Sketch Recognition with Natural Correction and Editing

Jie Wu[1,*] **, Changhu Wang**[2]**, Liqing Zhang**[1]**, Yong Rui**[2]
[1]Brain-Like Computing Lab, Shanghai Jiao Tong University, P. R. China
[2]Microsoft Research, Beijing, P. R. China

Abstract

In this paper, we target at the problem of sketch recognition. We systematically study how to incorporate users' correction and editing into isolated and full sketch recognition. This is a natural and necessary interaction in real systems such as Visio where very similar shapes exist. First, a novel algorithm is proposed to mine the prior shape knowledge for three editing modes. Second, to differentiate visually similar shapes, a novel symbol recognition algorithm is introduced by leveraging the learnt shape knowledge. Then, a novel editing detection algorithm is proposed to facilitate symbol recognition. Furthermore, both of the symbol recognizer and the editing detector are systematically incorporated into the full sketch recognition. Finally, based on the proposed algorithms, a real-time sketch recognition system is built to recognize hand-drawn flowcharts and diagrams with flexible interactions. Extensive experiments show the effectiveness of the proposed algorithms.

Introduction

Sketching is a natural way for human to create flowcharts and diagrams in the early design process. It provides an informal environment, where we can use a pen to explore and refine our rough ideas through natural corrections and editing from time to time. Nowadays, as the increasing popularity of touch-screen devices, it is highly desired if we have a practical system to recognize sketched symbols and then convert to formal shapes, which will make flowchart/diagram creation more natural and easier.

Sketch recognition has been studied for more than twenty years. Researchers tried to recognize diverse hand-drawn shapes, such as symbols, flowcharts, diagrams (Peterson et al. 2010), or general objects (Sun et al. 2012b). From the view of sketch complexity, it can be classified to two categories, i.e., isolated hand-drawn symbol recognition, and full hand-drawn sketch recognition. The former one has the assumption that there is only one shape in the sketch, while the latter one is not based on this assumption and thus needs to segment and recognize the strokes at the same time.

Typical methods to recognize isolated hand-drawn symbols include gesture-based, appearance-based, and geometric-based methods. Gesture-based approaches (Rubine 1991) consider the input as a time-evolving trajectory and mainly focus on stroke gesture recognition. Appearance-based approaches (Ouyang and Davis 2009b; Kara and Stahovich 2004) rely on the visual appearance of a sketched shape and are of high robustness to drawing variations. Geometric-based approaches (Paulson and Hammond 2008) attempt to describe objects as geometric primitives and convert it to a constraint satisfaction problem.

Full sketch recognition (Ouyang and Davis 2009a; Bresler, Prua, and Hlavác 2013; Shilman and Viola 2004; Hammond and Davis 2005; Hammond and Paulson 2011; Stevens, Blagojevic, and Plimmer 2013) is more challenging. It tries to segment a sketch into individual objects, and then recognize each of them by leveraging isolated symbol recognizers. A mainstream class of approaches is to first generate candidate stroke groups, and then leverage certain learning framework to find optimal groups.

In spite of continuous research efforts, we still cannot see much technology transfer of sketch recognition in real products, even in the current era of touch screens. For example,

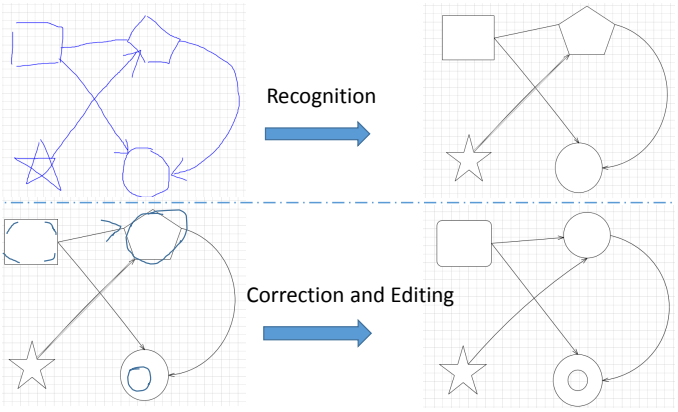

Figure 1: Example results of the SmartVisio system. Users can draw various shapes and connectors in this system, which can be automatically segmented, recognized, and replaced by formal shapes. Users are also able to add strokes to correct or edit any part of the diagram, which will be refined in real time. (Best view in color for all figures.)

