International Journal on Document Analysis and Recognition (IJDAR) (2018) 21:253–268
https://doi.org/10.1007/s10032-018-0306-1

ORIGINAL PAPER

Augmented incremental recognition of online handwritten mathematical expressions

Khanh Minh Phan[1] · Anh Duc Le[2] · Bipin Indurkhya[3] · Masaki Nakagawa[1]

Received: 15 April 2017 / Revised: 17 March 2018 / Accepted: 29 May 2018 / Published online: 16 June 2018

Abstract
This paper presents an augmented incremental recognition method for online handwritten mathematical expressions (MEs). If an ME is recognized after all strokes are written (batch recognition), the waiting time increases significantly when the ME becomes longer. On the other hand, the pure incremental recognition method recognizes an ME whenever a new single stroke is input. It shortens the waiting time but degrades the recognition rate due to the limited context. Thus, we propose an augmented incremental recognition method that not only maintains the advantage of the two methods but also reduces their weaknesses. The proposed method has two main features: one is to process the latest stroke, and the other is to find the erroneous segmentations and recognitions in the recent strokes and correct them. In the first process, the segmentation and the recognition by Cocke–Younger–Kasami (CYK) algorithm are only executed for the latest stroke. In the second process, all the previous segmentations are updated if they are significantly changed after the latest stroke is input, and then, all the symbols related to the updated segmentations are updated with their recognition scores. These changes are reflected in the CYK table. In addition, the waiting time is further reduced by employing multi-thread processes. Experiments on our dataset and the CROHME datasets show the effectiveness of this augmented incremental recognition method, which not only maintains recognition rate even compared with the batch recognition method but also reduces the waiting time to a very small level.

Keywords Handwriting recognition · Mathematical expression recognition · Incremental recognition · Batch recognition

1 Introduction

Nowadays, mathematical expressions (MEs) are used widely in many fields such as science, engineering, education and economy. Basically, there are three methods to input mathematical expressions to a device: a user can use an editor like Microsoft Equation Editor (MEE); input the mathematical equation by a math description language like LATEX; or employ handwritten ME recognition. The former two are practical, but the user must select menus and find symbols/expressions in a long list, or remember the grammar of math symbols/ expressions. They are useful for scientists, engineers, teachers, businesspeople and other professionals, but awkward for ordinary people. The last method based on handwritten ME recognition is easy to use for everybody, but the recognition rate is still poor.

In recent years, touch-based and pen-based devices using display-integrated tablets have become more commonplace. Many people own one or more such devices: they can read and annotate documents, write memos and draw figures on their devices in a natural way using a pen or just their finger. Moreover, they can receive immediate feedback so that they can interact with these devices in real time.

Self-learning systems on these devices are also becoming available. Learners write MEs for answers, the devices recognize handwritten MEs, the learners verify or correct

✉ Masaki Nakagawa
nakagawa@cc.tuat.ac.jp

Khanh Minh Phan
pmkhanh7890@gmail.com

Anh Duc Le
leducanh841988@gmail.com

Bipin Indurkhya
bipin@agh.edu.pl

[1] Department of Computer and Information Sciences, Tokyo University of Agriculture and Technology, 2-24-16, Naka-cho, Koganei-shi, Tokyo 184-8588, Japan

[2] NTT Hi-Tech Institute, Nguyen Tat Thanh University, 300A Nguyen Tat Thanh, District 4, Ho Chi Minh city, Vietnam

[3] Department of Computer Science, AGH University of Science and Technology, Al. Mickiewicza 30, 30-059 Kraków, Poland

recognition results, and then they want to know whether their answers are correct. The success of such systems relies on handwritten ME recognition, and so the recognition rate should be high enough for them to be considered trustworthy.

Handwritten ME recognition, especially online handwritten ME recognition (hereafter we call OHME), however, is not a new research theme. It started around 1970 with a top-down approach by Anderson [1] and a bottom-up approach by Chang [2]. Recently, OHME recognition has been getting more attention as touch-based and pen-based devices are proliferating into education and learning environments. A series of contests named CROHME has been held in 2011, 2012, 2013, 2014 and 2016 at International Conferences on Document Analysis and Pattern Recognition and International Conference on Frontiers in Handwriting Recognition.

There are two approaches to recognize an OHME: after the entire OHME is written (batch recognition), or incrementally as each stroke is written one by one (incremental recognition). Especially, pure incremental recognition refers to the method of recognizing an input stroke sequence every time a new stroke is written. Batch recognition can produce higher recognition rate because the entire context is available, but it takes longer to output the result. On the other hand, incremental recognition reduces the waiting time but the recognition rate is lower.

We briefly survey previous works in the batch recognition approach. They reported better recognition rates and achieved higher ranks in the above-mentioned competitions.

Alvaro et al. [3], who won the first prize at CROHME 2011, second prize at CROHME 2013 and CHROME 2014, modified the Cocke–Younger–Kasami (CYK) parsing algorithm to parse an input OHME in 2 dimensions (2D) under two-dimensional stochastic context-free grammar (SCFG). Their method is stroke order-free. They employed range search to decrease time complexity from $O(n^4|P|)$ to $O(n^3 log_n|P|)$.

Awal et al. [4] presented another SCFG-based method. They proposed a global learning approach to learn symbol segmentation and symbol recognition directly from training OHMEs. Their system received the second prize and the third prize at CROHME 2011 and CROHME 2012, respectively. Their method is stroke order-free.

Yamamoto et al. [5] and Simistira et al. [6] also employed SCFG red. Stroke order is used to decrease the number of sub-partitions that must be considered during parsing to $O(n^2)$ and reduce the complexity of the parsing algorithm to $O(n^3|P|)$. Although the complexity is low, it is not robust for stroke order variations. Le et al. [7] extended the grammar rules to handle common symbol order variations; their system received the third prize at CROHME 2013 and at CROHME 2016.

Okamoto et al. [8] published another stroke order-dependent method using positional relations rather than parsing. Their method partitions an input OHME into components by recursive horizontal and vertical cuts. Then, it unifies separated components, recognizes symbols, analyses the structure and generates the output.

Zanibbi et al. [9] proposed a stroke order-dependent method, which decomposes the recognition process into symbol segmentation, symbol recognition and structural analysis. Their method constructs a 2D arrangement of input symbols, groups multiple input symbols (such as decimal numbers and function names), analyzes expression syntax and produces an operator tree. Hu et al. [10] used Edmonds algorithm to find the maximum spanning tree (MST) in a directed line-of-sight graph. The MST-based parser finds the higher formula structure and expression rates.

Julca-Aguilar et al. [11] proposed another graph-based method. This method builds a context-free graph from the input strokes and then employs a top-down parsing algorithm to partition each node until a terminal symbol is reached.

Garain et al. [12] employed a bottom-up approach using a stroke order-dependent method. It combines feature template matching and HMM for recognizing symbols. The mathematical structures, like superscripts, square roots and limits, are detected by using the baseline information of each symbol. Finally, the sub-expressions are merged into a larger expression until the final expression is constructed.

Recently, Le et al. [13] and Zhang et al. [14] introduced a stroke order-dependent method based on deep learning. This end-to-end encoder–decoder framework can train symbol segmentation, symbol recognition and classification of spatial relations together. Experiments show an improved performance when compared with the other systems.

Our interest is in incremental recognition since it is much needed for user interfaces of educational applications. We review these works below.

MacLean et al. [15], who got the second prize at CROHME 2012, proposed a top-down parsing algorithm. Whenever a new stroke is written, they incrementally construct a shared parse forest representing all recognizable parses of an input by Ungers method [16]. Then, the most highly ranked tree is extracted from this forest. They sort input strokes in x-direction (horizontal) and y-direction (vertical) to segment them using grammar production rules. This method restricts the infeasible segmentations while achieving stroke order independence. However, the worst-case complexity of parsing is still $O(n^4|P|)$, which is quite large.

Predovic et al. [17] proposed another incremental method following the stroke order-free approach by partitioning the input strokes into multiple stroke regions. Strokes of a region are ordered according to left-to-right, top-to-bottom and outside-to-inside relations. They score both the non-terminal and the terminal grammar objects included in each region and represent these objects as chart entries. Whenever a system detects pause or a new action from the user, it determines

Table 1 Properties of previous works on OHME recognition

Method	Properties	
	Incremental/batch	Stroke order-free/dependent
Alvaro et al. [3]	Batch	Free
Awal et al. [4]	Batch	Free
Yamamoto et al. [5]	Batch	Dependent
Simistira et al. [6]	Batch	Dependent
Le et al. [7]	Batch	Dependent
Okamoto et al. [8]	Batch	Free
Zanibbi et al. [9]	Batch	Dependent
Hu et al. [10]	Batch	Free
Julca-Aguilar et al. [11]	Batch	Free
Garain et al. [12]	Batch	Free
Le et al. [13]	Batch	Free
Zhang et al. [14]	Batch	Free
MacLean et al. [15]	Incremental	Free
Predovic et al. [17]	Incremental	Free
Vuong et al. [18]	Incremental	Free
Phan et al. [19]	Incremental	Dependent

which chart entries are affected and updates them accordingly.

Vuong et al. [18] proposed a progressive structural analysis for dynamic recognition by using the mathematical expression tree (MET). This method is stroke order-free. The latest input symbol is updated into the corresponding position of the MET. Meaningful consecutive symbols in MET are grouped into a mathematical unit. The advantage of this method is that it allows the users to correct misrecognized symbols.

