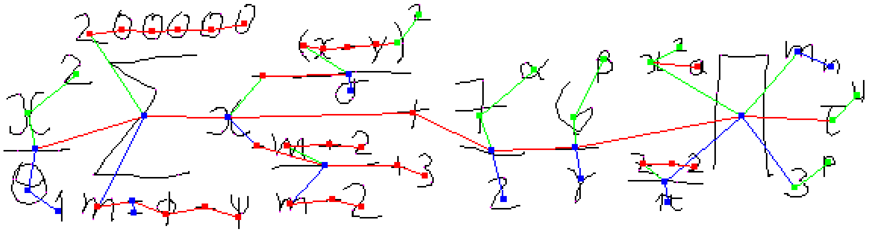
(a)



(b)



(c)



(d)

Fig. 3. (a) Minimum spanning tree of strokes. MST of the first recursion step of our method (b) without using dominance analysis and (c) using dominance analysis. (b) Final tree of spatial relations

baseline $(x, -, y, \sum, z, \prod)$ with different grey tones. Finally, the pixel is colored with the grey tone corresponding to the "nearest" symbol of the baseline, in terms of the edge weight used in the MST construction. Figure 5(a) shows the

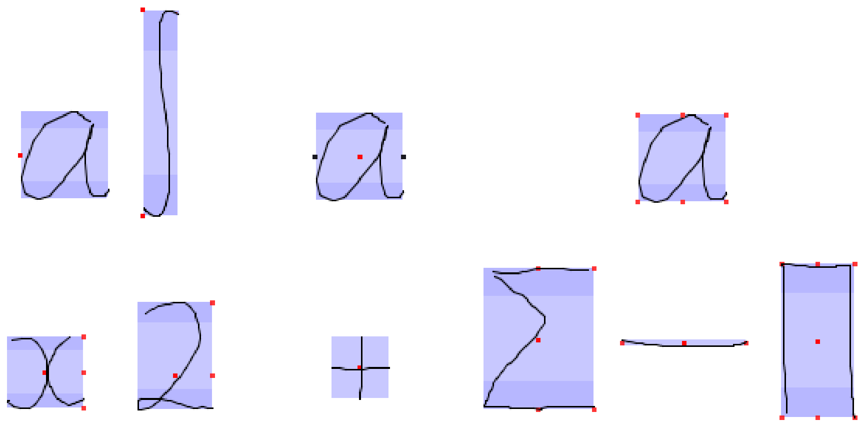


Fig. 4. Above: Attractor points of symbols not belonging to the MST. Below: Attractor points of symbols and operators belonging to the MST

regions for each symbol, using the distance between centroids as edge weights. We can appreciate in Fig. 5(b) that using symbol dominance to delimit regions, corresponds to the expected range of symbol operators.

Figure 5(d) shows how the range can be extended during MST construction. In this example, the regions of the symbols z and a are merged in such a way that the new symbol b lies in their range and ambiguities arising from argument association with $\prod$ are overcome. The regions were found as described before, but in this case we use the symbol b instead of a.

It can be seen why using the MST construction allows more flexibility to handle irregular layouts. For example, this is the case when we change an expression by adding some super indexes after entering it. The same applies when associating arguments to non-standard operators as ${}^{*}_{*}\prod^{*}_{*}$ as shown in Fig. 3.

Our method encounters problems when scripted symbols lie too close to the arguments of fraction or sum-like operators. The horizontal baselines of dominated symbols are merged incorrectly, when they are written too far away from operators and the last are written to close to each other. See Fig. 5(c) and Fig. 6(a) for an example of this. To avoid the problem, we multiply the corresponding weight by a factor $0 < \alpha < 1$ during MST construction, when the symbol in the dominant baseline is a sum-like operator or a fraction line. See Fig. 6(b).

3.3 Recognition of Matrices and Tabular Arrangements

The symbol '[' and ']' were taken as reserved symbols to construct matrices. The range of the symbol '[' is the bounding box which contains it and its corresponding closing square bracket.

