Sketch Recognition via String Kernel

Shizhong Liao, Menghua Duan
School of Computer Science and Technology
Tianjin University, Tianjin 300072, P. R. China

***Abstract*—Sketch recognition is one of the essential step of sketch understanding. Challenge in sketch recognition is the variation and imprecision present in sketch. Free drawing styles of sketching make it difficult to build a robust sketch recognition system. This paper proposes a novel recognition approach that can recognize primitive shapes, as well as combinations of these primitives. The approach is independent of stroke order, number, as well as invariant to size and aspect ratio of sketch. Feature string is used to represent primitives. We defined a similarity measure on these feature strings that counts common substrings in two input strings, which is referred to as the string kernel in the field of kernel methods. Support vector machine(SVM) is then trained with labeled examples to handle the task of classification. The experiment on hand drawn digital circuit diagrams shows that our system can recognize sketching efficiently and robustly.**

Keywords–Sketch Recognition; Sketch; Support Vector Machines; String Kernel

I. Introduction

Sketch recognition is one of the essential step of sketch understanding. It is the interpretation of hand drawn diagrams, which seeks to understand the users' intent while allowing them to draw unconstrained diagrams. Challenge in sketched symbol recognition is the variation and imprecision present in sketches, the free drawing style of sketching makes it difficult to build a robust sketch recognition system. Sketch recognition has been widely studied and applied to a variety of domains [1]–[5]. And there have been many sketch recognition approaches.

Exiting sketch recognition systems can be roughly divided into three categories: stroke-based [6]–[8], rule-based [2], [3] and feature-based [9], [10] .

Stroke-based recognition is based around the premise that each stroke has a specific role in representing a sketch. Sketch is recognized by first breaking up the strokes into geometric primitives such as lines, arcs, and ellipses. Then symbols are composed of primitive geometric parts and the constraints between them. Recognition is then posed as a subgraph matching problem between predefined shape descriptions and the geometric primitives from the strokes. However, graph matching in the classification stage can be very computationally expensive.

Rule-based recognition usually employ heuristics that work well for a small set of shapes. However, this type of recognizer is usually hard to extend or adapt, and are not very robust to noise.

A number of approaches have stepped back from the properties of individual strokes to classify shapes based on a set of features such as length, ratio, speed that calculated on the whole shape. i.e. a specific number of numerical features are extracted for sketch object. These features form so-called feature vectors and then static machine learning algorithms such as support vector machine can be used to handle the task of classification. Mapping sketch symbols to feature vectors is attractive, as it transfers a complex data type into a simpler one, on which a huge family of distances, similarity measures and efficient data mining algorithms are available. However, they do not represent the individual details of the shape. And loses internal structure information about symbols.

Different from vectors, strings are structured data, providing information about the structure of an object. Thanks to the suffix tree, distances, similarity measures of strings can be performed very efficiently [11]. For this reason, we decided to explore a novel approach for sketch recognition : Instead of a vector calculation, we examine if a structured description of sketch symbols, namely feature strings, can be used for efficient and accurate sketch recognition.

With the approach proposed in this paper, we built an sketch recognition system, which focused on hand drawn digital circuit diagrams. The system has a number of interesting capability:

- It can deal solidly with the ambiguity and variability that are so pervasive in sketches.
- It can provide an efficient and convenient user interface for sketch editing.
- The low level recognizer can be easily integrated as an input module into other more complex systems.

The rest of this paper is organized as follows: In section II an overview of our recognition system is given. We will define the feature string and the implementation of our sketch recognizers in section III. In section IV we evaluate the performance of our system by some experiments. Finally, we conclude this paper in section V.

II. Overview

In this section we shortly review the basic ideas behind string kernel. Kernel methods are a family of related machine learning and data mining algorithms. The building block of kernel methods is kernel function, i.e. a function returning the inner product between the mapped data points in a higher dimensional space, which can be thought as a measure of similarity on input data. Based on this kernel function, several algorithms can be formulated for classification, clustering and regression [11], [12]. In this paper support vector machines is used to deal with the task of classification.