Phan et al. [19] proposed an incremental recognition method for OHMEs, which is based on the method proposed by Le et al.. This method employs a bottom-up approach and updates the CYK table whenever a new stroke is input.

The recognition methods mentioned above incrementally recognize an input OHME after each written stroke, so we categorize them as the pure incremental approach. The pure incremental approach reduces the waiting time but yields a lower recognition rate due to limited context. To rectify this, Phan et al. [20] proposed a semi-incremental approach, which considers larger context to recognize input strokes. Table 1 lists the above-mentioned works on OHME recognition with their properties.

It is also important to consider the user interface. Lazy recognition interface delays the feedback of recognition until it is needed. Users may not need continuous feedback from ME recognition while writing, but only to see the final results when they are done. Misrecognition and even display of correct recognition during writing may interrupt the users thinking [21]. The batch recognition method is suitable for such a user interface.

On the other hand, a user interface based on busy recognition or on-the-fly recognition provides the recognition feedback in real time, thereby enabling the users to interact with the feedback to erase and rewrite strokes in case of incorrect recognitions. It takes advantage of interactive touch-based or pen-based devices employing display-integrated tablets. The incremental recognition approach is suitable for such a user interface, but it can be applied also for the lazy recognition interface.

The method proposed here is an incremental recognition method, which can be incorporated into busy recognition or on-the-fly recognition user interfaces.

This paper is based on our earlier prototype on which uses pure incremental and semi-incremental recognition of OHMEs [19,20]. Since then, we have refined our technique while removing some redundancies to reduce the waiting time. Here, we redesign the prototype incorporating these new techniques and describe here the revised approach in more detail. We also evaluate prototype on four datasets, three of which are publicly available. We also present a detailed analysis of misrecognitions.

The augmented incremental recognition method minimizes the waiting time while increasing the recognition rate to be nearly the same as that of the batch recognition. Unlike the pure incremental recognition method, which only processes the latest stroke, the augmented incremental recognition method considers the previous segmentation and recognition results as well as the latest stroke.

Our method is basically stroke order dependent so that its merit is a smaller time complexity, but it is less robust with respect to interspersed strokes over multiple symbols compared to stroke order-free methods. Although another work considers stroke reordering to solve this problem [22], we confine ourselves to making incremental recognition based on batch recognition.

We define a stroke as a time sequence of pen-tip or fingertip coordinates from pen (finger)-down to pen (finger)-up.

The rest of the paper is organized as follows. Section 2 presents an overview of the batch recognition method and features employed. Section 3 explains how the augmented incremental method works. Section 4 presents recognition experiments, and the conclusions are presented in Sect. 5.

2 Batch recognition method and employed features

In this section, we summarize the batch recognition method to recognize OHMEs, from which we reformulate the incremental recognition method. We describe its features in detail as they form the basis of the incremental method as well.

The batch recognition is composed of five major tasks: feature extraction, symbol segmentation, symbol recognition, spatial relation classification and structure analysis. After all the strokes for an OHME are input, it is segmented, each symbol is recognized, its structure and relation are analyzed, and the Cocke–Younger–Kasami (CYK) [23–25] algorithm is employed to find the best interpretation of the OHME. We follow a soft-decision paradigm in that multiple choices are considered in performing segmentation and recognition. The process is described in detail below

2.1 Feature extraction

When the user writes an entire OHME on a tablet, twenty-one geometric features are extracted for each pair of consecutive strokes, where eight features are normalized by the average height of all input strokes ($\overline{h}$) such as such as x- and y-projections of the nearest bridge between two strokes, x- and y- distances between two bounding boxes of two strokes and x- and y-coordinates of the midpoint between centers of two bounding boxes, while the remaining thirteen features are calculated from the ratios of two local values. Thus, all the features are scale invariant. In "Appendix," Table 7 lists these features, which are depicted in Fig. 13 and Table 8 explains the terms and symbols used in Table 7.

2.2 Symbol segmentation

An SVM classifier is used for segmentation, which takes 21 geometric features extracted in the previous step as the input and generates symbol hypotheses. We assume that the maximum number of strokes forming a single mathematical symbol is four plus one additional stroke, because a stroke is sometimes accidentally chopped while writing (maximum of four strokes for E plus one additional stroke).

2.3 Symbol recognition

A symbol recognizer is used to recognize each symbol hypothesis from the previous task. The recognizer is a combination of offline and online recognition methods [26]. For the online recognizer, we use Markov random field (MRF) to get a similarity measure for each symbol class. For the offline recognition, we use the modified quadratic discriminant function (MQDF) to calculate the distances of the input pattern to each symbol class. The measurement is different between MRF (probability, the higher the better) and MQDF (distance, the lower the better). We combine the MRF and MQDF as follows:

$$score_{comb} = CDF(w_1 \times score_{on} + w_2 \times score_{off}) \quad (1)$$

where $score_{comb}$, $score_{on}$ and $score_{off}$ are the combination score, online recognition score and the offline recognition score; w_1 and w_2 are the weighting parameters for combination; CDF is a cumulative distribution function to normalize the combination score to [0, 1]. The advantages of both the methods are maintained in this combined system while reducing the weaknesses of each method. Particularly, the online method works well for connected strokes or cursive strokes, while the offline method can overcome the problem of out-of-order strokes or duplicated strokes.

The mathematical symbol recognition is almost the same as the Japanese character recognizer [26] except for three main modifications. The first and the most important change is to replace all categories in the Japanese character dictionary by 101 mathematical symbol categories. Second, the recognition result of each symbol pattern is limited to the five highest score candidates. Third, the candidates of period, comma and prime are considered for each recognized symbol. If the symbol has both the height (h) and width (w) less than a half of the average height ($\overline{h}$) and width ($\overline{w}$) of the current OHME, respectively, the probability (old_P) of "period," "comma," or "prime" is added a bonus probability B which is calculated as shown in (2).

$$new\_P_{(period/comma/prime)} = \min(old\_P + B, 1) \quad (2)$$

$$B = F(h) \times G(w) \quad (3)$$

where F and G are fuzzy functions as shown in (4) and (5).

$$G(w) = \begin{cases} 1 & \text{if } 0 < w < \frac{\overline{w}}{10} \\ \frac{5\overline{w}-10w}{4} & \text{if } \frac{\overline{w}}{2} > w > \frac{\overline{w}}{10} \\ 0 & \text{if } w > \frac{\overline{w}}{2} \end{cases} \quad (4)$$

$$F(h) = \begin{cases} 1 & \text{if } 0 < h < \frac{\overline{h}}{10} \\ \frac{5\overline{h}-10h}{4} & \text{if } \frac{\overline{h}}{2} > h > \frac{\overline{h}}{10} \\ 0 & \text{if } h > \frac{\overline{h}}{2} \end{cases}. \quad (5)$$

2.4 Structural relation

Structures or relations among symbols in OHMEs are ambiguous in some cases even for humans. The spatial relation only considers the relation between a pair of elements (an element can be a symbol or a sub-expression). Therefore, features of a structural relation do not need to be normalized by the global features of an OHME. First, a soft-decision approach is developed using the body box, which can represent the main position and size of each symbol [7]. For example, the body boxes for ascendant symbols like "d," "k" and "h" are lower than their bounding boxes. Figure 1 shows an example of four groups of symbols. The body boxes of two symbols represent their relations

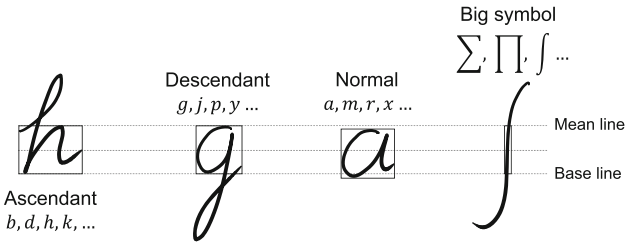


Fig. 1 Four groups of symbols

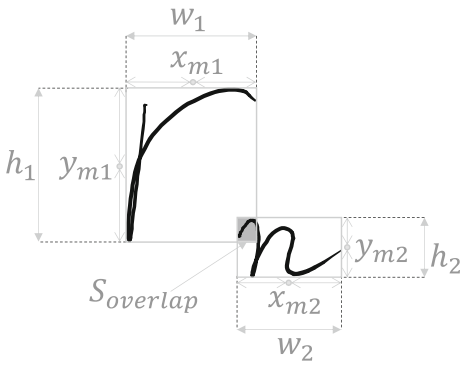


Fig. 2 Parameters in structural relations

better than the bounding boxes. Here, it should be noted that the body box for each symbol pattern is trainable and different for each symbol recognition candidate. The body box of a composite expression (non-terminal symbol) is defined from its constituent symbols including non-terminal symbols

Second, from the body boxes of a pair of two successively written symbols, four features are extracted by (6), (7), (8) and (9).

$$D_x = \frac{(x_{m2} - x_{m1})}{w_1} \tag{6}$$

$$D_y = \frac{(y_{m2} - y_{m1})}{h_1} \tag{7}$$

$$H = \frac{h_2}{h_1} \tag{8}$$

$$O = \frac{S_{overlap}}{h_2 \times w_2} \tag{9}$$

where (x_{mi}, y_{mi}), w_i and h_i $(i = 1, 2)$ denote the center of the width and the height of the bounding box, respectively, and the suffix i is 1 for the preceding body box and 2 for the succeeding body box.