[*]This work was performed at Microsoft Research Asia.

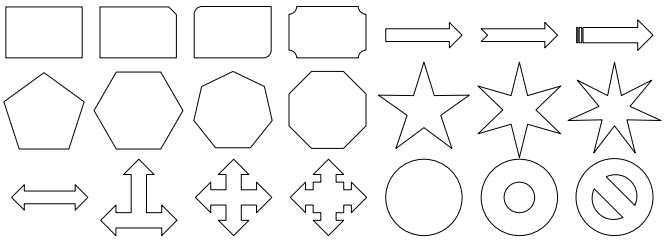

Figure 2: Visually similar shapes in Visio. Top row: similar shapes with local difference. Middle row: globally similar shapes. Bottom row: shapes with inclusion relation.

the leading software for flowchart/diagram creation, Visio, still relies on the traditional point-click-drag style of interaction. This gap mainly comes from the following two aspects, i.e., the challenges from shape data in real systems, and the interaction needs from common users.

- **The challenges from shape data.** In most public data sets for flowchart/diagram recognition, the number of shapes are relatively small and shapes usually have obvious appearance difference, e.g., about 10-25 shapes that differ in a global way in data sets in (Delaye and Anquetil 2012). However, in Visio, there are about 70 shapes only in stencil 'basic shapes' and 'arrow shapes', and thousands of shapes in total, in which some shapes are quite similar, such as the rectangles with different corner types in stencil 'basic shapes', as shown in Fig. 2. It is difficult to distinguish these sketched shapes using existing symbol recognizers, considering the drawing variability of sketches.
- **The interaction needs from common users.** It is observed that users often make corrections or editing by simply adding strokes to an existing symbol (or a full diagram) without erasing the original shapes (Hammond and Paulson 2011; Ouyang 2012), as shown in Fig. 3. In a real system such as Visio where very similar shapes exists, it will be more natural for a user to change a shape (e.g. rectangle) to a related shape (e.g. 'single snip corner rectangle'), because of a correction or mind changing. However, most existing approaches do not explicitly handle this kind of interaction in the recognition process.

In this paper, we systematically study how to handle very similar shapes, and how to incorporate users' natural correction and editing on existing sketches/shapes in isolated and full sketch recognition. First, to discover and leverage the shape relationship in a data set, three editing modes are defined according to visual similarity between shapes, i.e., *local correction* for similar shapes with local difference, *replacement* for globally similar shapes, and *enhancement* for shapes with inclusion relation, as shown in Fig. 2. Then, an algorithm is proposed to discover the prior shape knowledge for these modes. Second, by leveraging the learnt shape knowledge, a novel symbol recognition algorithm is proposed, which has potential to differentiate shapes with abovementioned shape relationships. Besides the symbol recognizer, we also propose an algorithm to detect a user's correction and editing during drawing, and then recognize the symbol in real time. This will speed up the recognition process, and thus make the full sketch recognition more efficient. Third, both of the symbol recognizer and the editing detection algorithm are systematically incorporated into a general framework for full diagram recognition. Finally, based on the proposed algorithms and framework, we build a real-time sketch recognition system based on Visio, called SmartVisio, to recognize hand-drawn flowchart/diagram with flexible interactions, as shown in Fig. 1. To the best of our knowledge, this is the first work to systematically model, detect, and recognize users' correction and editing in sketch recognition. Extensive experiments show the effectiveness of the proposed algorithms on sketch recognition.

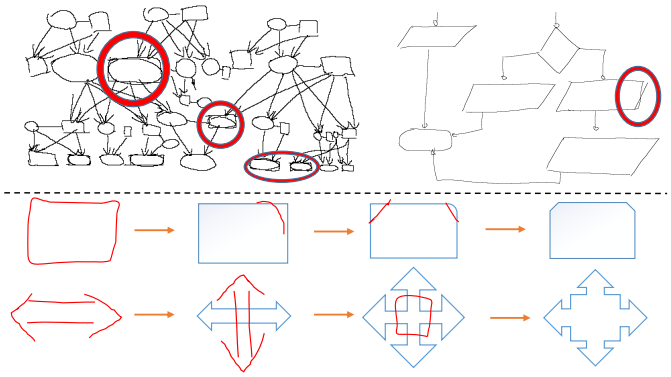

Figure 3: Examples of online correction and editing. Top: correction and editing on sketched symbols in real-world hand-drawn diagrams. Bottom: correction and editing on formal shapes.