978-1-4577-2133-5/10/$26.00 ©2012 IEEE

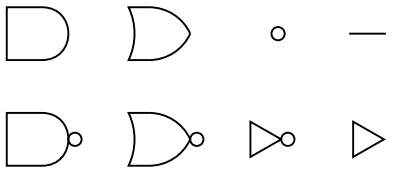

Fig. 1. Target symbol set.

Kernel methods have been applied successful in lots of domains, but there are many cases that input data can't readily be represent by feature vectors. Watkins [13] defined a principled way of designing kernels on structured objects. Based on this framework, kernels on structured objects such as strings and trees, have been defined over recent years. Now we describe a kernel between two strings. The idea is to compare them by means of substrings they contain: the more substrings in common, the more similar they are [11], [14].

The string kernel is defined as follows: Let Σ be a finite alphabet. str is a string from Σ. $|str|$ denotes the length of string str. We denote by Σ^n the set of all finite strings of length n. We now define feature spaces $F_n = \mathbb{R}^{\Sigma^n}$. The feature mapping ϕ for a string str is given by defining the u coordinate $\phi_u(str)$ for each $u \in \Sigma^n$. We define

$$\phi_u(str) = num_u(str), \tag{1}$$

where $u \in \Sigma^n$, $num_u(str)$, is the number of occurrences of substring u in string str.

$$\begin{aligned} K(str_1, str_2) &= \omega_u * \sum_{u \in \Sigma^n} \langle \phi_u(str_1) \cdot \phi_u(str_2) \rangle \\ &= \omega_u * \sum_{u \in \Sigma^n} num_s(str_1) * num_s(str_2), \end{aligned} \tag{2}$$

where $\omega_u = 1$ if $\min(|str_1|, |str_2|) \geq n$ else $\omega_u = 0$, n is the length of common substring.

To scale all kernel values to the same range, we normalized all kernel values by

$$\hat{K}(str_1, str_2) = \frac{K(str_1, str_2)}{\sqrt{K(str_1, str_1) * K(str_2, str_2)}}. \tag{3}$$

A direct computation of these features require $O(|\Sigma|^n)$ time and space, since this is the number of features involved [14]. In our sketch recognition system the implementation of string kernel is based on package SASK, which is an implementation of string kernel that based on suffix trees. Surprisingly the kernels can be computed in linear time and linear space using suffix trees [15]. We modified SASK to support combine string kernel. In this similarity measure methods, as a positive definite kernel function, the string kernel allows us to employ kernel methods such as support vector machine, for classification.

The architecture of our system can be seen in Fig (2). The overall process can be divided into two stages: primitive symbol recognition and combine symbol recognition. Symbol

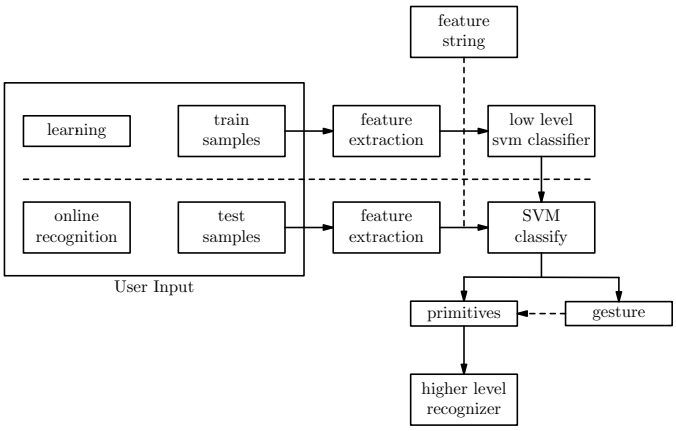

Fig. 2. Architecture of the system.

is defined as a cluster of one or more strokes that represent an entity in the domain, they are the unit of classification in our system.

Our system is focused on a set of digital circuit diagrams collected in a free sketching interface. The target symbol set is shown in Fig (1). In the digital circuit symbol set, many of the gates are similar in appearance and there are great variety in possible drawing styles. As such it provides a good test of the robustness of our representations and classifier. Of course, other shapes can be added and learned by the system, if desired.

In [1], [16], [17] Alvarado has done some work on recognition of hand drawn digital diagrams. The experiment result in [17] declared that training on synthesis data does not appear to improve the accuracy, even when recognizing synthesis data. So we would like to collect isolated symbols instead of synthesis sketches. Five primitives such as triangle, AND gate as well as three combine symbols such as NOR gate are collected. To date, we had collected two sets of labeled data, the first set consisted of 375 isolated primitives. The second set contain 150 combine symbols.