The features D_x, D_y show the distances between the horizontal centers and the vertical centers of the two body boxes scaled by the width and the height of the preceding body box, respectively. The feature H is the ratio of the height of the preceding body box and that of the succeeding body box. The feature O is the ratio of the overlapping area over the size of the succeeding body box. Figure 2 shows an example of all the terms used to obtain the four features.

Finally, these features are taken as input to obtain the probabilities for six relations between the pair (horizontal, superscript, subscript, above, below and inside) by SVMs.

2.5 Structure analysis

We have defined a two-dimensional stochastic context-free grammar (2D-SCFG). Some rules of the 2D-SCFG for MEs are shown in Table 2. Although the method is not robust with respect to unexpected stroke order variations, all the expected writing order variations are registered in the grammar, such as all the permutated orders of an integer in the integer part, and the numerator, the denominator and the fraction bar in the fraction part.

Table 2 Some rules of 2D-SCFG for mathematical expressions

No.	Rule
1	Exp → SupExp
2	SupExp → superscript, Number
3	Exp → horizontal, OpExp
4	OpExp → horizontal, Operator_Exp
5	Exp → Number
6	Number → {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
7	Operator → {+|−}

We have employed the CYK algorithm, which takes segmentation probabilities, symbol recognition scores and relation probabilities of all terminal and non-terminal symbol candidates obtained in the previous steps as the input. Each candidate in a cell has a score, which is a combination of segmentation probability, symbol recognition score and relation probabilities for six relations.

The system configured by the 2D-SCFG invokes the CYK algorithm to produce a list of candidates for each input sequence of strokes $s = s_1, s_2, \ldots, s_n$ for an OHME. The algorithm is composed of two stages:

Initial stage: We initialize the first five rows in the CYK table, because a mathematical symbol is formed from at most five strokes. In each cell, we store an array of nodes, each of which contains the candidate production and its score.

Parsing stage: We employ $X \overset{rel}{\rightarrow} AB$ production rules to reduce two sub-MEs to a non-terminal. Then, we employ $Y \rightarrow C$ production rules to reduce the non-terminal further to another non-terminal. We store five best candidates in each cell of the CYK table for an OHME. The candidates of the final result are extracted from the top cell.

3 Augmented incremental recognition method

We not only aim to minimize the waiting time but also to keep the recognition rate as high as that of the batch recognition (where symbol recognition and building the CYK table are time-consuming). In this section, we introduce an effective method that can be processed in parallel. This method does not only focus on the latest stroke and its neighboring strokes, which may be combined with the latest stroke to form a symbol, but can also correct erroneous segmentations and recognitions made far away from the latest stroke, even up to the first stroke. The proposed method is applicable to batch recognition methods employing segmentation and isolated symbol recognition.

In our previous research [20], we have found that human writers typically spend about 0.8 s between strokes when they are writing. Long breaks over 2 s are not included in the average. We propose to utilize this elapsed time for processing input strokes.

Because of these reasons, we expect to return a high recognition rate without interrupting the users.

3.1 Processing flow

The processing flow of the augmented incremental recognition is shown in Fig. 3. This flow contains the pure incremental recognition method shown in white to which steps for correcting erroneous segmentations and misrecognitions shown in black are added. To begin with, the system receives a new stroke from the user. In the second step, it updates the geometric features and segmentation. In the third step, it recognizes the symbols related to the recently received stroke. Then, it analyzes the structure and updates the CYK table to get the OHME recognition result. The latest result is used for the next processing cycle, and the process is repeated until the input is finished.

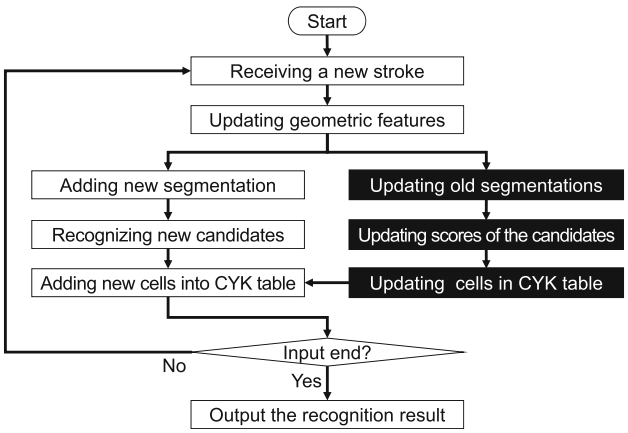


Fig. 3 Flow of the augmented incremental recognition

The parts shown in black in Fig. 3 are to correct the erroneous segmentations and recognitions away from the current stroke. The augmented incremental method applies the pure incremental process, but additionally performs other operations to correct erroneous segmentations and misrecognitions. This step updates the available segmentations, recognitions and cells in the CYK table if there is any change caused by the latest stroke. These steps are repeated after every new stroke rather than after a whole OHME so that the recognition result is shown immediately after the user is finished writing, while employing a larger context to segment and recognize the input OHME

When the user enters a new stroke, the geometric features are updated and the segmentation related to the latest stroke is added. If previous segmentations are changed, they are also updated in the following steps. First, all the candidates related to the latest stroke are recognized. Second, the candidates related to the updated segmentations in the first step are also updated by the new scores. Third, the CYK table is updated with new cells added. Finally, the latest result is reused for the next processing cycle.

3.2 Updating geometric features

In the batch recognition, eight features are normalized by the average height of all the strokes. For incremental recognition, however, all the strokes are not available to calculate the exact average height. Therefore, we approximate the average height by taking the average from the first stroke to the latest stroke. Whenever a new nth stroke is input, the average height ($\overline{h}$) is recalculated for the n strokes, eight features of the stroke pairs in Table 1 for the $(n-2)$ previous pairs, i.e., a total of $8 \times (n-2)$ features are renormalized by the updated $\overline{h}$ and the twenty-one features are extracted for the latest pair. Extending the approach of our earlier prototype, the remaining thirteen features for the previous $(n-2)$ pairs need not be updated as they are scaled locally by the neighboring features.

3.3 Adding new segmentation and updating old segmentations

The process of segmentation includes two steps. First, the new segmentation probability between the latest stroke and its previous stroke is calculated. Second, the previous segmentations may be changed because they are dependent on the eight features normalized by $\overline{h}$. Therefore, we recalculate the new probabilities of all previous pairs. Then, if there is any pair, the segmentation probability of which changes more than the specified segmentation threshold, the probability of this pair is updated.

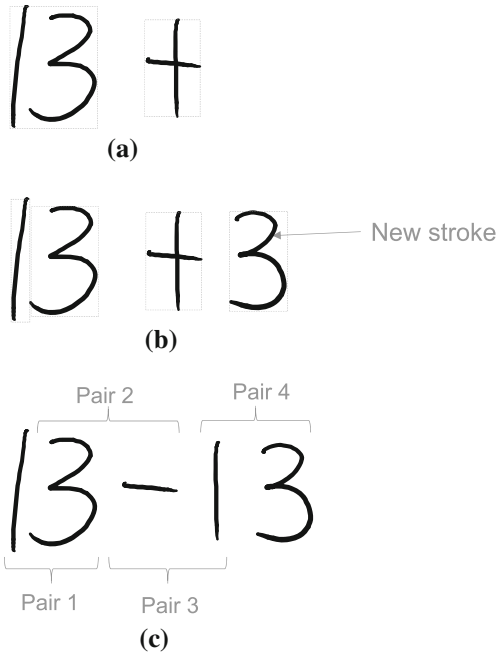


Fig. 4 Example of segmentation changes after inputting a new stroke. **a** Segmentation before stroke "3" is added. **b** Segmentation after stroke "3" is added. **c** List of strokes in writing order

If the segmentation threshold is too small, many updates need to be performed without recovering from the wrong segmentations. Therefore, we must select a suitable value for the segmentation threshold.

Figure 4 shows an example of the segmentation step in the augmented incremental recognition method. A user writes "13+" and then "3." The candidate "B+" has the highest segmentation probability before the new stroke for "3" is written. When the new stroke "3" is input, however, the segmentation probability of the first pair of strokes changes more than the segmentation threshold; hence, this probability needs to be updated with the result that the segmentation into "1" and "3" has a higher probability. For the second pair and the third pair, their segmentation probabilities change less than the threshold; hence, their probability values need not be updated. The last pair includes the new stroke, so we calculate the new segmentation probability for it.

3.4 Recognizing new symbol candidates

In the processing cycle, the result of the OHME recognition is updated up to the latest stroke. The maximum strokes that form a mathematical symbol are limited to five strokes. Consequently, the latest stroke may affect up to four previous strokes. All the new candidates that may form symbol patterns with the latest stroke are registered into the recognition table.

3.5 Updating scores of candidates

If a candidate is composed of a single stroke, its score is free from the geometric features and segmentation probability. If a candidate is composed of multiple strokes, however, its score is a combination of a recognition score and segmentation probabilities as shown in (10).

$$S_{cand} = \log\left(S_{recog}\right) + \sum_{i=k}^{k+m} \log\left(P_{prob}(stroke_i, stroke_{i+1})\right) \tag{10}$$

where

- S_{cand}: Score of a candidate.
- S_{recog}: Score of a recognition result.
- P_{prob}: Segmentation probability of 2 consecutive strokes.
- k: Index of the first stroke of a candidate.
- m: Number of strokes of a candidate.

If the segmentation probability changes, scores of the related candidates (S_{cand}) are updated immediately. Scores of the single-stroke candidates are not affected by (10); hence, their scores need not to be updated.