After building the MST, we check for each $s \in Db$ whether it is an open square bracket or not. If it is, we proceed to identify row structures in the child

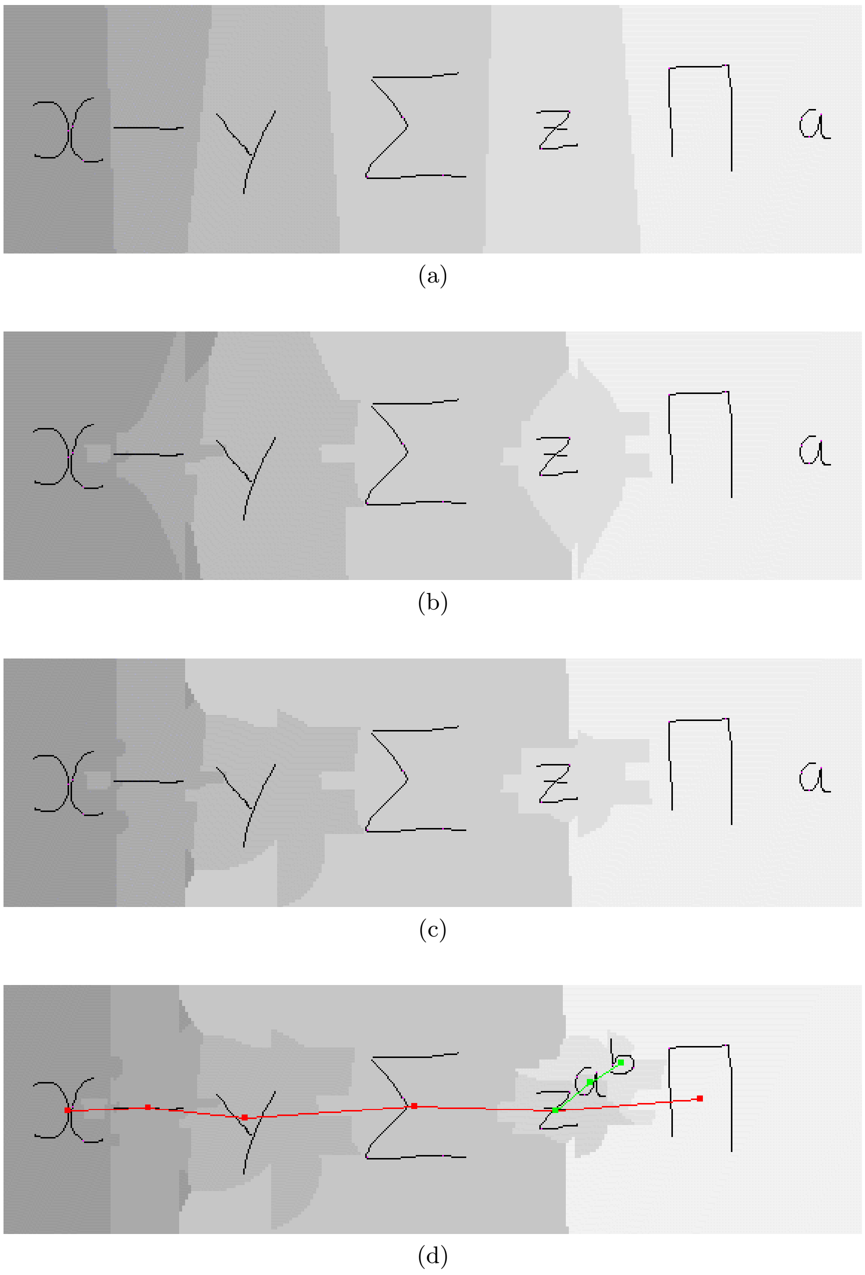


Fig. 5. Regions defined (a) using only the centroids, (b) using the attractor points and symbol dominance without distance factor and (d) with distance factor. (d) Growing regions in the MST construction