III. Recognition System

A. Low Level Recognition

We define primitive symbol as symbols that cannot be represented as a combination of simpler symbols. The low level recognizer is trained with a set of commonly used components of digital circuit diagrams. After the group and preprocessing of input strokes, the low level recognizer will be performed immediately. The recognition results are then sent to a higher level recognizer.

It's difficult to make a trade off between freedom and accuracy in sketch recognition. For example, users may vary the number as well as the drawing order of strokes used to draw a symbol. This variation presents a challenge for a system can't know how many strokes each symbol will contain, nor the order in which these strokes will be drawn [1].

Our system does make two assumptions about the hand drawn digital circuit diagrams. First, people usually pause

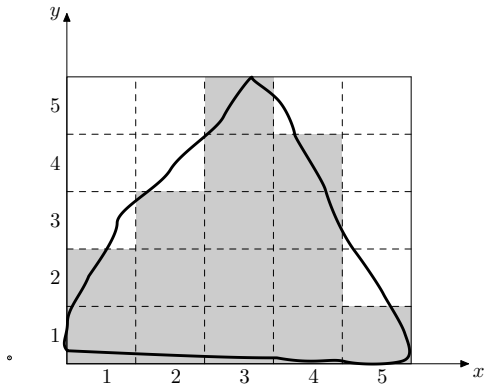

Fig. 3. Feature string construction of a hand drawn triangle,n=5.

some seconds when one symbol are finished. Second, all the parts of one symbol are drawn before drawing the next.

A symbol is deemed finished when the elapse time between two strokes is longer than a threshold (This threshold is adjustable and it is set to 1 second). Strokes input before the pause will be grouped into a cluster to represent an individual symbol. The grouped strokes will be sent to a preprocessor, where strokes in a cluster will be joins if the minimum distance between them is less than a threshold. For example, when the endpoint of a stroke is very close to the endpoint of another stroke, the two strokes are linked.

One of the steps in applying string kernel to a problem is to decide on a set of features that are sufficiently expressive the given problem at hand. After the group and preprocess of strokes, these strokes that grouped into one cluster will construct a primitive symbol. In the following we will transform those preprocessed strokes to strings, i.e. using a feature string to describe the sketch symbols.

The low level recognizer identifies the user's input by overlaying a $n * n$ grid with squares on its bounding box, which determining the distribution of the region that envelop with those strokes. A grid is considered filled if and only if the center point of this grid is in the region that surrounded by this cluster strokes. The 9*9 grid seems optimal for the symbol recognition in our experiment result. Fig (3) is an example and for simplicity n is set to 5.

Each grid can be addressed by its coordinates (x, y), where $1 \leq x, y \leq n$ and $x, y \in \mathbb{N}$. A grid g can then be related to two functions:

$$g_x : \mathbb{N}^2 \rightarrow \{0, x\}$$

$$g_x(x,y) = \begin{cases} x & \texttt{if } (x,y) \texttt{ is a filled grid} \\ 0 & \texttt{else} \end{cases}$$

and

$$g_y : \mathbb{N}^2 \rightarrow \{0, y\}$$

$$g_y(x,y) = \begin{cases} y & \texttt{if } (x,y) \texttt{ is a filled grid} \\ 0 & \texttt{else.} \end{cases}$$

The feature string generation is a essential step of our approach. We aim for strings that describe the distribution of filled grids along each of the two axes x and y. For the computation of the feature strings we repeat through the grid, once for each dimension x, y and create one feature string for each dimension. The algorithm for the generation of feature string is as follows:

Algorithm 1: FEATURE STRING GENERATION

Require: An integer $n \geq 0$, an empty string str_x.
Ensure: Feature string str_x.

```
for each x ∈ [1 : n] do
    for each y ∈ [1 : n] do
        append_character (str_x, 'g_x(x, y)');
    end for
end for
```

