Graph-Based Locality-Sensitive Circuit Sketch Recognizer

ABSTRACT
The understanding of circuit diagram is very important for the study of electrical engineering. Existing circuit diagram simulation tools are mostly based on GUI interface and rely on users to click or drag icons with mouse, which requires them to be familiar with the software and distracts a great deal of their attention from the circuit diagram itself. This paper constructs a prototype of pen-based circuit diagram system. It enables users to draw circuit diagrams directly on the digital screen without learning how to use it. At the same time, a graph-based sketch recognition algorithm is proposed to recognize diagram components efficiently and it is not sensitive to different drawing habits. Our approach has achieved 93.04% recognition accuracy on an experiment of 158 samples collected from 17 users and 4.53 out of 5 on average for user satisfaction. Theoretical derivation and experiments have demonstrated that our algorithm and prototype system are efficient as well as stable with high value in practice compared with previous state-of-the-art methods. The same approach can also be applied to other general sketch recognition scenarios. To facilitate future researches and applications, we publish our source code, model, and training data.

KEYWORDS
Pen Interaction, Graph, Circuit Sketch

1 INTRODUCTION
Circuit diagram is the basis and difficulty of electricity in physics and engineering. With modern technologies, circuit diagrams can be simulated on screens with users operating positions and relationships of a series of components. Users especially students can get the visualized results from such simulators and understand electrical rules and laws gradually with the assistance of interactive programs, which also benefits teaching in education. A circuit simulation system should be user-friendly, i.e., users can input to the system conveniently, and understandable enough for them to learn the internal electrical principles. At present, most existing public circuit simulation systems require users to drag components to proper positions with mouse, e.g., [3]. This approach is easy to design yet relatively complex and tedious for users since these operations can distract user’s attention from the circuit diagram itself significantly and fail to provide an immersive drawing and learning environment. Therefore, this paper designs a pen-interaction circuit diagram recognition and analysis prototype system. It allows users to draw circuit diagrams on a tablet and is as convenient as drawing on paper.

There are a number of works focusing on hand-drawn circuit diagram recognition. However, the methods in these works either have strict restrictions on users sketch input (e.g., all the strokes must follow a required order [9]), or are not robust enough to handle a noisy and poorly drawn sketch (e.g.,[11]). Some recent deep learning methods like [17], [25], [21], and [16], which perform well on image object detection and semantic analysis tasks, are not suitable for this scenario since data sets with tons of labeled samples are needed to train a reliable deep neural network model and it is hard to guarantee that the model can deal with people’s different drawing habits.

In this work, we use graph models to represent users’ input sketches. And then local cycles in graphs are utilized to detect and localize circuit components as well as analyze their connection relationships. The pen-interaction circuit recognition and analysis system is developed based on this model and algorithm. Theoretical guarantee and experiments have shown that our algorithm and system are efficient as well as stable.

This paper has mainly three contributions:

1 proposes an efficient representation model and algorithm for circuit recognition task;
2 designs a complete, stable, and satisfying prototype system for circuit diagram teaching and learning;
3 makes up for the deficiency of open-source code and datasets in the areas of sketch recognition and circuit component classification.

The following parts will be organized as follows: section 2 introduces some related works about this paper; section 3 elaborates system architecture and our algorithms; some implementing details and our experimental study are shown in section 4; section 5 concludes this paper and introduces some future works; and there is an appendix section in the end of this paper giving theoretical analysis to our algorithm.

1https://github.com/AnAnonymousProgrammer/LS4D
2 RELATED WORKS

A number of works have been devoted to addressing problems similar to circuit sketch recognition. Nevertheless, each approach has its own limitations.

Firstly, there are some arts operating sketch image directly and using pixel information to analyze connection relationships between components. For instance, De Silva et al. [9] took advantage of the concept of ink density (density of painted pixels) to identify the occurrence of components and connecting points. Similar ideas are also adopted in [11], [23], [19], and [7]. One major concern of these methods is that they require a completed sketch image and is weak to recognize the circuit online.