Editing Modes and Shape Knowledge Graph

In this section, we first introduce three editing modes in sketch recognition, and corresponding shape relationships. Then, a method is presented to discover these relationships.

Editing Modes and Shape Relationships

It is natural and necessary for users to add additional strokes on existing sketches/shapes to modify them when drawing a flowchart or a diagram (Hammond and Paulson 2011; Ouyang 2012), especially when basic shapes are visually related. For example, as shown in Fig. 4, it is quite convenient for users to directly add a short stroke at the right corner of a 'rectangle' to convert it to 'single snip corner rectangle', or draw a 'circle' to overwrite an 'octagon', or add some strokes to change a 'rectangle' to a 'cube'.

Thus, in this work, we take into account three editing modes users probably adopt during their drawing, i.e., *local correction*, *replacement*, and *enhancement*, each of which represents a type of shape relationship[1].

[1] In this work, we only consider these three editing modes which correspond to different shape relationships, and leave other potential interactions such as deleting to future work.

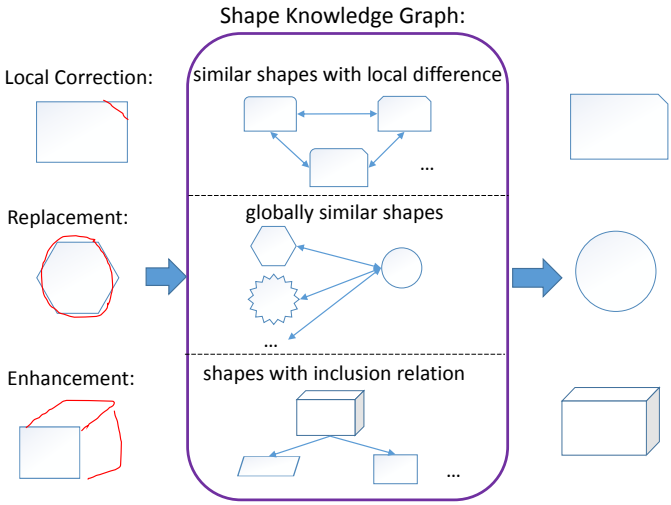

Figure 4: Three editing modes and corresponding shape knowledge graph segments.

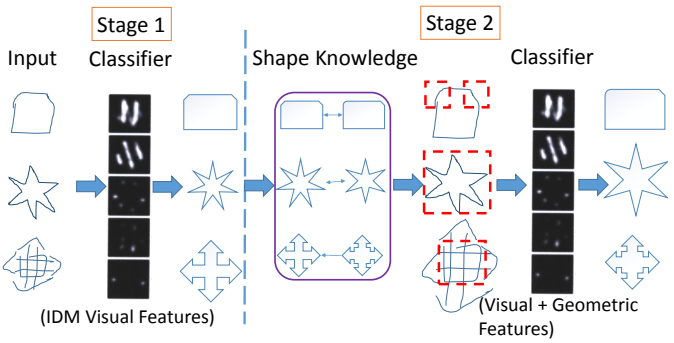

Figure 5: Illustration of the proposed symbol recognition algorithm. In Stage 1, we use IDM recognizer to get the initial results for the input sketch. In Stage 2, first, based on the results of Stage 1, shape knowledge is utilized to detect locally different areas (labeled by red rectangles of dashes). Second, we use a fine-grained recognizer to get the final results, by extracting features only on locally different areas.

- **Local Correction**: adding strokes to correct some local parts of an existing sketch/shape (top row in Fig. 4). This mode corresponds to similar shapes with local difference.
- **Replacement**: overlapping an entire sketched shape on an existing similar sketch/shape (middle row in Fig. 4). This mode corresponds to two globally similar shapes.
- **Enhancement**: adding strokes to an existing sketch/shape to get a new shape (bottom row in Fig. 4). This mode corresponds to two shapes with inclusion relation.

Since these three editing modes correspond to different shape relationships with unique characteristics, we need to design an algorithm to discover these relationships in a shape set and build a shape knowledge graph, which will be introduced next.

