Reducing Annotation Workload Using a Codebook Mapping and its Evaluation in On-Line Handwriting

Jinpeng Li, Harold Mouchère, Christian Viard-Gaudin
IRCCyN (UMR CNRS 6597) - L'UNAM - Université de Nantes, France
{*jinpeng.li , harold.mouchere , christian.viard-gaudin*}*@univ-nantes.fr*

***Abstract*—The training of most of the existing recognition systems requires availability of large datasets labeled at the symbol level. However, producing ground-truth datasets is a tedious work. Two repetitive tasks have to be chained. One is to select a subset of strokes that belong to the same symbol, a next step is to assign a label to this stroke group. In this paper, we discuss a framework to reduce the human workload for labeling at the symbol level a large set of documents based on any graphical language. A hierarchical clustering is used to produce a codebook with one or several strokes per symbol, which is used for a mapping on the raw handwritten data. Evaluation is proposed on two different datasets.**

***Keywords*-On-Line Handwriting; Modified Hausdorff Distance; Symbol Annotation; Hierarchical Clustering;**

I. Introduction

Many existing recognition systems [11] require the definition of the character or symbol set, and rely on a training dataset which defines the ground-truth at the symbol level. Such datasets are essential for the training, evaluation, and testing stages of the recognition systems. However, collecting all the ink samples and labeling them at the symbol level is a very long and tedious task. Hence, it would be very interesting to be able to assist this process, so that most of the tedious work can be done automatically, and that only a high-level supervision needs to be done to conclude the labeling process.

We can divide such process into two steps, (a) segmenting handwritten scripts into symbols using an unsupervised symbol extraction method [7], [8], and (b) grouping them into a codebook in which a user can label symbols in order to reduce the human effort. This paper is limited to the second step: the codebook generation, annotation and its assessment. An offline handwriting annotation system [13] proposes a similar idea to label a large number of well segmented isolated characters; clustering them into several clusters of characters, and labeling the clusters in order to reduce human effort.

Let us show an example to introduce the problem. Fig. 1 considers an example of a graphical language. For clarity, all the strokes are indexed "$(.)$". Fig. 2a displays the correct segmentation into symbols. Dashed rectangles represent the proposed segments. In an ideal case, each segment contains exactly one graphical symbol. Then, according to their shapes, we group the segments in clusters. The corresponding clusters are shown in Fig. 2b. Choosing a pattern representative of each cluster yields a visual codebook used by a human to be labeled as (Fig. 2c) "4", "+", "=", and "8" respectively. Strokes in the pattern representative are marked by the index "$(*.)$". In this paper, we choose, as the pattern representative, the segment which minimizes the sum of distances to the other segments in the same cluster. Hence, we can label the handwritten scripts at the codebook level (the high level supervision) from a perfect symbol segmentation.

However, generating the perfect segmentation (each segment being precisely composed of a symbol) is far from being trivial [7], [8]. For instance, if we assume that segmentation is based on an unsupervised learning scheme to extract frequent patterns, then some segments that contain a symbol plus sub-parts of another symbol, or even several symbols (multi-symbols) will be produced.

Similarly, if the segmentation is based on the connected strokes as displayed in the example of Fig. 3a, the same problem of multi-symbol segment will be present. In that case, the cluster *C3* contains a digit "4" and a sub-part of "=", while the cluster *C2* contains two symbols, "4" and "+". A user can separate the symbols, and then label them in the visual codebook (Fig. 3c). The cluster *C3* can be labeled as "4-". If we cannot recognize a sub-part of symbol "-", the user can leave it unlabeled. In addition, a multi-symbol mapping problem will be studied, e.g. the cluster *C2* mapping.

After mapping the pattern representatives to the raw handwritten scripts with the codebook, some mistakes will be present. For example, we can find that the label "-" (minus) is wrong and we have to correct it. Thus, we also propose a criterion that measures how much work has been reduced. This criterion assesses the workload at the stroke level since in a manual labeling process, basically ink is manipulated at the stroke level.