3.6 Adding new cells into CYK table

At this step, the geometric features, segmentations and scores of all the candidates are available, which are used as the input for the CYK algorithm. In the batch recognition [7], the CYK table is constructed in ascending order vertically from the bottom row to the top row. The lowest row is composed of terminal symbols. The lower rows must be completed before the upper rows are generated. On the contrary, the pure incremental recognition adds a cell on the right of each row, from the bottom row to the top row, in the CYK table whenever a new stroke is input. Figure 5 shows the difference of the pure incremental recognition method and the batch recognition method, where the candidate shown in each cell is the candidate with the highest score. Each cell also keeps other symbol candidates recognized for corresponding strokes. For each processing step, the latest result is extracted from the cell at the top of the CYK table.

3.7 Updating cells in CYK table

In the pure incremental recognition method, the CYK table is expanded without changing the existing cells whenever it receives a new stroke. Therefore, erroneous segmentations and recognitions in the previous strokes cannot be corrected.

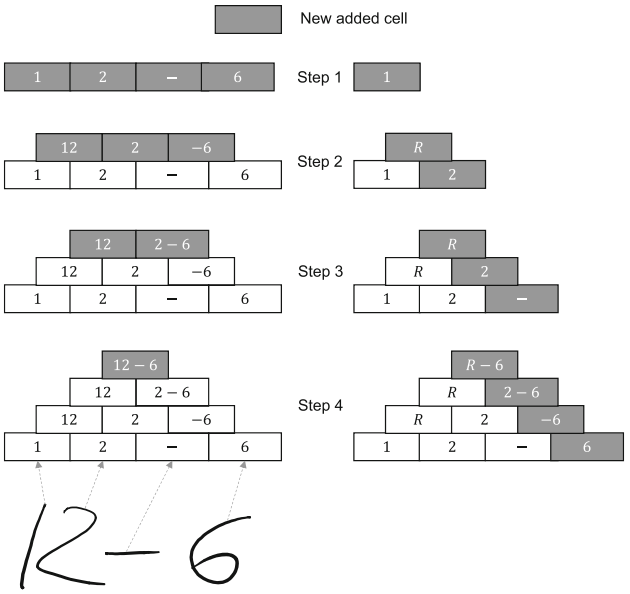


Fig. 5 An example of building CYK table in the batch recognition method (left) and the pure incremental method (right)

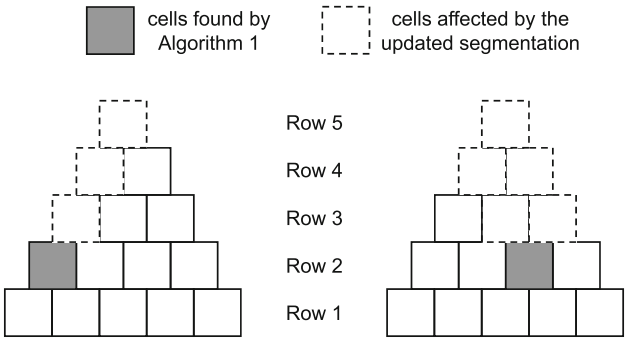


Fig. 6 Examples of cells that need to be updated with $p = 0$ (left figure) and $p = 2$ (right figure)

In the augmented incremental recognition method, cells are updated whenever segmentation probabilities and recognition scores are updated. Then, if a cell in row n is updated, all cells in row $(n+1)$ that connect directly with the updated cell in row n are updated. All cells in the first row that represent the recognition results of single strokes do not change. Only cells in the second and upper row may be updated.

Each updated segmentation affects its corresponding cell in the CYK table. Based on the position of the corresponding cells, we can find a list of cells that need to be updated according to the algorithm shown in Algorithm 1. Two examples of such cells are shown in Fig. 6; the gray cell is found by the algorithm in Algorithm 1, and all the cells in the dashed rectangle need to be updated. All the cells in this list are updated by the algorithm shown in Algorithm 2. The update is performed from bottom to top in the CYK table because the cells in lower rows are used to build the cells in upper rows.

Algorithm 1 To find cells to be updated

```
for i = 1 to n − 1 do                                   ▷ row 2 to n
    for j = (p − i + 1) to p do   ▷ cell order in a row (left to right)
                                  ▷ p is a position of updated segmentation
        if (−1 < j < n) then      ▷ check if a cell is in CYK table
            Add cell(i, j) into listForUpdate
```

Algorithm 2 To update cell(i, j)

```
for each cell(i,j) ∈ listForUpdate do
    if i < 5 then                                       ▷ lower than 5th row
        t ← group of strokes s_i, ..., s_{i+j}
        if t does not in rejecting invalid hypotheses then
            for each production X → a do
                if P_recog(t|a) > 0 then
                    Add node(X → a, P_recog(t|a)P_seg, t) into cell(i,j)
    for each X →(rel) AB do
        for k = 0 to i do                   ▷ % get two sub-cells of cell(i,j)
            C1 = getcell(k,j)
            C1 = getcell(i − k − 1, j + k + 2)
            prob = P(C1|A) × P(C2|B) × P_rel(C1, C2|r)
            × P_seg(s_{j+k}, s_{j+k+1}) × P_gram(X →(rel) AB)
            if prob > 0 then
                Add node(X →(rel) AB, prob, C1, C2) into cell(i, j)
    for each X → A do
        if cell(i,j) has node A then
            Add node(X → A, P(A)) into cell(i, j)
```

These two algorithms are evoked whenever segmentation is updated.

An update changes the scores of the candidates in each cell, after which these candidates are reordered by their scores. As a result, the new candidate with the highest score in each cell of the CYK table shows the result of the current OHME.

Like the complexity of the CYK algorithm from Le et al. [7], the complexity of this algorithm is still $O(n^3|P|)$, whereas the complexity of the algorithms proposed by Alvaro et al. [3] and Maclean et al. [15] are $O(n^3 log n|P|)$ and $O(n^4|P|)$, respectively, where n is the number of input strokes for an OHME and $|P|$ is the number of production rules in the grammar.

3.8 Examples of updating CYK table

Figure 7 illustrates an example of updating the CYK table in the augmented incremental recognition method. The parenthesized number on the bottom denotes the step of updating the CYK table after the latest stroke is input. The symbol or the OHME in each cell is the candidate with the highest score in that cell.

We assume that the user has been writing "13 + 3." Figure 7a shows the CYK table before the stroke "3" is input. When the system receives the stroke "3," the segmentation score between the strokes "1" and "3" are changed more than the segmentation threshold, so all the cells related to this seg-

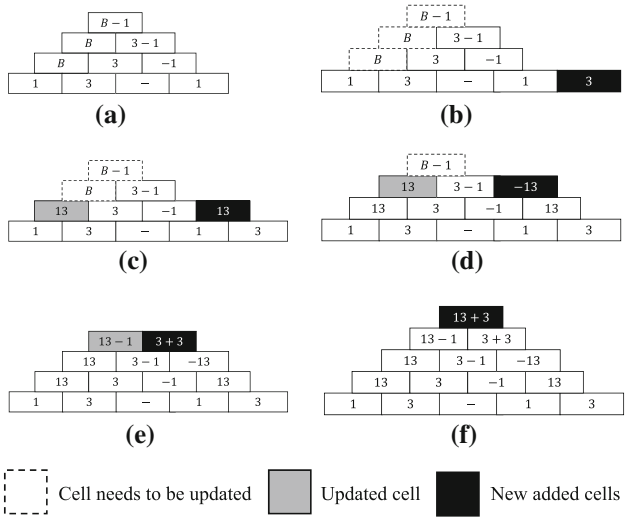


Fig. 7 Update of CYK table for "13 + 3" where (i, j) denotes position of the cell. **a** CYK table before adding stroke "3," **b** finding cells to be updated Added the new cell (0, 4), **c** updating the cell (1, 0) Adding the new cell (1, 3), **d** updating the cell (2, 0) Adding the new cell (2, 2), **e** updating the cell (3, 0) Adding the new cell (3, 1), **f** adding the new cell (4, 0)

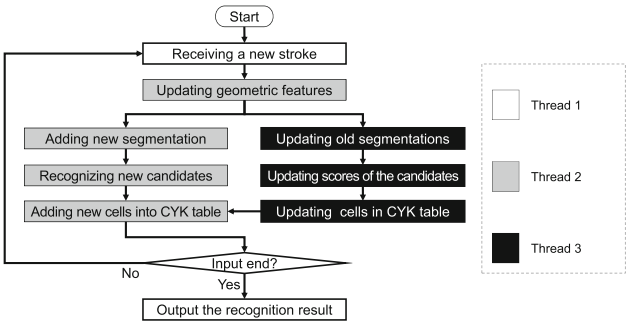


Fig. 8 Function arrangement on three threads

mentation, which are bound by red boxes in Fig. 7b, need to be updated. Secondly, Fig. 7c shows that the first cell in row 2 is updated. Consequently, the highest score candidate in this cell is changed from "B" to "13." Thirdly, the cell in row 3 which is connected directly to the cell "13" in row 2 is updated as shown in Fig. 7d. Similarly, the cells in row 4 and row 5 related to the cell "13" in the previous stage are updated as shown in Fig. 7e, f.

3.9 Multi-thread

Following the flow of recognition shown in Fig. 3, if a user is required to wait for writing a new stroke until the current recognition process is finished, it makes the user feel uncomfortable and unnatural. Therefore, to reduce the time delay in writing, the augmented incremental recognition is performed on three threads as shown in Fig. 8.