The discovered shape knowledge graph plays an important role in the proposed sketch recognition and editing detection algorithms. By leveraging the shape knowledge, on the one hand, we can focus on the locally different parts to differentiate very similar shapes during sketch recognition; on the other hand, we can reduce the search space to the shapes that have relations with the original shape if a correction or editing is detected.

Shape Knowledge Graph Construction

Here we present a method to mine the shape relationships for the following editing modes: local correction, replacement, and enhancement.

First, for each shape pair S_1 and S_2 in the formal shape collection, we find the correspondences between densely sampled points on the two shapes by context shape matching (Belongie, Malik, and Puzicha 2002), a widely used point set matching algorithm. It can handle partial match, robust to noisy or missing points in shape matching. We skip the details of this algorithm due to space limitation.

The outputs of shape matching for S_1 and S_2 include: 1) average shape matching distance $D(S_1, S_2)$, 2) the missing point number between S_1 and S_2, i.e., $D_m(S_1 \rightarrow S_2)$ and $D_m(S_2 \rightarrow S_1)$, and 3) the point correspondences and distances between S_1 and S_2.

Based on the shape matching results, we construct the shape knowledge graph with each shape as a node and each relationship as a directed edge. Each edge represents one of three modes, i.e., local correction, replacement, and enhancement. We judge whether there is a directed edge from one node S_1 to another node S_2 with certain mode by the following algorithm:

- **Local Correction.** An edge with mode *local correction* is added if 1) average distance $D(S_1, S_2) < T_{d1}$, and 2) the missing point numbers $D_m(S_1 \rightarrow S_2) < T_{m1}$ and $D_m(S_2 \rightarrow S_1) < T_{m1}$.
- **Replacement.** An edge with mode *replacement* is added if $D(S_1, S_2) < T_{d2}$.
- **Enhancement.** An edge with mode *enhancement* is added if $D(S_1, S_2) < T_{d3}$ and $D_m(S_1 \rightarrow S_2) > T_{m3}$.

The parameters $\{T_{di}\}$ and $\{T_{mi}\}$ are set empirically. If an edge has multiple modes, only the mode with smallest threshold T_{d*} is reserved. Note that although this algorithm can automatically construct the shape knowledge graph, in real systems, it also supports manually adding edges or removing edges in the graph.

Isolated Sketch Recognition

In this section, we introduce how to leverage the shape knowledge graph in isolated sketch recognition. First, a novel symbol recognizer is proposed, followed by an editing detection algorithm to handle online correction and editing.

Hierarchical Symbol Recognition Algorithm

To differentiate visually similar shapes, we introduce a two-stage algorithm for symbol recognition by leveraging the shape knowledge graph. As shown in Fig. 5, in the first stage,

we use an isolated symbol recognizer to get the initial shape label. As mentioned, in a real system where different shape relationships exist, it is not easy to differentiate visually similar shapes, especially for imprecise hand-drawn strokes. To alleviate this problem, in the second stage, we perform a fine-grained comparison on the locally different parts (areas) of visually similar shapes in the knowledge graph.

Any well-performed symbol recognizer can be used as the initial recognizer in the first stage. In this work, we adopted the Image Deformable Model (IDM) recognizer (Ouyang and Davis 2009b), which used 720-dimension visual features, and achieved state-of-the-art performance in benchmark data sets. Using this recognizer, we can get the initial shape label C for the hand-drawn sketch S. In the second stage, we first get all the shape nodes $\{C_k\}$ if there is an edge from C to C_k. We discard the nodes whose labels do not exist in top (e.g. 10) results of the initial recognizer. Then for each C_k, we use a fine-grained recognizer to get a refined label $C_k^* = C$ or C_k with a score $Score(C_k^*)$.

In this work, we performed two-class 1NN on locally different areas between C and C_k as the fine-grained recognizer to get C_k^* and $Score(C_k^*)$ for sketch S. The distance between C and C_k is defined as the average L2 distance between the features (at each locally different area) of the query sketch and its nearest neighbor, denoted by $avgDist(C, C_k)$. Then the score is given by $Score(C, C_k) = exp^{-avgDist(C,C_k)}$.