In this paper, we introduce the proposed strategy for reducing workload on symbol labeling in Section II. The codebook generation, its mapping and assessment are presented in Sections III, IV, and V respectively. The experiment results and the conclusion will be given in Section VI and in Section VII.

978-0-7695-4774-9/12 $26.00 © 2012 IEEE
DOI 10.1109/ICFHR.2012.259

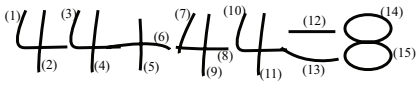
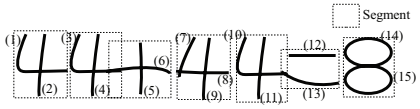
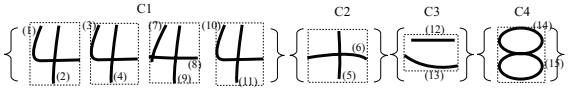


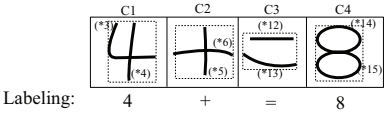
Figure 1: A raw handwritten expression



(a) Well segmented handwritten symbols to be labeled in the expression (Fig. 1)

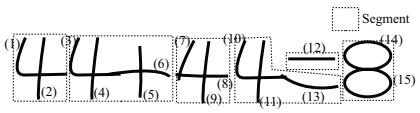


(b) Grouping the segments in clusters

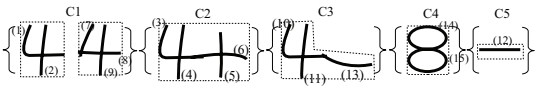


(c) Visual codebook for the user labeling

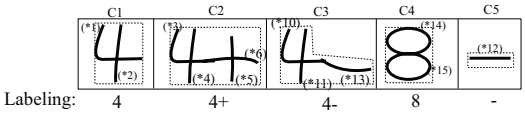
Figure 2: Reducing the human labeling workload in on-line handwriting graphical language: the perfect case.



(a) A connected-stroke segmentation in the expression (Fig. 1)



(b) Grouping the segments in clusters



(c) Visual codebook for the user labeling

Figure 3: A connected-stroke segmentation and its labeling

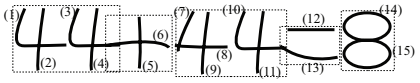


Figure 4: Merging the top-1 frequent bigram in Fig. 1

II. Overview

First of all, we introduce an overview of our annotation system in Fig. 5. The system is divided into three main steps: generating the segmentation (segments), clustering the segments and producing the codebook (different segment shapes), and codebook mapping from the user labeled codebook to the raw data.

In the first step, we need to generate a segmentation in order to apply our mapping procedure. Three different segmentations are used in this paper. The first segmentation is user defined and corresponds to the ground-truth, i.e. the perfect segmentation. To study the ability of our algorithm to deal with multi-symbol segments, we produced an under-segmentation by merging the top-n frequent bigrams at the symbol level. This can be done easily with the "Calculate" dataset (presented in Section VI-A) where symbols can be ordered from left to right. For instance, top-1 frequent bigram in Fig. 1 is "44", and Fig. 4 shows a segmentation by merging "44" at the symbol level.

A third segmentation is considered, it relies on the connected strokes to define a segment (like in Fig. 3a). Using these three segmentations, we will generate three different codebooks in the next section and then use them for the labeling stage.