Thread 1 contains all the functions related to the interface. Thread 2 includes all the functions related to the pure incremental recognition process. Thread 3 covers all the functions that handle the previous strokes. In the previous method, Threads 2 and 3 must wait for each other to complete each task before going to the next task, which incurs unnecessary waiting time. However, we improved this by letting these two threads run independently in the update of the CYK table, so that Thread 2 need not wait for Thread 3. This optimization decreases the waiting time compared with the previous method. There are two advantages of dividing the system into three threads. First, the user can write an OHME smoothly without waiting, even when the system has not finished the current recognition process for the latest input stroke yet. Second, the overhead of organizing multiple threads is little and they decrease the waiting time generally by half (e.g., from 0.1 to 0.05 s when the number of strokes is 16 and from 0.24 to 0.12 s when it is 40) because the system can run the newly added functions and the old ones in parallel.

4 Experiments and discussions

4.1 Datasets

To evaluate our proposed incremental method, we conducted the following experiments employing Window 10 Professional on an Intel Core i7-3770 CPU of 3.40GHz with 8GB memory. We used Hands-Math dataset and the CROHME datasets. The Hands-Math dataset includes 10,864 OHMEs that have been collected from 62 elementary school children, 27 junior high school students and 26 members of our laboratory. We use 8266 OHMEs for training and 2598 OHMEs for testing. The number of symbol classes is ninety-four, including math symbols used in Japanese elementary and junior high schools such as numerals, operators, uppercase and lowercase letters and unit symbols.

The CROHME 2013 [27], CROHME 2014 [28] and CROHME 2016 [29] events were organized at ICDAR 2013, ICFHR 2014, and ICFHR 2016. They share the same training set, which contains 8,836 OHMEs, whereas the numbers in the testing sets are 671 OHMEs, 986 OHMEs and 1147 OHMEs in CROHME 2013, CROHME 2014 and CROHME 2016, respectively. The number of symbol classes is 101, including many similar symbols such as $\{1, |, l\}$, $\{P, p\}$, $\{S, s\}$, $\{C, c\}$, $\{X, x,\}$, $\{V, v\}$ and $\{O, o, 0\}$.

4.2 Experiments and discussions on the updated segmentations

We made the following three experiments. The results shown are measured for the testing set. The first experiment is on segmentation. When a new stroke is input, the segmenta-

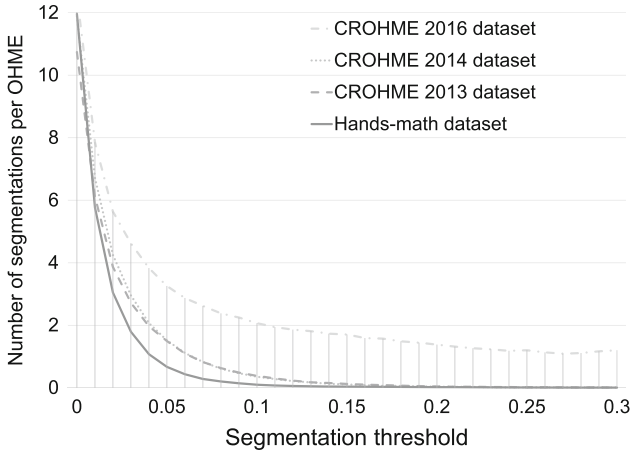


Fig. 9 Average number of updated segmentations per OHME

tions made so far are updated if one of their probabilities changes more than the segmentation threshold. Figure 9 shows the average number of updated segmentations per OHME throughout its incremental recognition depending on the segmentation threshold (T_s). In this experiment, Ts is varied from 0 to 0.3 in steps of 0.01. In all the three datasets, when $T_s = 0$, the average number of updated segmentations is more than 10 per OHME. As T_s is set higher, however, the number of updated segmentations decreases. This decrease is faster for the Hands-Math dataset than for the CROHME 2013, 2014 and 2016 datasets. We suggest the explanation for it. The OHMEs in Hands-Math dataset follow simple grammars for elementary and junior high schools so that the structural relations are not so rich and the heights of symbols within an OHME are not so different. On the other hand, the CROHME datasets cover complex grammars so that the structural relations are rich and the heights of symbols within an OHME vary widely; hence, the segmentations are more strongly affected by succeeding strokes. When T_s reaches 0.3, however, the average number of updated segmentations is asymptotic to 0% for Hands-Math, CROHME 2013 and CROHME 2014 datasets but to 1 for CROHME 2016.

4.3 Experiments and discussions on the recognition performance

The second experiment is to evaluate the whole system. We measured four factors. The first is the symbol detection ("Sym Seg" or symbol segmentation), the second is the segmentation of symbols with their correct classification ("Seg + Class"), the third is the correct relationship between all pairs of symbols ("RelTree," for Hands-Math, CROHME 2013, and CROHME 2014), plus correct segmentation of symbols ("Struc Rec," for CROHME 2016), and the fourth is the ME recognition rate ("Exp Rec"). We consider that an OHME is correctly recognized if all symbols, relations and its structure are recognized correctly. We also implemented a tandem recognition method, which recognizes first j strokes by the batch method and the later strokes by the pure incremental method. Tables 3, 4 and 6 show Sym Seg, Seg + Class, RelTree, Struc Rec and Exp Rec by the batch method, the pure incremental method, the tandem method and the augmented incremental method.

In the experiment on the three datasets, the augmented incremental recognition method records the Exp Rec as high as the batch recognition method when $T_s = 0.05$ and does not degrade much even when $T_s = [0.15, 0.30]$. It has a better Exp Rec than the tandem method when $T_s \leq 0.10$ and is always better than the pure incremental method in the range of $T_s = [0.00, 0.30]$. The pure incremental method has the lowest Exp Rec for the three datasets because the context information it can use is limited. The tandem method uses all the context information in the first j strokes, but in the remaining strokes it employs only a limited context; hence, its Exp Rec is higher than the pure incremental method but lower than the batch method. In the augmented incremental recognition, as the segmentation threshold is lowered, more segmentations are updated as shown in Fig. 9, with the result

Table 3 Recognition performance on Hands-Math dataset (%)

Method	Sym Seg	Seg+Class	RelTree	Exp rec
Batch	92.24	88.24	89.79	68.07
Augmented incremental (seg. threshold: T_s)				
0.05	92.46	88.24	89.79	68.07
0.10	92.39	88.22	89.79	67.92
0.15	92.32	88.09	89.72	67.86
0.20	92.24	88.01	89.68	67.78
0.25	92.19	87.92	89.60	67.71
0.30	92.15	87.78	89.58	67.68
Pure incremental	92.11	87.54	89.49	66.96
Tandem	92.34	88.17	89.72	67.89

Table 4 Recognition performance on CROHME 2013 dataset (%)

Method	Sym Seg	Seg+Class	RelTree	Exp Rec
Batch	88.49	79.63	71.09	32.34
Augmented incremental (seg. threshold: T_s)				
0.05	88.49	79.63	71.09	32.34
0.10	88.48	79.61	71.09	32.34
0.15	88.41	79.56	71.09	32.19
0.20	88.36	79.49	70.94	32.04
0.25	88.30	79.45	70.64	31.89
0.30	88.25	79.38	70.19	31.74
Pure incremental	88.22	79.33	69.30	31.45
Tandem	88.43	79.59	71.09	32.19

Table 5 Recognition performance on CROHME 2014 dataset (%)

Method	Sym Seg	Seg+Class	RelTree	Exp Rec
Batch	84.67	77.21	68.66	32.86
Augmented incremental (seg. threshold: T_s)				
0.05	84.67	77.21	68.66	32.86
0.10	84.65	77.18	68.66	32.86
0.15	84.62	77.11	68.46	32.75
0.20	84.58	77.05	68.26	32.66
0.25	84.53	76.95	68.05	32.56
0.30	84.49	76.89	67.65	32.56
Pure incremental	84.46	76.85	67.14	32.15
Tandem	84.59	77.06	68.26	32.66

Table 6 Recognition performance on CROHME 2016 dataset (%)

Method	Sym Seg	Seg+Class	Struc Rec	Exp Rec
Batch	91.62	86.05	61.55	43.94
Augmented incremental (seg. threshold: T_s)				
0.05	91.62	86.05	61.55	43.94
0.10	91.50	86.01	61.46	43.76
0.15	91.31	85.94	61.29	43.50
0.20	91.21	85.77	61.03	43.33
0.25	91.19	85.71	60.94	42.98
0.30	91.16	85.67	60.59	42.81
Pure incremental	91.11	85.59	59.81	42.28
Tandem	91.26	85.91	61.03	43.15

that the rate of Sym Seg increases as high as that in the batch recognition method. Lowering the segmentation threshold also increases the rates of Seg+Class and Struc Rec because their rates depend on Sym Seg. As a consequence, the candidates in the CYK table are updated closer to those in the CYK table of the batch recognition method. Therefore, the Exp Rec is maintained closer to the batch method (Table 5).

The results described in Tables 3, 4 and 6 demonstrate that the performance of the augmented incremental recognition method mainly depends on the segmentation. For StrucRec, only one wrong segmentation can make the whole relation tree wrong. Therefore, the system can recognize more symbols and relations correctly if the correct segmentations are recovered. Although the augmented incremental recognition method turns the batch recognition method employing segmentation and isolated symbol recognition to incremental and realize recognition performance as the same as the batch method, it cannot overcome the limitation in Sym Seg, Seg+Class, RelTree, Struc Rec and Exp Rec even if the threshold $T_s = 0$.