Different shape relationships (editing modes) will result in different feature extraction areas (locally different areas) and features for C and C_k:

- **Local Correction.** An algorithm is conducted to discover locally different regions for two similar shapes with *local correction* mode. The basic idea is to cluster nearby points with large matching differences obtained during shape knowledge graph construction. The features for classification are the IDM visual features. The details will be omitted due to space limitation.
- **Replacement.** Different from *local correction* mode, we extract features on the whole symbol instead of different areas. In this mode, we use only one geometric feature for classification: the number of segments after segmenting the symbol into lines and arcs, in order to focus more on the geometric property to differentiate similar symbols.
- **Enhancement.** In this mode, we focus on the parts with missing point correspondences. A simple clustering method is used to get significant parts with trivial parts removed. The features for classification are IDM visual features.

After fine-grained recognition for each shape pair C and C_k, we can obtain the final shape label by $C^* = \text{argmax}_{C_k^*} Score(C_k^*)$.

Correction/Editing Detection and Recognition

The shape knowledge graph is not only useful for differentiating similar shapes in symbol recognition, but also very natural to help handle users' online correction and editing on existing sketches or formal shapes.

As shown in Fig. 3, a user can draw additional strokes on an existing sketched symbol or formal shape, meaning to correct or update the original shape to a new shape. Let us denote the existing sketch or shape as S_{old}, its label C_{old}, and the newly drawn strokes S_{new}. We present an algorithm to automatically classify it to one of the four labels: three editing modes and "none", and predict a new label C.

The basic idea is to calculate a matching score $f(mode)$ for every editing mode. If the highest score is lower than a threshold, we classify it to "none", otherwise we select the mode with the highest score.

First, based on the isolated sketch recognizer introduced in the last subsection, we can obtain both the predicted shape label and corresponding score for S_{old}, denoted by C_{old} and P_{old} respectively. Similarly, we can also get $\{C_{new}, P_{new}\}$ and $\{C_{both}, P_{both}\}$.

For each editing mode, the matching score $f(mode)$ and corresponding shape label $C(mode)$ are determined in a different way:

- **Local Correction.** In this mode, users want to use newly drawn stokes S_{new} to replace the old sketch/shape S_{old} at the same region, with other parts unchanged, resulting in a new shape denoted by $S_{correct}$. Then, we use the isolated sketch recognizer to get $P_{correct}$ and $C_{correct}$. Thus, $C(correct) = C_{correct}$, and the final score is defined as
$$f(correct) = P_{correct} P_{old} Edge_{local}(C_{old} \rightarrow C_{correct}), \quad (1)$$
where $Edge_{local}(C_x \rightarrow C_y)$ is set to 1 if there is an edge from C_x to C_y with editing mode *local correction* on the shape knowledge graph, and 0 otherwise.
- **Replacement.** In this mode, only the newly drawn strokes are useful to users. Thus, we have $C(replace) = C_{new}$, and $f(replace)$ is defined as
$$\begin{aligned} f(replace) =& P_{new} P_{old} Edge_{replace}(C_{old} \rightarrow C_{new}) \\ & (1 - CM(S_{new}, S_{old})/T_{CM}), \end{aligned} \quad (2)$$
where $CM(*, *)$ is the two-way chamfer distance(Borgefors 1988) between two shapes, to punish the score if new strokes and old strokes are not similar enough. T_{CM} is a constant.
- **Enhancement.** In this mode, users care about both the old and new strokes. Thus, we have $C(enhance) = C_{both}$, and $f(enhance)$ is given by
$$f(enhance) = P_{both} P_{old} Edge_{enhance}(C_{old} \rightarrow C_{both}). \quad (3)$$
- **None.** This mode means there is no correction or editing, and thus the old and new strokes will be used together for recognition. Thus, we have $C(none) = C_{both}$, and $f(none)$ is a constant value learnt from the training data.

Finally, the editing detection and recognition algorithm will output the predicted editing mode and shape label by
$$\begin{aligned} \{mode, C_{mode}\} &= EditDetector(S_{new}, S_{old}) \\ &= \underset{mode}{\text{argmax}}\, f(mode). \end{aligned} \quad (4)$$

Full Sketch Recognition

In this section, we incorporate the proposed isolated sketch recognition and editing mode detection algorithms into the full sketch recognition framework. As shown in Fig. 1, in our SmartVisio system, a user can flexibly add strokes to change existing (sketched or formal) diagrams and flowcharts.