III. Codebook Generation using Hierarchical Clustering

In this section, we generate a codebook from the ready-made segmentation using a hierarchical clustering. Each segment may contain several strokes. In addition, because of the nature of on-line handwriting, two instances of the same symbol can be drawn with a different number of strokes, a different stroke order and different stroke orientations. To overcome this problem, we propose to use a modified Hausdorff distance [3], [4] as initially introduced in image processing. Thus we consider each segment as a set of points, $seg = \{pt\}$. For being size independent, all the segments should be normalized into a reference bounding box $\{x \in [-1, 1], y \in [-1, 1]\}$ by keeping the ratio, and re-sampled into a fixed number of $n_{pt} = 100$ points. In addition to the raw data (x, y) , we used the local direction (sine, cosine) and the local curvature (cosine) to have a 5-feature local description of a point. The modified Hausdorff distance [4] between two segments (seg_a and seg_b) is defined by:

$$MHD_{seg}(seg_a, seg_b) = \frac{1}{2n_{pt}}(subhauf(seg_a, seg_b) + subhauf(seg_b, seg_a)) \quad (1)$$

where

$$subhauf(seg_a, seg_b) = \sum_{pt_i \in seg_a} \min_{pt_j \in seg_b} (dist(pt_i, pt_j)) \quad (2)$$

and $dist(pt_i, pt_j)$ is the Euclidean distance between two points, computed in the 5-dimension feature space introduced previously.

A clustering technique is used for producing the codebook, which is then brought into play for computing the

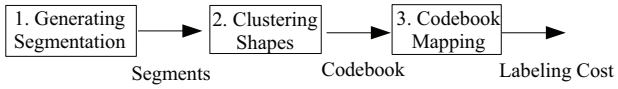


Figure 5: Three main steps on the annotation system

membership of each segment. It exists many clustering methods, hierarchical clustering [6], k-means [12], self-organizing map [5], neural gas [9], etc. We have chosen an agglomerative hierarchical clustering [6] since it only needs a distance matrix between all the segments; the other methods require to embed a segment in a feature space of fixed-number dimensions beforehand. We use the Lance-Williams formula [6] which provides an efficient computational algorithm for hierarchical clustering. The membership of each segment is then generated: all the segments are grouped into n_p clusters.

We select, as the pattern representative, the sample seg_c which minimizes the sum of modified Hausdorff distances to the other samples of the same cluster C:

$$seg_c = \arg\min_{seg_p \in C} \Big(\sum_{seg_q \in C} MHD_{seg}(seg_p, seg_q) \Big). \quad (3)$$

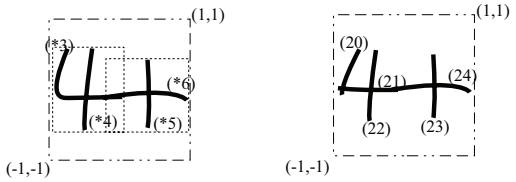
The pattern representatives will be organized as a visual codebook, an example is displayed in Fig. 2c. In the next section, the codebook mapping problem will be discussed.

IV. Codebook Mapping from a Visual Codebook to Raw Scripts

In the previous section, a codebook composed of multi-stroke segments has been obtained. A representative sample has been selected from each cluster to generate a visual codebook. A user labels therefore these chosen segments stroke by stroke in the visual codebook. In this section, we discuss how to label raw scripts with the labeled visual codebook.

In the visual codebook, segments in a cluster are not always from the same single symbol, e.g. "4+" in Fig. 3c represent more than one symbol. If we meet unknown symbols (sub-parts of symbol), we can leave them unlabeled. This task of segmentation and partial labeling of the representatives is quite simple. A mapping algorithm has been developed to complete the labeling of all unlabeled strokes in the original cluster. The mapping procedure involves the normalization of a segment into a bounding box, and then searching for all unlabeled strokes with the closest labeled stroke using modified Hausdorff distance. After this mapping process, the symbols are segmented and labeled.

With the example of the cluster C2 and the visual codebook given in Fig. 3c, we assume a new instance of two symbols "4+" in Fig. 6b for better explanation. This instance contains one more stroke (5 instead of 4), and belongs to the cluster C2. A user has first to manually label the two symbols contained in the representative of the C2 cluster, i.e. "4" for strokes (3, 4) and "+" for strokes (5, 6), as displayed in Tab. Ia. In our system, each stroke is associated with a symbol index and its label. The symbol index denotes the symbol the stroke belongs to.