Although the commercial system by Myscript reports a much higher recognition rate in the CROHME competitions using a large set of roughly 30,000 training patterns rather than the CROHME training set and a large corpus [27], our system is the state of the art among academic systems [3,7,9]. Its performance on Hands-Math is better than on the CROHME datasets, since we can train the system with samples that need to be recognized. The difficulty of OHME recognition is that even misrecognition of a single symbol in an OHME causes misrecognition of the entire OHME. Thus, editing and correction functions are provided for OHME recognition systems in practice.

The most important point, however, is that the proposed method can turn the batch recognition method into an incremental recognition method. While the batch method can yield an improved recognition rate with more training patterns and a larger context, our proposed method provides an incremental system with almost the same recognition rate.

4.4 Experiments and discussions on the waiting time

The third experiment is to measure the average waiting time for recognizing an OHME by the augmented incremental method in comparison with the pure incremental method and the tandem method. A comparison with the batch recognition is discussed later because the difference is too large to show in the same graph.

In both the pure incremental and the augmented incremental methods, the waiting time for an OHME is not only affected by the latest stroke but also by the previous strokes if their processing was not completed before processing the latest stroke.

In the augmented incremental method, the waiting time for an OHME is modeled as follows. After a new stroke i is written, the thread 2 performs segmentation and the thread 3 performs parsing for stroke i, taking time $p2_i$ and $p3_i$, respectively, neglecting the small time for the thread 1. In addition, there is the elapsed time between two consecutive strokes. We call the elapsed time after writing stroke i as e_i. The waiting time for an OHME is calculated in (11).

$$T_{augm\_incremetal} = \sum_{i=1}^{n-1} f(\max(p2_i, p3_i) - e_i) + \max(p2_n, p3_n) \tag{11}$$

where $f(x) = x$ if $x > 0$ otherwise 0.

The complexity of $p2_i$ and that of $p3_i$ are $O(i^2|P|)$ and $O(i_c|P|)$, respectively, where i_c is the number of cells that need to be updated and it is much less than i^2.

Similarly, the waiting time of the pure incremental recognition method for an OHME is calculated as shown in (12).

$$T_{pure\_incremetal} = \sum_{i=1}^{n-1} f(p2_i - e_i) + p2_n \tag{12}$$

where $f(x) = x$ if $x > 0$ otherwise 0.

Since the four datasets employed for this study do not have time stamp for each stroke, we measured the average elapsed time between two strokes using another small dataset of 2520 handwritten OHMEs with the result of $e_i = 0.7862$ s where the elapsed time over 2 s is considered too large and excluded from the average. Although this dataset includes patterns from elementary school children and older adults, the elapsed time across all the participants is stable (a difference of less than 0.2 s), and we consider $e_i = 0.7862$ s to be reliable.

Figure 10 shows the average waiting time of the pure incremental method, the tandem method [14] and the augmented incremental method with 6 values of the segmentation threshold (T_s). In the pure incremental method, the average waiting time becomes larger as the number of strokes n increases, since $\max(p2_i, p3_i) > e_i$ for some i on an ordinary CPU, and the remaining processes for the previous strokes accumulate until the last stroke. Nevertheless, the average waiting time of the augmented incremental method is less than 0.1 s in most cases, which is acceptable as the instantaneous response limit. The worst case of 0.35 s occurs when $T_s = 0.05$ with more than 35 input strokes, but it is still within the uninterrupted response limit of 1 [30] or 2 s [31,32]. The waiting time of the batch recognition method is also less than 2 s but exceeds 2 s when the number of strokes is increased. Moreover, when the program is run on a low-performance CPU, the waiting time exceeds 2 s even before the number of strokes reaches 40.

The tandem method has a noticeable delay (> 0.1 s) at the beginning of writing an OHME and subjects users to unnatural pauses. The pure incremental method shows the best performance with respect to the waiting time, but its superiority to the augmented incremental method is very small for OHMEs in Hands-Math, and even for high-level OHMEs in the CROHME datasets, as long as n is less than 30 or $T_s > 0.10$.

For example, assume that a user inputs an OHME composed of 30 strokes, the elapsed time between the 29th stroke and the 30th stroke is 0.7 s, the time to process the previous strokes is 0.9 s after the 29th stroke is input and that to process the 30th stroke is 0.3 s When the user completes the 30th stroke, the system needs 0.2 s to finish processing the previous process before executing the 30th stroke and 0.3 s for the last stroke. Therefore, the waiting time for this OHME is the remaining processing time of the previous strokes and the processing time for the 30th stroke, i.e., 0.2 + 0.3 s

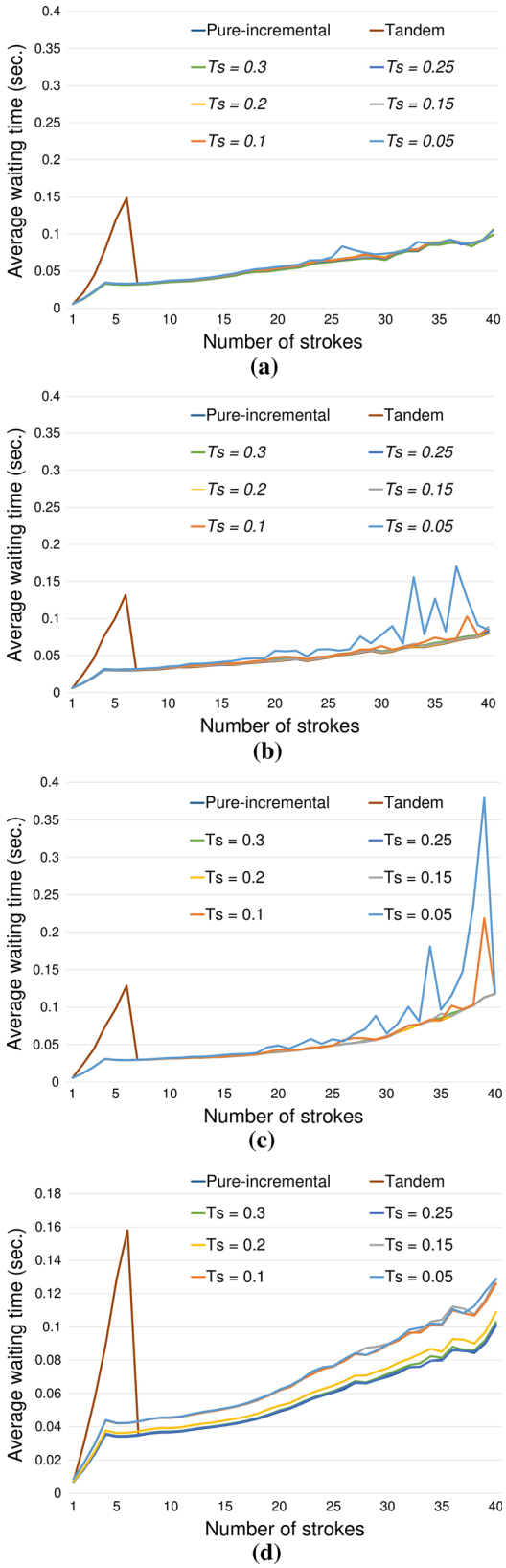


Fig. 10 Average waiting time of the incremental method and the tandem method. **a** Hands-Math dataset, **b** CROHME 2013 dataset, **c** CROHME 2014 dataset, **d** CROHME 2016 dataset

In the experiment of the average waiting time which is shown in Fig. 10, the average waiting time generally increases when the number of strokes increases. However, some unusual cases can happen. For example in CROHME 2014 dataset with $T_s = 0.05$, the average waiting time deeply drops when the number of strokes increases from 39 to 40. The first reason is that the OHMEs of 39 strokes have many updated segmentations and updated cells in the CYK table than those of 40 strokes so that it need more time to recognize the OHMEs. Second reason is that the number of OHMEs that have 39 or 40 strokes is 4 and 2, respectively, so that the average is affected by one or two OHMEs, which require large processing time. The similar cases happened in the experiments in which the average waiting time decreases, while the number of stroke increases.

The next experiment is to compare the average waiting time of the augmented incremental method with that of the batch method. We modeled the waiting time of the augmented incremental recognition in the previous experiment. On the other hand, the waiting time in the batch method is trivially the time to recognize a whole OHME.

Figure 11 shows the average waiting time of the batch method and the augmented incremental method with $T_s = 0.05$ which has the worst average waiting time in the third experiment. The waiting time of the batch method becomes longer as the number of strokes n increases. The average waiting time of the batch method crosses the instantaneous response limit (0.1 s) even when n is less than 5, and the uninterrupted response limit of 1 s [32] when $n = 25$ or more from the user interface point of view.

From these experiment, we can see that the augmented incremental method is comparable with the pure incremental recognition method, and it is superior to the tandem method and the batch method significantly with respect to the average waiting time.

Consequently, we can conclude from all the experiments that the augmented incremental recognition method excels the batch method, the pure incremental method and the tandem method in the recognition rate and the average waiting time.