We will first introduce a general framework for full sketch recognition, followed by the brief algorithm to deal with online correction and editing.

Full Sketch Recognition Framework

Full sketch recognition needs to simultaneously segment and recognize shapes, and thus is more challenging. A mainstream framework is to first generate a large number of candidate groups of temporally and/or spatially neighboring strokes, and then select and recognize meaningful ones by learning and optimization technologies.

In the second step, different formulations and optimization methods were proposed, e.g., formulating the problem to segment-labeling problem and solving it by belief propagation inference (Ouyang and Davis 2009a), formulating it to max-sum problem and solving it by ILP, a powerful optimization algorithm (Bresler, Prua, and Hlavác 2013), or converting the group selecting problem to an optimization problem solved by an A-star search (Shilman and Viola 2004).

Thus, we study how to incorporate users' correction/editing to this framework. In the implementation, we conduct candidate group selection and optimization using (Ouyang and Davis 2009a). The unit is 'stroke' instead of 'segment', because the data set we evaluate is labeled in stroke level. For simplicity, we also ignore the Joint Context Potential in (Ouyang and Davis 2009a). The proposed isolated symbol recognizer can be used to label each candidate group for better distinctiveness. For more details, please refer to (Ouyang and Davis 2009a).

Handling Correction and Editing

As mentioned before, the users can add strokes to edit the hand-drawn sketch before conducting recognition, or interactively edit the formal diagram/flowchart after recognition when strokes have been replaced by formal shapes. Thus, we briefly introduce the ideas for both scenarios.

Editing on Hand-Drawn Sketch The basic idea is to compare nearby candidate stroke groups, and then merge and recognize them using the proposed editing detection and recognition algorithm. Specifically, we conduct editing detection in Eqn. 4 for every pair of candidate groups, if their distance is less than a threshold, e.g., half of the maximum diagonal of the bounding box of them. Then, a new group is obtained if an editing is detected, and all other groups overlapping with the new group will be removed.

There are two advantages to use this algorithm. First, it enables users to flexibly make corrections or editing, and can detect and recognize it, avoiding the recognition error brought by additional editing strokes. Second, the number of invalid candidate groups can be reduced a lot, which not only can speed up the optimization step, but also make the recognition more robust.

Editing on Formal Diagram/Flowchart In this case, all the symbols on the full sketch have been recognized and replaced by formal shapes. Denote the newly added strokes as St_{new}. We only need to update the formal shapes near St_{new}, denoted as St_{old}. Then, two candidate stroke group collection can be generated for these two types of data, denoted as G_{new} and G_{old}. Note that in G_{old}, the number of candidate groups is larger than that of formal shapes and thus more possibilities are provided, since we enable users to correct, merge, or split existing formal shapes. Then, we conduct editing detection and recognition in Eqn. 4 for each pair of $(g_{old} \in G_{old}, g_{new} \in G_{new})$, followed by the group selection (Ouyang and Davis 2009a) on the small candidate group set.

Experiments

We evaluated the proposed algorithms on three tasks: 1) isolated symbol recognition, 2) editing detection on formal shapes, and 3) recognizing full diagrams with correction and editing.

Isolated Symbol Recognition. To evaluate the algorithms on differentiating visually similar shapes in real systems, we collected a new dataset of 68 commonly used shapes in Visio, i.e., all shapes in the 'arrow' and 'basic' stencils, which are two most widely used stencils in Visio. The dataset contains 1520 samples with 25 examples for each shape. It consists of very similar shapes. As shown in Fig. 2, the shapes in the star point family, rectangle family, arrow family, or polygon family are quite similar.

Since the proposed hierarchical symbol recognizer was built based on IDM (Ouyang and Davis 2009b), the state-of-the-art symbol recognizer, we compared it with IDM (Ouyang and Davis 2009b) and another variant IDM(no rotation) which removes the rotation normalization in preprocessing from IDM. The leave-one-out cross validation was performed. As shown in Table 1, after adding the step of fine-grained recognition, our method correctly recognized 95.92%, outperforming the other methods. It is mainly because that our method can better differentiate the similar shape families, such as the Point Star Family and the Rectangle Family, as shown in Table 2.