(a) User Manually Labeling in **C2** **(b)** Raw Scripts

Figure 6: The user manually labels the cluster C2 (a), and then the system finds a mapping for raw scripts (b).

Then we have to automatically label the remaining strokes (20 to 24) belonging to C2. This is done by a mapping procedure to find the best match between the unlabeled strokes and the labeled ones. Considering two segments $\{(*3), (*4), (*5), (*6)\}$ and $\{(20), (21), (22), (23), (24)\}$, Tab. I shows the mapping procedure which normalizes the segments and looks for the corresponding labeled stroke. The numbers of strokes between two mapping segments are not necessarily equal. The mapping pairs $\{\{(20) \rightarrow (*3)\}, \{(21) \rightarrow (*3)\}, \{(22) \rightarrow (*4)\}, \{(23) \rightarrow (*5)\}, \{(24) \rightarrow (*6)\}\}$ are achieved. The symbol "4" $\{\{(20) \rightarrow (*3)\}, \{(21) \rightarrow (*3)\}, \{(22) \rightarrow (*4)\}\}$ and the symbol "+" $\{\{(23) \rightarrow (*5)\}, \{(24) \rightarrow (*6)\}\}$ are segmented and labeled.

Table I: Each stroke in raw segment (b) is given the label contained in its closest stroke of labeled representative (a).

Str	Sym	Label
(*3)	1	4
(*4)	1	4
(*5)	2	+
(*6)	2	+

Str: stroke index
Sym: symbol index

Str	Sym	Label
(20)	1	4
(21)	1	4
(22)	1	4
(23)	2	+
(24)	2	+

(a) The representative labeled by the user **(b)** A raw segment in the cluster

In the next section, we introduce the labeling cost to evaluate how much annotation work has been reduced.

V. Labeling Cost

In the previous section, the visual codebook was manually labeled. We then execute the mapping procedure described in the previous section to label all the other segments. Since the user labels the segments and raw handwritten scripts in a dataset stroke by stroke, we define the labeling cost C_{label} at the stroke level by:

$$C_{label} = \frac{N_c + N_{db} - N_{correct}}{N_{db}}, \quad (4)$$

where N_c is the number of strokes in the proposed codebook, N_{db} is the number of strokes in the dataset, and $N_{correct}$ is the number of strokes which are correctly labeled in the original dataset. $N_{db} - N_{correct}$ is the number of strokes for which the label has to be corrected or filled in the original dataset. N_c and N_{db} can be easily obtained by counting how many strokes are in the codebook and dataset respectively. We compute $N_{correct}$ according to the number of strokes which correspond to well segmented and well labeled symbols. If $C_{label} < 1$, the system reduces the human effort for labeling. The lower labeling cost is preferable. In fact, we can consider C_{label} as the percentage of strokes in dataset which still need a manual operation. For instance, after labeling the visual codebook and mapping in Fig. 3, the labeling cost is $C_{label} = \frac{15+12-13}{15} = 0.933$.

In the next section, our proposed method will be tested on two different datasets, single-line mathematical expressions and a flowchart dataset.

VI. Experiment

In this section, two handwritten datasets will be first presented, and then we evaluate the proposed method on such two datasets.