Analogously we can construct feature string str_y for the sketch symbol. Now we can use two feature strings str_x and str_y length of n^2 to represent a sketch symbol. Note that in this scheme feature strings of straight line will be constructed only by character '0' and can be distinguished from other classes. Science the $n*n$ grid is inscribed in the input symbol's bounding box, so the algorithm works for all sizes and aspect ratios. Obviously this scheme is adaptable to vary number and order of strokes when a symbol was drawn. Further more it can deal with the over trace and pen-drag in sketch. Analogously to our approach, Gross [18] identifies the input glyph's shape by overlaying a 3*3 grid with squares numbered 1-9 on its bounding box, determining the sequence of grid squares that the pen moved through to draw the symbol. This sequence is used as a lookup key in a hash table of previously trained symbols. However their scheme can not adapt to the vary of drawing order. Another important different is that our system recognize a symbol by classification not search.

Given two sketch symbols s_1 and s_2 we can compute the string kernel pairwise for the str_x and str_y of s_1 and s_2. We get two kernel values, which is a measure of similarity. A simple way of fusing these two similarity measures into one combine kernel is addition.

$$\begin{aligned} K_{sum}(str_1, str_2) = \; & \omega_{K_x} * K_x\big(str_x(s_1), str_x(s_2)\big) + \\ & \omega_{K_y} * K_y\big(str_y(s_1), str_y(s_2)\big), \end{aligned} \tag{4}$$

where ω_{K_x} and ω_{K_y} are weight coefficient of each kernel and both set to 0.5. K_x and K_y are string kernel that defined in (2). All kernel values are normalized beforehand by (3), to scale them to the same range.

As a positive definite kernel function, the combine string kernel allows us to employ support vector machines for classification. Using the kernel we had trained a classifier to learn the differences between different symbol classes from labeled examples. This provides a way to distinguish one class from another, for example AND gate from OR gate. However, we aim to assign each input symbol a single label from the set of possible shape classes. This is accomplished by using

the common one-vs-one strategy for combining a set of binary classifiers. We used the package LibSVM in our system.

In our system the low level recognizer could treat with 5 different primitive symbols. And some gesture commands are supported, which allow users to edit the sketched symbols naturally and efficiently. Gestures are trained and recognized in the same way with other symbols. There exist ambiguities between some gestures and ordinary primitives, e.g. a circle can be interpreted as component of gate symbols or the select command. To solve this conflict, a heuristic rule is used: if the circle is drawn on an blank space, then it is most likely an ordinary stroke, otherwise the select command is assumed.

While this paper focused on hand drawn digital circuit diagram, we made efforts to design our low level recognition to be as general as possible. With domain knowledge our system is capable of recognize specific domain symbols. An earlier version of our low level sketch recognizer is tested in plan geometry and the result is encouraging. However this does not mean that our system will generalize to other domains without modification.

B. Higher Level Recognition

The program uses a bottom-up parsing approach to assemble primitive symbols into combine symbol. After the low-level recognizer has tried to identify the primitives drawn by the users, the higher-level recognition will take place. Combine symbols are described as composition of primitives of certain types that arranged in certain spatial relation. Together with primitives, a set of spatial constraint are key for higher level recognition. With an eye to the domain knowledge of digital circuit, spatial relations such as left, right, above, below and intersect are supported in our system. Fig (4) shows the spatial relations can be exited in a NOT gate. The inspect of relations between primitives can be done through bounding boxes and endpoints.

In the recognition of combine symbols, representation of combine symbols is the first and essential problem. Considering the digital circuit diagram, the number of primitives in a combine symbol is small. A feature string of length 8 is used to represent a combine symbol, every bit of this string stand for certain information.

The first five bits of the string represent the numbers of each primitive in a combine symbol. They are organized as follow: (line, circle, triangle, AND gate, OR gate). For example NOT gate is constructed by a circle and triangle, so in our scheme the first five bits will be ”01100”. In this model, a combine symbol is represented as an unordered collection of primitives, disregarding the order they appeared.

The sixth bit stands for the distance information between primitives. Just as shown in Fig (1), primitives in combine symbol intersect at a point. So we think it’s necessary that the minimum distance between the primitives is less than a threshold (adjustable). If this condition is meet, the sixth bit will be ’1’, else set to ’0’.