We also tested the proposed method in a public benchmark dataset HHReco (Hse and Newton 2004). User independent cross validation is performed. As shown in Table 1, our method performed slightly better than IDM, the best external benchmark, but only with small improvement[2]. This is because HHReco dataset only contains 13 shape classes, which are not very similar. However, if we only look at 'Pentagon' and 'Hexagon' classes, two classes with visually very similar shapes, the accuracies of our method and IDM are 95.97% and 92.19%. The improvement is more significant.

[2]Although it was reported 98.20% using IDM, in our implementation, we only achieved 97.67%. That might result from implementation difference. Since it is the first stage of our algorithm, we think the performance of our algorithm will be improved if we have a better implementation for IDM.

Method	Visio	HHReco
IDM	94.67%	98.20% (97.67%)
IDM(no rotation)	94.00%	95.20%
Ours	**95.92%**	**98.43%**

Table 1: Results of isolated symbol recognition.

Method	Star Point Family	Rectangle Family	All
IDM	86.50%	76.47%	94.67%
Ours	**98.78%**	**80.88%**	**95.92%**

Table 2: Comparison of star point and rectangle families.

Editing Detection on Formal Shapes. In this task, we test the performance of editing detection on formal shapes. We performed this task using the interface of our SmartVisio system, where users can conveniently get various formal shapes. The formal shapes in this test are restricted to the 68 shapes in 'arrow' and 'basic' stencils. Each subject was asked to add additional strokes to make correction or editing on one formal shape in the interface. Each formal shape can be obtained either by dragging the shape from the stencils, or by drawing the symbol in the interface (our system can return the formal shape using our symbol recognition algorithm). Then, the subjects were asked to label the editing mode and shape label $\{mode, C_{mode}\}$, where editing mode is from: Local Correction, Enhancement, Replacement, and None. Our algorithm predicts $\{mode, C_{mode}\}$, and the accuracy of C_{mode} is evaluated. Finally, we collected 140 correction/editing events with labeled editing modes and shape labels from three subjects.

Besides the final editing detection algorithm ('Complete' in Table 3), a baseline algorithm was also evaluated. The baseline algorithm ('Baseline') simply combines the old and new strokes and gets the shape label from the proposed isolated symbol recognition algorithm. Table 3 summarizes the accuracies of the detection results of the two algorithms, from which we can see the effectiveness of shape knowledge and the proposed editing detection algorithm.

Recognizing Full Diagrams with Editing and Correction. In this task, we added correction/editing into hand-drawn full diagrams in the public benchmark dataset (Lemaitre et al. 2013), and generated more challenging synthetic diagrams with natural correction and editing. We used the same setting in (Lemaitre et al. 2013), i.e., 248 flowcharts for training and 171 for testing. We excluded text in the flowchart, since it is out of the scope of this work. We randomly chose some symbols in the flowchart and added corrections on them. Because there are only six shape classes in the flowcharts, we restricted our correction/editing to switch inside the 6 classes. The following editing types were adopted: 1) replace 'Rectangle'/'Parallelogram'/'Diamond' with 'Circle', 2) change 'Rectangle' to 'Parallelogram' (or 'Parallelogram' to 'Rectangle') by adding strokes.

We compared our method ('EnableEditing') with two baseline methods. The first one ('FullFramework') is the full sketch recognition framework without 'handling correction and editing'. The second one ('GreedyBackward') is the sketch segmentation framework for general objects (Sun et al. 2012a), in which the sketch entropy is replaced with our isolated symbol recognizer to adapt it to this domain.

Method	Correctly detected types
SymbolRecognizer	60.71%
Complete	**80.71%**

Table 3: Results of editing detection on formal shapes.

Method	Correctly labeled stroke
GreedyBackward	58.91%
FullFramework	65.62%
EnableEditing	**74.12%**

Table 4: Results of full diagram recognition with editing.

Table 4 shows the comparison results. The performances of two baseline methods are not that good, because this kind of correction is very hard to detect, resulting in the performance drop. The proposed approach achieved the best performance, showing the effectiveness of our framework to handle users' correction or editing.

Time Cost. In the SmartVisio system, the isolated symbol recognizer cost 15 milliseconds on average to recognize a sketch or an editing within 68 shape classes, on a PC with an Intel Core i7-2600 CPU. For the full sketch recognizer, we used the efficient belief propagation inference described in (Ouyang and Davis 2009a). It cost about 0.5 second for a flowchart with 20 symbols.