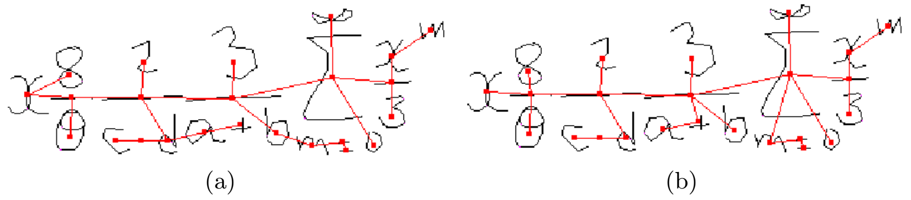


Fig. 6. The result of the MST construction (a) without using the α factor in weight calculation and (b) using the factor

list Ds of symbols dominated by s (found by the `countructMST` function). To this purpose, we define the *area projection function* f as

$$f(y) = \sum_{\substack{s \in Ds \\ y_s \leq y \leq y_s + H_s}} W_s H_s, \tag{1}$$

where $y_{Ds} \leq y \leq y_{Ds} + H_{Ds}$. We use the local maxima of a smoothed version of f, located at y_i, $i = 1, \ldots, n$, to define the attractor points (x_{Ds}, y_i) of s (see Fig. 7). The next step is to construct the MST of s and Ds using the "dynamically" constructed attractor points. Because we want to find rows in the symbol list, we multiply the x-coordinates of attractor points and centroids by a factor $0 < \beta < 1$ and we re-calculate the weights of the graph with this modification. This is a way to contract horizontally the distance between symbols in Ds and give more weight to the vertical variations of symbol's positions. Finally, we assign rows to the corresponding children lists of s, locate spaces in the rows and apply the method recursively to those lists.

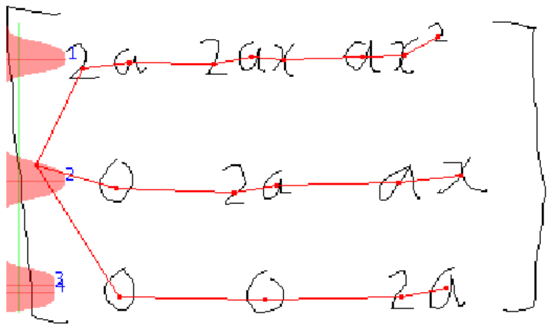


Fig. 7. The area projection function and the MST found in the matrix mode

4 Comments and Further Work

We presented a method for the structural analysis of on-line handwritten mathematical expressions based on a minimum spanning tree construction and symbol dominance. Our experiments showed that our method is robust to handle some layout irregularities. Our method handles non-standard layouts, like ${}^{*}_{*}\prod^{*}_{*}$. This can be easily extended to recognize expression which contains operators like ${}_{n}\mathbf{C}_{k}$, or other operators which similar layouts. We also consider a method for the recognition of tabular arrangements, which can be easily extended to recognize stacked arguments of sum-like operators, like the one used in (1). This two characteristics are not found in other systems for on-line recognition. Fig. 8 shows some examples of expressions recognized by the current system.