The next two bits will be used to represent spatial relations between primitives in a combine symbol. Note that all the three combine symbols is construct by a circle and another primitive. For ease of description we take NOT gate as an example, Among the two bits, the value of the former can be {’L’, ’R’, ’0’} which means the triangle is Left/Right of the circle or none of them presence. While the value of the laster can be {’A’, ’B’, ’0’}, which means the triangle is Above/Below of the circle or none of them presence. Fig (4) shows the possible spatial relations that exist in a NOT gate, in our scheme they can be represent as ”0B”, ”0A”, ”L0”, ”R0” respective. Analogously, relations between primitives in NAND gates and NOR gates can be described.

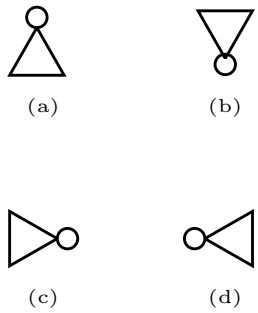

Fig. 4. In NOT gate the triangle can be below/above, left/right of the circle.

Now we can use a feature string of length 8, which contains the numbers of primitive as well as the relations of primitives, to represent a combine symbol. For example a NOT gate shows in Fig (4.a) can be described as ”011001L0”, which means NOT gate is constructed by a triangle and circle. They are intersected in a point and the triangle is on the left of the circle. Analogous to low level recognizer, the higher level recognizer is trained with labeled examples and common substring of length 8 is computed using (2). Primitives are organized according to the order they appear. With the assumptions described at the begin of this section, that all the parts of one symbol are drawn before drawing the next. Higher level recognizer will try to combine a primitive respectively with it’s next one as well as previous one to test if they compose a combine symbol in train set.

IV. Experimental Results

In designing the experiments for evaluating our recognition system, we want to examine whether our sketch recognition approach via string kernel works well. In each of the two data sets, ratio between train data and test data is set to 2:1. Note that for all the experiment described in this section we used the LibSvm package and the SVM parameter C was set to 5.

A. Effectiveness of Varying Substring length

Our low level recognizer identify user’s input by overlaying a $n*n$ grid with squares on its bounding box and then compute the common substring of length n. Note that for each new value of n, the length of feature string changes too. For each new value of n, we get a different kernel. In this test, we will observe the effectiveness of varying substring length. Five-fold cross-validation is performed on primitives data set to get a

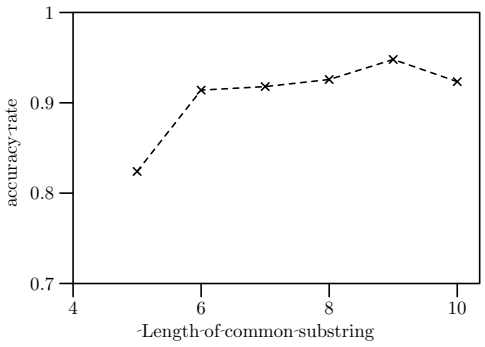

Fig. 5. The effect of the variability of substring length on performance. Five-fold cross-validation is performed to get a average of accuracy rate.

average of accuracy rate. We tested the value of n range from 5 to 10. And when the value of n is set to 9, the low level recognizer achieved the best 94.8% accuracy rate. The results is shown in Fig (5).

B. Performance of Each Recognizer

In this experiments we trained a low level recognizer and a higher level recognizer with respective training data. And the value of n was set to 9. As for the higher level recognizer common substring of length 8 was computed. Experiment results on test data set is shown in Table I.

Alvarado [17], described a free sketch recognizer using gesture recognition technology. In the domain of digital circuit diagrams, they can achieve overall 94% accuracy. Our recognition result is comparable to theirs. In addition their recognizer cannot address noise like vary of numbers in an unknown symbol, or over trace, while our approach is very robust to those noise.

V. Conclusions

In this paper we proposed a novel approach to sketch recognition. The evaluation in last section indicates that sketch recognition via string kernel is feasible. Our system proved to be comparable to the current state-of-art recognition systems. In addition our system can deal with noise like vary of strokes number in an unknown symbol and over-trace. It can recognize sketching efficiently and reliably.

The approach proposed has been applied in hand drawn digital circuit diagrams and plain geometry. Other domains will be tested in future. In our system, time clues is used to deal with the problem of grouping strokes. Further research is needed to find ways of addressing the problem of stroke grouping automatically.