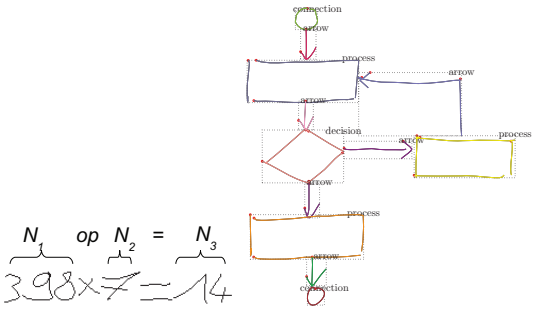
A. Handwritten Corpus

The first simple database is a synthetic handwriting database named "Calculate" [1] of realistic handwritten expressions synthesized from isolated symbols. The expressions in "Calculate" are produced according to the grammar N_1 *op* $N_2 = N_3$ where N_1, N_2 and N_3 are numbers composed of 1, 2 or 3 real isolated handwritten digits. The distribution of the number of digits for $N_{i=\{1,2,3\}}$ is 70% of 1 digit, 20% of 2 digits and 10% of 3 digits randomly. Furthermore, *op* represents one of the operators $\{+, -, \times, \div\}$. Fig. 7a shows an example picked from "Calculate" with N_1, N_2, N_3 and *op* containing 3 digits, 1 digit, 2 digits and "$\times$" respectively. Fifteen classes exist in total.

The second handwriting database is a realistic handwritten flowchart database named "FC" database [2]. We use only the six different graphical symbols that represent the basic operations (data, terminator, process, decision, connection, arrows) without any handwritten text, as displayed in Figure 7b. It contains six classes.

Tab. II shows statistical information on the two databases. Each of them is composed of a training part (first line) and a test part (second line).

In the next section, the values of the labeling cost will be studied with respect to the number of prototypes of the clustering stage.



(a) (b)

Figure 7: Two different handwritten graphical languages: (a) a synthetic expression from "Calculate" composed of real isolated symbols , (b) an example of flowchart in FC database.

Table II: Symbol number and class number on two databases

	Symbol Number	Class Number	Writer Number
Calculate	5472	15	180
	3035	15	100
FC	3641	6	31
	2494	6	15

B. Evaluation

In this section, we evaluate the different codebook size (prototype number) during the hierarchical clustering on the two datasets, and with different segmentation methods. As an illustration, we also display a subset of the visual codebook.

Evaluation of Codebook Size:

Several different metrics can be used to control the hierarchical clustering. Six metrics are proposed in [10]. First, we use the *Average* metric to calculate the codebook. The comparison between the metrics will be discussed later. Fig. 8 shows the labeling costs on two datasets with two segmentations: the ground-truth segmentation and the connected-stroke segmentation. Using the ground-truth segmentation, the labeling cost is very low on both datasets respectively: 8.8% with 250 prototypes on "Calculate" dataset training part and 4.3% with 100 prototypes on "FC" dataset training part. It shows that in the ideal case we can reduce most of the human workload.

Using the connected-stroke segmentation, the labeling cost on the training part of "FC" dataset reports a high value, 94% with 250 prototypes. It means that most of graphical symbols on "FC" dataset are not connected-stroke component. On the training part of "Calculate" dataset, the labeling cost is much lower, 47.4% with 250 prototypes, since the most of graphical symbols, digits, are connected-stroke component. As a conclusion, the segmentation quality

is vital for the labeling cost.

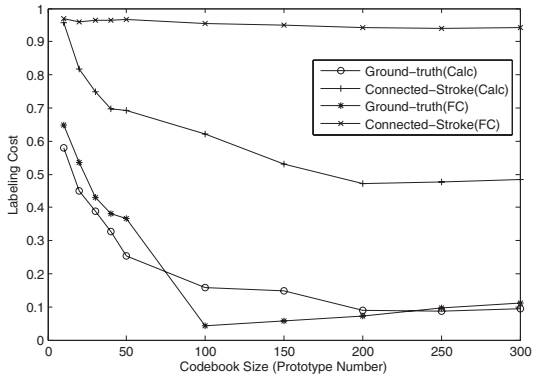


Figure 8: Labeling cost with different codebook sizes on the training parts of two datasets with the ground-truth segmentation and the connected-stroke segmentation

Evaluation on Hierarchical Clustering Metrics:

Six hierarchical clustering metrics are evaluated on two datasets respectively using their best codebook size: (1) *Single*, (2) *Average*, (3) *Complete*, (4) *Centroid*, (5) *Median*, and (6) *Ward* (minimum variance) [10]. Fig. 9 shows the labeling cost for the six metrics using the ground-truth segmentation. Clearly, the *Average* metric reports the lowest labeling cost on the training parts of both datasets.