We would like to make an additional remark regarding the recognition rate: it is not so high as the online handwritten text recognition rate [33,34]. This is because the language context of mathematical expressions is weaker than that of natural languages, and the two-dimensional structure of mathematical expressions is more difficult than the one-dimensional structure of the text. The most important reason, however, is due to the condition that an OHME is correctly recognized if all the symbols, all the relations and its structure as a mathematical expression are recognized correctly. At this moment, the OHME recognition systems are being deployed for primary or junior high school students to learn math on a tablet. The recognition rate must be improved, however, for high

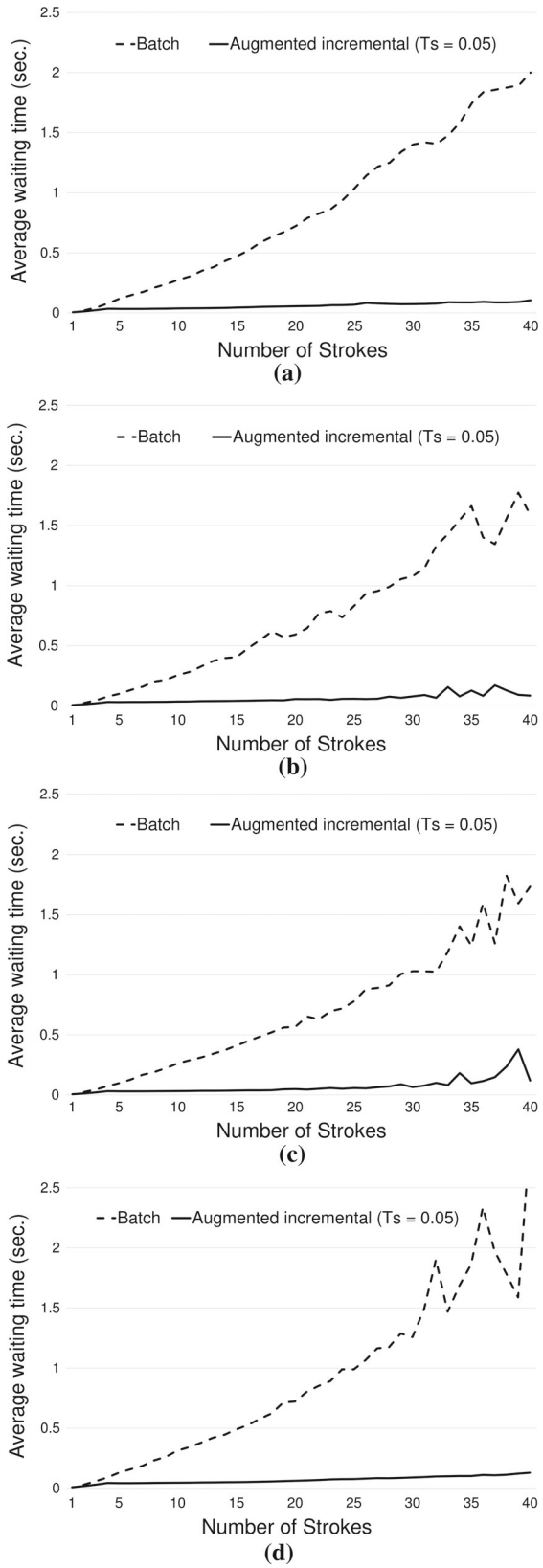
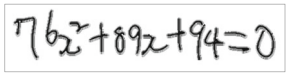


Fig. 11 Average waiting time of the batch method and the augmented incremental method. **a** Hands-Math dataset, **b** CROHME 2013 dataset, **c** CROHME 2014 dataset, **d** CROHME 2016 dataset

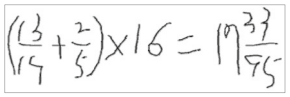
$51 + 89 = {}^{1}40$

(a)



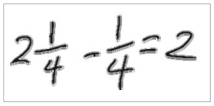
$7678 + 89x + 94 = 0$

(b)



$\frac{13}{14} + \frac{2}{5} \times 16 = 19\frac{33}{95}$

(c)



$2\frac{1}{4}\frac{1}{-4} = 2$

(d)

Fig. 12 Examples of correctly and incorrectly recognized samples with their ground truth in the datasets. **a** Ground truth as $51 + 89 =^{1} 40$, **b** ground truth as $76x^2 + 89x + 98 = 0$, **c** ground truth as $\frac{13}{19} + \frac{2}{5} \times 16 = 19\frac{33}{95}$, **d** ground truth as $2\frac{1}{4} - \frac{1}{4} = 2$

school students, university students and scientists to use the OHME recognition.

Although the underlying OHME methods must be improved in their recognition rates, the method of augmented incremental recognition can be applied to the existing OHME recognition methods to minimize the waiting time without sacrificing their recognition rates.

4.5 Analysis of results and future works

Correctly and incorrectly recognized samples in the datasets are shown in Fig. 12. Samples in Fig. 12a, b are recognized incorrectly by the pure incremental method but correctly by the augmented incremental method. The gray rectangles located the errors: wrong relation analysis in Fig. 12a (recognized as $51 + 89 = 140$) and wrong symbol recognition in Fig. 12b (recognized as $7678 + 89x + 98 = 0$). Figure 12c, d shows misrecognized samples by all of our recognition methods due to wrong relation analysis (recognized as $2\frac{1}{4}.\frac{1}{-4} = 2$) and a wrong symbol recognition (recognized as $\frac{13}{14} + \frac{2}{5} \times 16 = 19\frac{33}{95}$), respectively.

The misrecognized OHMEs show that both of the augmented incremental recognition method and the batch recognition method are limited for them. First, our method cannot work well for OHMEs containing interspersed strokes. Second, the current method misrecognizes OHMEs whose segmentation, symbol recognition or structural relations are ambiguous although the method keeps multiple candidates. Third, the structure analysis is the key task degrading the recognition rate. This task includes the 2D-SCFG which is defined manually and does not cover all the contexts in a huge variety of handwritings. The performance of the augmented incremental recognition method could be improved by that of the underlying batch method through gathering more context information from OHMEs and training the system using more OHME patterns with a large ME corpus. Stroke order-free parsing [12,13] is another candidate to cope with interspersed strokes.

5 Conclusion

We have presented an augmented incremental recognition method for online handwritten mathematical expressions. By updating segmentations and recognitions in the CYK table after receiving every new stroke, augmented incremental recognition method not only achieves a recognition rate as high as the batch recognition method, but also reduces the average waiting time to the same degree as the pure incremental recognition method. This has been confirmed by the experiments made on the Hands-Math dataset and the CROHME datasets. The control variable of segmentation threshold should be set according to the application environment.

This augmented incremental recognition method is a stroke order-dependent method so that it is still limited to presumed order of strokes. In order to extend this approach, we are developing a stroke order-free method based on the augmented incremental recognition approach.

Appendix

See Tables 7, 8 and Fig. 13.

Table 7 Geometric features for symbol segmentation

Feature	Definition	Feature	Definition
F1	$nb_x(S_p, S_s)/\overline{h}$	F12	$w\_U(B_p, B_s)/h(B_s)$
F2	$nb_y(S_p, S_s)/\overline{h}$	F13	$w\_O(B_p, B_s) \times h\_O(B_p, B_s)$
F3	$D_x(B_p, B_s)/\overline{h}$	F14	$(y_c(B_p) - y_c(B_s))/h\_O(B_p, B_s)$
F4	$D_y(B_p, B_s)/\overline{h}$	F15	$h(B_p)/h(B_s)$
F5	$x_m(B_p, B_s)/\overline{h}$	F16	$(x_l(S_s) - x_l(S_p))/w(B_p)$
F6	$y_m(B_p, B_s)/\overline{h}$	F17	$(x_r(S_s) - x_r(S_p))/w(B_s)$
F7	$d_x(S_s)/\overline{h}$	F18	$w(B_p)/w\_U(B_p, B_s)$
F8	$d_y(S_s)/\overline{h}$	F19	$w(B_s)/w\_U(B_p, B_s)$
F9	$w\_O(B_p, B_s)/w(B_p)$	F20	$h(B_p)/h\_U(B_p, B_s)$
F10	$w\_U(B_p, B_s)/h(B_p)$	F21	$h(B_s)/h\_U(B_p, B_s)$
F11	$w\_O(B_p, B_s)/w(B_s)$		

Table 8 Terms for features

Symbol	Definition
$\overline{h}$	Average height of all input strokes
S_p	Immediate preceding stroke
S_s	Immediate succeeding stroke
B_p	Bounding box of S_p with 4 positions ($y_t(S_p)$, $y_b(S_p)$, $x_r(S_p)$ and $x_l(S_p)$)
B_s	Bounding box of S_s with 4 positions ($y_t(S_s)$, $y_b(S_s)$, $x_r(S_s)$ and $x_l(S_s)$)
$nB_x(B_p, B_s)$	Projection of the nearest bridge between S_p and S_s to x-axis
$nB_y(B_p, B_s)$	Projection of the nearest bridge between S_p and S_s to y-axis
$D_x(B_p, B_s)$	Absolute distance between B_p and B_s on x-axis
$D_y(B_p, B_s)$	Absolute distance between B_p and B_s on y-axis
$x_m(B_p, B_s)$	Midpoint between centers of B_p and B_s on x-axis
$y_m(B_p, B_s)$	Midpoint between centers of B_p and B_s on y-axis
$d_x(S_s)$	Distance between the first point and the last point of S_s on x-axis
$d_y(S_s)$	Distance between the first point and the last point of S_s on y-axis
$w\_O(B_p, B_s)$	Width of overlap of B_p and B_s: $\min(x_r(S_p), x_r(S_s)) - \max(x_l(S_p), x_l(S_s))$
$w\_U(B_p, B_s)$	Width of union of B_p and B_s: $\max(x_r(S_p), x_r(S_s)) - \min(x_l(S_p), x_l(S_s))$
$h\_O(B_p, B_s)$	Height of overlap of B_p and B_s: $\min(y_b(S_p), y_b(S_s)) - \max(y_t(S_p), y_t(S_s))$
$h\_U(B_p, B_s)$	Height of union of B_p and B_s: $\max(y_b(S_p), y_b(S_s)) - \min(y_t(S_p), y_t(S_s))$
$y_c(B_p)$	Center of B_p on y-axis
$y_c(B_s)$	Center of B_s on y-axis
$w(B_p)$	Width of B_p
$h(B_p)$	Height of B_p
$w(B_s)$	Width of B_s
$h(B_s)$	Height of B_s