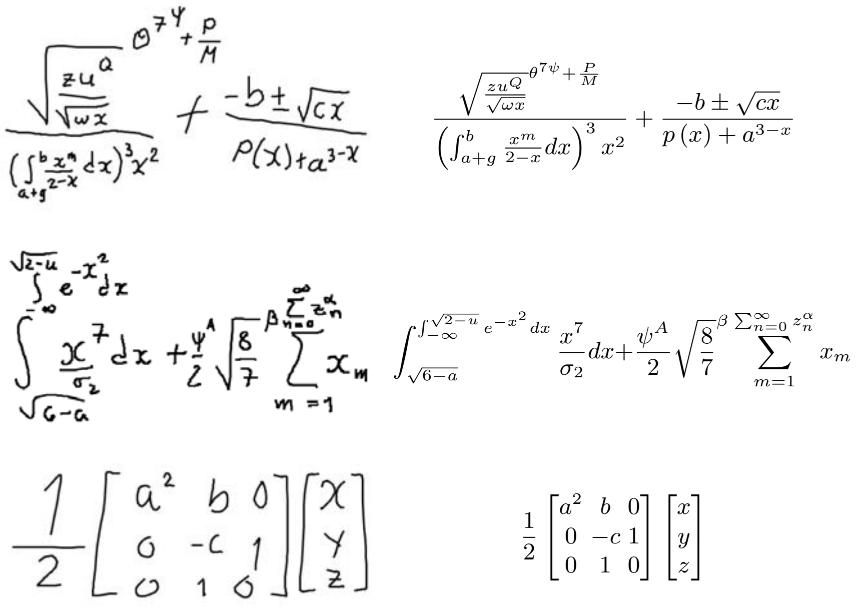


Fig. 8. Some expressions recognized by our current system

The recognition of on-line handwritten symbols is done using support vector machines and neural networks [8]. It is clear that some classification errors can occur. This could be problematic, for example, when recognizing the operator '$\int$' as '5', because our structure analysis assumes perfect symbol recognition. To avoid it, we can use an interface similar to the one we developed in [7], which allows immediate feedback and has undo-redo and visualization capabilities.

Determining heuristic values for α and β as described in the previous section, requires some experimentation. We have obtained satisfactory results by using the values $\alpha = 1/4$ and $\beta = 1/15$. We plan to construct a benchmark of on-line handwritten mathematical expression to determine the optimal values of the parameters α and β and the other ones required by our algorithm, as well as to obtain a numerical estimation of recognition rates.

The motivation of this research is to incorporate formula and gesture recognition capabilities in the electronic chalk board (E-Chalk), a multimedia system for distance-teaching [4, 8]. At present, E-chalk can convert the stored lectures into PDF format. One of our objectives is that lectures will be converted also into some electronic format like LaTeX.

Acknowledgments

The authors like to thank the anonymous reviewers for a variety of suggestions that helped to improve this paper. Ernesto Tapia thanks the Mexican National Council for Science and Technology (CONACyT) for its financial support during his Ph.D. studies via the credit-scholarship number 154901.

References

1. D. Blostein and A. Grbavec. Recognition of Mathematical Notation. In P. S. P. Wang and H. Bunke, editors, *Handbook on Optical Character Recognition and Document Analysis*. World Scientific Publishing Company, 1996.
2. Kam-Fai Chan and Dit-Yan Yeung. Mathematical Expression Recognition: a Survey. *International Journal on Document Analysis and Recognition (IJDAR)*, 3(1):3–15, 2000.
3. S. Chang. A Method for the Structural Analysis of Two-Dimensional Mathematical Expressions. *Information Sciences*, 2:253–272, 1970.
4. G. Friedland, L. Knipping, and R. Rojas. E-Chalk Technical Description. Technical Report B-02-11, Freie Universität Berlin, Institut für Informatik, 2002.
5. H. J. Lee and J. S. Wang. Design of a Mathematical Expression Undrestanding System. In *Proceedings of the Third International Conference on Document Analysis and Recognition (ICDAR)*, pages 1084–1087, 1995.
6. N. Matsakis. Recognition of Handwritten Mathematical Expressions. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, May 1999.
7. E. Tapia. JMathNotes: A Java-Based Editor for On-Line Handwritten Mathematical Expressions, January 2004. `http://www.inf.fu-berlin.de/~tapia/JMathNotes`.
8. E. Tapia and R. Rojas. Recognition of On-Line Handwritten Mathematical Formulas in the E-Chalk System. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR)*, August 2003.
9. R. Zanibbi, D. Blostein, and J. Cordy. Recognizing Mathematical Expressions Using Tree Transformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(11), November 2002.