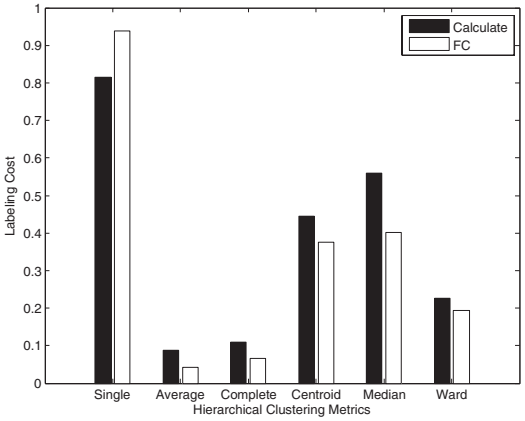


Figure 9: Evaluating the hierarchical clustering metrics on the training parts

Evaluation on Merging Top-N Frequent Bigrams:

On the "Calculate" dataset, the mathematical expressions are arranged from left to right. Using the ground-truth segmentation, we can calculate the bigram distribution. The top-n (t_n) frequent bigrams are merged as new multi-symbol segments to test multi-symbol mapping in the codebook.

Fig. 10 shows the labeling cost on the training part of "Calculate" dataset during the merging of the top-n (t_n) frequent bigrams from 0 to 50 with a step of 10. In Fig. 10, two mapping methods are used. The first is the proposed multi-symbol mapping of this paper. The second is a single-symbol mapping; each cluster can be associated with only one label. The zero in x-axis means that the ground-truth segmentation is used. It shows that the multi-symbol mapping obviously works better.

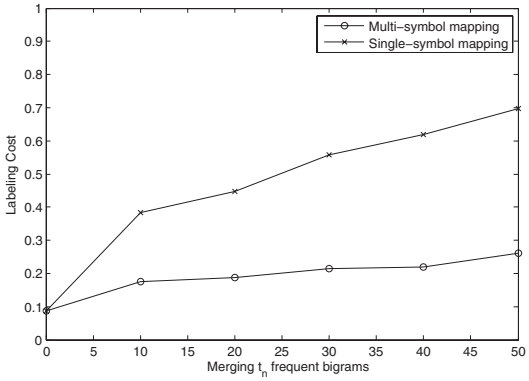


Figure 10: Labeling cost on merging the top-n (t_n) frequent bigrams on the training part of "Calculate" dataset

Evaluation on Test Parts:

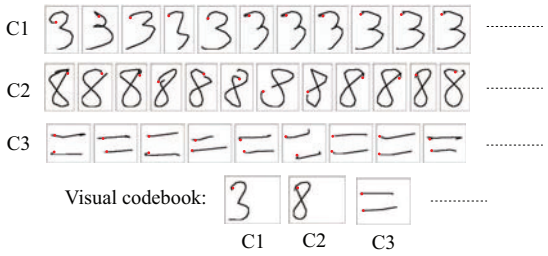
In the previous experiments, we use the training parts, actually used as validation sets, the two datasets to choose the best parameter setting: 250 prototypes on "Calculate" dataset and 100 prototypes on "FC" dataset with the *Average* metric hierarchical clustering. Using these parameters and the connected-stroke segmentation, we obtain fair labeling costs of 50.4% and 97.2% on the test parts of the two datasets respectively. But using the ground-truth segmentation, labeling costs of 13.1% and 13.5% are achieved respectively. These values show that the method is quite effective and that a lot of the annotation task can be saved.