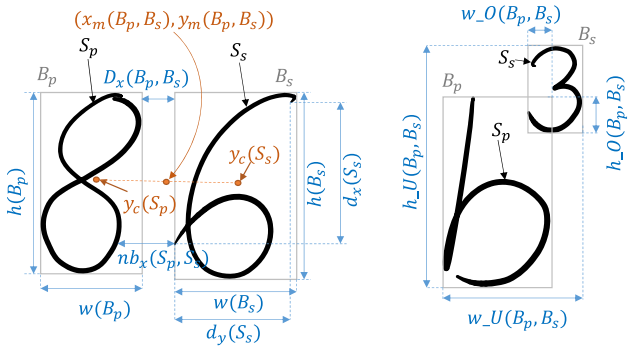


Fig. 13 Extracted features listed in Table 8

References

1. Anderson, R.H.: Syntax-directed recognition of hand-printed two-dimensional mathematics. In: Symposium on Interactive Systems for Experimental Applied Mathematics: Proceedings of the Association for Computing Machinery Inc. Symposium, pp. 436-459, Washington, USA (1967)
2. Chang, S.K.: A method for the structural analysis of two-dimensional mathematical expressions. Int. J. Inf. Sci. **2**(3), 253–272 (1970)
3. Ivaro, F., Snchez, Bened, J.: Recognition of on-line handwritten mathematical expressions using 2D stochastic context-free grammars and hidden Markov models. Pattern Recognit. Lett. **31**, 58–67 (2014)
4. Awal, A.M., Mouchre, H., Viard-Gaudin, C.: A global learning approach for an online handwritten mathematical expression recognition system. Pattern Recognit. Lett. **35**, 68–77 (2012)
5. Yamamoto, R., Sako, S., Nishimoto, T., Sagayama, S.: Online recognition of handwritten mathematical expressions based on stroke-based stochastic context-free grammar. In: Proceedings of the 7th International Conference on Document Analysis and Recognition, vol. 1, pp. 249-254, Edinburgh, UK (2006)
6. Simistira, F., Katsouros, V., Carayannis, G.: Recognition of online handwritten mathematical formulas using probabilistic SVMs and stochastic context free grammars. Pattern Recognit. Lett. **53**, 85–92 (2015)
7. Le, A.D., Nakagawa, M.: A system for recognizing online handwritten mathematical expressions by using improved structural analysis. Int. J. Doc. Anal. Recognit. **19**(4), 305–319 (2016)
8. Okamoto, M., Miao, B.: Recognition of mathematical expressions by using the layout structure of symbols. In: Proceeding of the 1st International Conference on Document Analysis and Recognition, pp. 242–250, Saint-Malo, France (1991)
9. Zanibbi, R., Blostein, D., Cordy, J.R.: Recognizing mathematical expressions using tree transformation. IEEE Trans. Pattern Anal. Mach. Intell. **24**(11), 1455–1467 (2002)
10. Hu, L., Zanibbi, R.: MST-based Visual Parsing of Online Handwritten Mathematical Expressions. In: Proceedings of the 15th International Conference on Frontiers in Handwriting Recognition, pp. 337–342, Shenzhen, China (2016)
11. Julca-Aguilar, F., Mouchre, H., Viard-Gaudin, C., Hirata, N.S.T.: Top-Down Online Handwritten Mathematical Expression Parsing with Graph Grammar. In: Proceedings of the 20th Iberoamerican Congress on Pattern Recognition, pp. 444–451, Montevideo, Uruguay (2015)

12. Garain, U., Chaudhuri, B.B.: Recognition of online handwritten mathematical expressions. IEEE Trans. Syst. Man Cybern. B Cybern. **34**(6), 2366–2376 (2004)
13. Le, A.D., Nakagawa, M.: A system for recognizing online handwritten mathematical expressions by using improved structural analysis. Int. J. Doc. Anal. Recognit. **19**(4), 305–319 (2016)
14. Zhang, J., Du, J., Zhang, S., Liu, D., Hu, Y., Hu, J., Wei, S., Dai, L.: Watch, attend and parse: an end-to-end neural network based approach to handwritten mathematical expression recognition. Pattern Recognit. Lett. **71**, 196–206 (2017)
15. MacLean, S., Labahn, G.: A new approach for recognizing handwritten mathematics using relational grammars and fuzzy sets. Int. J. Doc. Anal. Recognit. **16**(2), 139–163 (2013)
16. Unger, S.H.: A global parser for context-free phrase structure grammars. Commun. ACM **11**(4), 240–247 (1968)
17. Predovic, G., Abdulkader, A., Dresevic, B., Viola, P. A, Vukosavljevic, M.: Recognition of mathematical expressions. U.S. Patent US8009915 B2 (2011)
18. Vuong, B.Q., Hui, S.C., He, Y.: Progressive structural analysis for dynamic recognition of on-line handwritten mathematical expressions. Pattern Recognit. Lett. **29**(5), 647–655 (2008)
19. Phan, K.M., Nguyen, C.T., Le, A.D., Nakagawa, M.: An incremental recognition method for online handwritten mathematical expressions. In: Proceedings of the 3rd IAPR Asian Conference on Pattern Recognition, pp. 171–175, Kuala Lumpur, Malaysia (2015)
20. Phan, K.M., Le, A.D., Nakagawa, M.: Semi-incremental recognition of online handwritten mathematical expressions. In: Proceedings of the 15th International Conference on Frontiers in Handwriting Recognition, pp. 258–264, Shenzhen, China (2016)
21. Nakagawa, M., Machii, K., Kato, N., Souya, T.: Lazy recognition as a principle of pen interfaces. In: Proceedings of the ACM INTERCHI, pp. 89–90, Amsterdam, Netherlands (1993)
22. Le, A.D., Nakagawa, M.: Comparison of parsing algorithms for recognizing online handwritten mathematical expressions. In: Proceedings of the 15th International Conference on Frontiers in Handwriting Recognition, pp. 390–394, Shenzhen, China (2016)
23. Cocke, J., Schwartz, J.T.: Programming Languages and Their Compilers: Preliminary Notes, 2nd edn. Courant Institute of Mathematical Sciences, New York (1970)
24. Younger, D.H.: Recognition and parsing of context-free languages in time n^3. Inf. Comput. **10**(2), 189–208 (1967)
25. Kasami, T.: An Efficient Recognition and Syntax-Analysis Algorithm for Context-Free Languages. University of Illinois Coordinated Science Laboratory, Amsterdam (1966)
26. Zhu, B., Gao, J., Nakagawa, M.: Objective function design for MCE-based combination of on-line and off-line character recognizers for on-line handwritten Japanese text recognition. In: Proceedings of the 11th International Conference on Document Analysis and Recognition, pp. 594–599, Beijing, China (2011)
27. Mouchre, H., Viard-Gaudin, C., Zanibbi, R., Garain, U., Kim, D.H., Kim, J.H.: ICDAR 2013 CROHME: third international competition on recognition of online handwritten mathematical expressions. In: Proceedings of the 12th International Conference on Document Analysis and Recognition, pp. 1428–1432, Washington, USA (2013)
28. Mouchre, H., Viard-Gaudin, C., Zanibbi, R., Garain, U.: ICFHR 2014 competition on recognition of on-line handwritten mathematical expressions (CROHME 2014). In: Proceedings of the 14th International Conference on Frontiers in Handwriting Recognition, pp. 791–796, Heraklion, Greece (2014)
29. Mouchre, H., Viard-Gaudin, C., Zanibbi, R., Garain, U.: ICFHR2016 CROHME: competition on recognition of online handwritten mathematical expressions. In: Proceedings of the 15th International Conference on Frontiers in Handwriting Recognition, pp. 607–612, Shenzhen, China (2016)
30. Nielsen, J.: Usability Engineering. Academic Press Inc, Boston (1993)
31. Shneiderman, B.: Response time and display rate in human performance with computer. ACM Comput. Surv. **16**(3), 265–285 (1984)
32. Miller, R.B.: Response time in man-computer conversational transactions. In: Proceeding of the AFIPS Fall Joint Computer Conference, vol. 33, pp. 267–277, San Francisco, USA (1968)
33. Liu, C.L., Jaeger, S., Nakagawa, M.: Online recognition of Chinese characters: the state-of-the-art. IEEE Trans. Pattern Anal. Mach. Intell. **26**(2), 198–213 (2004)
34. Plamondon, R., Srihari, S.N.: On-line and off-line handwriting recognition: a comprehensive survey. IEEE Trans. Pattern Anal. Mach. Intell. **22**(1), 63–84 (2000)