Conclusions

In this paper, we systematically studied how to model, detect, and recognize users' natural correction and editing in sketch recognition, which is the first work to the best of our knowledge. An algorithm was presented to construct the shape knowledge graph that encodes three typical editing modes, based on which, a novel isolated sketch recognizer was proposed for similar symbol differentiation. We also proposed an editing detection and recognition algorithm, and then incorporated it into the full sketch recognition framework, enabling users to flexibly correct or edit existing sketch or recognized formal shapes. Based on the presented technology, we built a real-time sketch recognition system based on Visio, i.e., SmartVisio, to enable users to create diagrams/flowcharts by flexibly drawing. Extensive experiments have shown the effectiveness of the proposed algorithms.

Acknowledgements

The work of Jie Wu and Liqing Zhang was partially supported by the national natural science foundation of China (Grant Nos 61272251,91120305).

References

Belongie, S.; Malik, J.; and Puzicha, J. 2002. Shape matching and object recognition using shape contexts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 24(4):509–522.

Borgefors, G. 1988. Hierarchical chamfer matching: A parametric edge matching algorithm. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 10(6):849–865.

Bresler, M.; Prua, D.; and Hlavác, V. 2013. Modeling flowchart structure recognition as a max-sum problem. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, 1215–1219. IEEE.

Delaye, A., and Anquetil, E. 2012. Hbf49 feature set: A first unified baseline for online symbol recognition. *Pattern Recognition*.

Hammond, T., and Davis, R. 2005. Ladder, a sketching language for user interface developers. *Computers & Graphics* 29(4):518–532.

Hammond, T., and Paulson, B. 2011. Recognizing sketched multistroke primitives. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 1(1):4.

Hse, H., and Newton, A. R. 2004. Sketched symbol recognition using zernike moments. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 1, 367–370. IEEE.

Kara, L. B., and Stahovich, T. F. 2004. An image-based trainable symbol recognizer for sketch-based interfaces. In *AAAI Fall Symposium*, 99–105.

Lemaitre, A.; Mouchère, H.; Camillerapp, J.; and Coüasnon, B. 2013. Interest of syntactic knowledge for on-line flowchart recognition. In *Graphics Recognition. New Trends and Challenges*. Springer. 89–98.

Ouyang, T., and Davis, R. 2009a. Learning from neighboring strokes: combining appearance and context for multi-domain sketch recognition. In *Advances in Neural Information Processing Systems*, 1401–1409.

Ouyang, T. Y., and Davis, R. 2009b. A visual approach to sketched symbol recognition. In *Proceedings of the 2009 International Joint Conference on Artificial Intelligence (IJCAI)*, 1463–1468.

Ouyang, T. Y. 2012. *Understanding freehand diagrams: combining appearance and context for multi-domain sketch recognition*. Ph.D. Dissertation, Massachusetts Institute of Technology.

Paulson, B., and Hammond, T. 2008. Paleosketch: accurate primitive sketch recognition and beautification. In *Proceedings of the 13th international conference on Intelligent user interfaces*, 1–10. ACM.

Peterson, E. J.; Stahovich, T. F.; Doi, E.; and Alvarado, C. 2010. Grouping strokes into shapes in hand-drawn diagrams. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI- 10)*, 974–979.

Rubine, D. 1991. Specifying gestures by example. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '91, 329–337. New York, NY, USA: ACM.

Shilman, M., and Viola, P. 2004. Spatial recognition and grouping of text and graphics. In *Proceedings of the First Eurographics conference on Sketch-Based Interfaces and Modeling*, 91–95. Eurographics Association.

Stevens, P. C.; Blagojevic, R.; and Plimmer, B. 2013. Supervised machine learning for grouping sketch diagram strokes. In *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling*, 43–50. ACM.

Sun, Z.; Wang, C.; Zhang, L.; and Zhang, L. 2012a. Free hand-drawn sketch segmentation. In *Computer Vision–ECCV 2012*. Springer. 626–639.

Sun, Z.; Wang, C.; Zhang, L.; and Zhang, L. 2012b. Query-adaptive shape topic mining for hand-drawn sketch recognition. In *Proceedings of the 20th ACM international conference on Multimedia*, 519–528. ACM.