Visual Codebook:

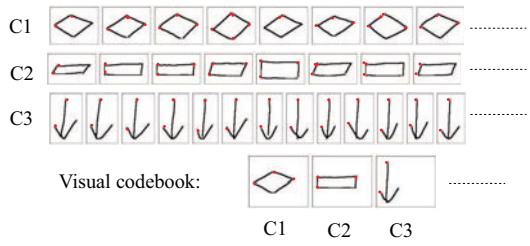
An illustration of the results of the clustering based on the ground-truth segmentation is displayed in Fig. 11. In these selected examples, we can see that the segment shapes are well grouped in the clusters. In each segment, the red point represents the starting point of a stroke. We can see that several digits "8" with different writing orientations and different pen-down position are actually grouped in the same cluster as displayed in Fig. 11a.

VII. Conclusion

In this paper, we proposed a framework for reducing the annotation workload using a codebook mapping for online graphical languages. Starting with the ready-made segmentation, the segments are grouped into a codebook using hierarchical clustering. The visualized codebook is generated for the user labeling. To evaluate the system performance, we define the labeling cost as how much labeling work has to be done by the user. On the test part of



(a) Clusters and pattern representatives from the "Calculate" dataset



(b) Clusters and pattern representatives from the "FC" dataset

Figure 11: Clusters and pattern representatives

two datasets, "Calculate" dataset and "FC" dataset, the low labeling costs of 13.1% and 13.5% are reported respectively using the ground-truth segmentation. Much of work has been reduced thanks to a good segmentation.

However, generating a good quality of segmentation is difficult by an unsupervised method. We cannot use any supervised classifiers to recognize and segment the symbols since our objective consists in labeling the symbols in an unknown language. Our previous work [7], [8] of symbol knowledge extraction based on the minimum description length principle is a possible option for generating the unsupervised segmentation. In future work, we will combine this unsupervised segmentation method to reduce furthermore the symbol labeling cost in this case.

ACKNOWLEDGMENT

This work is partly supported by the French Région Pays de la Loire in the context of the DEPART project www.projet-depart.org.

REFERENCES

[1] Ahmad Montaser Awal. *Reconnaissance de structures bidimensionnelles : application aux expressions mathématiques manuscrites en-ligne*. PhD thesis, Ecole polytechnique de l'université de Nantes, France, 2010.

[2] Ahmad Montaser Awal, Guihuan Feng, Harold Mouchere, and Christian Viard-Gaudin. First experiments on a new online handwritten flowchart database. In *Document Recognition and Retrieval XVIII*, 2011.

[3] M.-P. Dubuisson and A.K. Jain. A modified Hausdorff distance for object matching. In *Computer Vision Image Processing*, 1994.

[4] D. P. Huttenlocher, G. A. Klanderman, and W. A. Rucklidge. Comparing Images Using the Hausdorff Distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 1993.

[5] T. Kohonen. *Self-Organization and Associative Memory*. Berlin, 3rd edition, 1989.

[6] G. N. Lance and W. T. Williams. A General Theory of Classificatory Sorting Strategies: 1. Hierarchical Systems. *The Computer Journal*, 1967.

[7] Jinpeng Li, Harold Mouchère, and Christian Viard-Gaudin. Unsupervised handwritten graphical symbol learning-using minimum description length principle on relational graph. In *KDIR2011*, 2011.

[8] Jinpeng Li, Harold Mouchère, and Christian Viard-Gaudin. Quantifying spatial relations to discover handwritten graphical symbols. In *Document Recognition and Retrieval XIX*, 2012.

[9] Thomas Martinetz and Klaus Schulten. A "neural gas" network learns topologies. *Artificial Neural Networks*, 1991.

[10] Clark F. Olson. Parallel algorithms for hierarchical clustering. *Parallel Computing*, 1995.

[11] R. Plamondon and S.N. Srihari. Online and off-line handwriting recognition: a comprehensive survey. *TPAMI*, 2000.

[12] Guo Xian Tan, Christian Viard-Gaudin, and Alex C. Kot. Automatic writer identification framework for online handwritten documents using character prototypes. *Pattern Recogn.*, 2009.

[13] S. Vajda, A. Junaidi, and G. A. Fink. A semi-supervised ensemble learning approach for character labeling with minimal human effort. In *ICDAR*, 2011.