Acknowledgment

The work was supported in part by the National Natural Science Foundation of China under grant No. 61170019 and the Natural Science Foundation of Tianjin under grant No. 11JCYBJC00700.

TABLE I
Results for each recognizer. n was set to 9, and SVM parameter $C = 5$.

		Precision	Recall
Low Level Recognizer	Line	91.304%	95.454%
	Triangle	84.615%	100.0 %
	Circle	86.956%	86.956%
	AND gate	95.0 %	86.363%
	OR gate	90.909%	80.0 %
Higher Level Recognizer	NAND gate	78.42%	88.235%
	NOT gate	87.219%	82.352%
	NOR gate	83.137%	67.849%

References

[1] C. Alvarado and M. Lazzareschi, "Properties of real-world digital logic diagrams," *plt*, pp. 1–6, 2007.
[2] T. Hammond and R. Davis, "Tahuti: A geometrical sketch recognition system for uml class diagrams," in *ACM SIGGRAPH 2006 Courses*. ACM, 2006, pp. 25–es.
[3] Y. Liu, Q. Lin, and G. Dai, "Pigp: A pen-based intelligent dynamic lecture system for geometry teaching," in *Proceedings of the 2nd international conference on Technologies for e-learning and digital entertainment*. Springer-Verlag, 2007, pp. 381–390.
[4] T. Chen, M. Cheng, P. Tan, A. Shamir, and S. Hu, "Sketch2photo: internet image montage," in *ACM Transactions on Graphics (TOG)*, vol. 28, no. 5. ACM, 2009, p. 124.
[5] S. Liao, X. Wang, and J. Lu, "An incremental bayesian approach to sketch recognition," in *Proceedings of the 4th international conference on machine learning and cybernetics, Guanzhou*, 2002.
[6] D. Rubine, "Specifying gestures by example," in *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*. ACM, 1991, pp. 329–337.
[7] X. Xu, W. Liu, X. Jin, and Z. Sun, "Sketch-based user interface for creative tasks," in *Proceedings of the 5th Asia Pacific conference on computer human interaction, Beijing*. Citeseer, 2002, pp. 560–570.
[8] T. Hammond and R. Davis, "Ladder: A language to describe drawing, display, and editing in sketch recognition," in *International Joint Conference on Artificial Intelligence*, vol. 18. Citeseer, 2003, pp. 461–467.
[9] H. Hse and A. Richard Newton, "Recognition and beautification of multi-stroke symbols in digital ink," *Computers & Graphics*, vol. 29, no. 4, pp. 533–546, 2005.
[10] M. Fonseca, C. Pimentel, and J. Jorge, "Cali: an online scribble recognizer for calligraphic interfaces," in *AAAI spring symposium on sketch understanding*, 2002, pp. 51–58.
[11] J. Aßfalg, K. M. Borgwardt, and H.-P. Kriegel, "3dstring: a feature string kernel for 3d object classification on voxelized data," in *CIKM*, 2006, pp. 198–207.
[12] C. Burges, "A tutorial on support vector machines for pattern recognition," *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121–167, 1998.
[13] C. Watkins, "Dynamic alignment kernels," *Advances in Neural Information Processing Systems*, pp. 39–50, 1999.
[14] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins, "Text classification using string kernels," *The Journal of Machine Learning Research*, vol. 2, pp. 419–444, 2002.
[15] C. Teo and S. Vishwanathan, "Fast and space efficient string kernels using suffix arrays," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 929–936.
[16] E. J. Peterson, T. F. Stahovich, E. Doi, and C. Alvarado, "Grouping strokes into shapes in hand-drawn diagrams," in *AAAI*, 2010.
[17] M. Field, S. Gordon, E. J. Peterson, R. Robinson, T. F. Stahovich, and C. Alvarado, "The effect of task on classification accuracy: Using gesture recognition techniques in free-sketch recognition," *Computers & Graphics*, vol. 34, no. 5, pp. 499–512, 2010.
[18] M. Gross and E. Do, "Ambiguous intentions: a paper-like interface for creative design," in *Proceedings of the 9th annual ACM symposium on User interface software and technology*. ACM, 1996, pp. 183–192.