There are also works dealing with online sketches. For this representation, a key issue is to sort input strokes into their belonged components or specify which component each stroke belongs to. In early stage, this classification was controlled by users manually. For example, Fonseca et al. [14] divided strokes by pause, i.e., if the user pauses for a certain time interval before drawing next stroke, the previous strokes would belong to a component and next strokes would belong to another. Liwicki et al. [20] extracted independent circuit components by switching modes. These methods can certainly work yet not flexible and user-friendly enough, since they highly rely on cooperation of users.

Many studies have proposed automated solutions to this inconvenience. Valois et al. [28] completed the identification and beautification of hand-drawn circuit diagrams by extracting the structure and topological relationship of the images; Gennari L et al. [15] combined the geometric features of the image and domain knowledge to explain circuit sketches; Dreijer et al. [10] proposed a novel normalization process to make it easier to recognize components; Alvarado et al. [8] utilized bayes nets to infer the relationships between each strokes according to stroke temporal information. However, there are more or less user restrictions on hand drawing. For example, users are not allowed to draw other parts of a circuit diagram before finishing the current component, which is not always the case actually. Latter on, Sezgin et al. [26] extended the probability graph model [8] to support interspersed drawings. Nevertheless, it is still influenced by stroke order to a large extend.

Feng et al. [13] used two-dimensional dynamic programming to store the information of each state of each stage of the user’s drawing of the circuit diagram to identify the circuit diagram. This method relaxes stroke order restriction for users by considering temporal and spacial information of input strokes at the same time and solve the existing problems mentioned above successfully. However, in terms of running time, it takes nearly one minute to process a regular circuit diagram.

Different from previous works, this paper proposes an $O(n)$ algorithm, where $n$ is the total number of strokes. It can accurately detect and extract sketch circuit components with a novel graph model. Furthermore, we also build a prototype system that benefits teaching and learning for electrical knowledge. It is shown that our method can be applied to other fields with minor changes.

3 METHODS

This section elaborates our approaches to build the circuit recognition and analysis system. It consists of four modules: UI module, sketch analysis module, circuit state and attribute calculation module, and circuit component classifier. UI module interacts with users directly and takes users’ input strokes. Sketch analysis module firstly applies the model and algorithm mentioned before to detect and localize sketch circuit components. And then utilizes these results to analyze the connection relationship between each components. Circuit state and attribute calculation module takes advantage of the results in sketch analysis module to make a list of equations about unknown attributes and solves it to get the results. And circuit component classifier is a convolutional neural network (CNN) encoder. The dependency relations of these modules are shown in Fig.1.

Interface Design

As shown in Fig.2, the sketch interface of our system is simple. There is no need for users to learn to use it, but just draw on it directly. The system would detect circuit components, and draw a rectangular box with a specific color around it. Different kinds of components are assigned different colors. The type of current drawn component is also shown in the left-top corner. When system detects a circuit component, a small button would appear near this...
component and there would be an input box if the user clicks on the button, which is used to input some attributes for it, e.g., current, voltage, resistance values, etc. The input information would be shown in green. After user finishes sketch of the entire diagram, click on Start Analysis button to start calculating the remaining attributes and states of the diagram and calculated results would be shown near each component in red. The circuit components used in this paper are the same as those in [2] and shown in Table 1.

### Recognition Algorithms

Our model runs in an online scenario. Each stroke is represented as a vertex in the graph. There would be an edge between two vertexes if and only if their corresponding strokes come into contact with each other, i.e., there is a common pixel on one of strokes’ endpoint on the screen. One advantage of using such graph model is that it is an abstract representation and immune to users’ different sketch habits geometrically. As shown in Fig.3, every stroke in Fig.3(a) corresponds to its own unique vertex in Fig.3(b), which is similar to the concept of dual graph. It is worth noting that edge set of our graph model should be a multi-set since there can be more than one contacting point between two strokes, as shown in strokes with number 6 and 7 in Fig.3. In addition, we allow a stroke contacts with itself to form a selfloop in graph. As stroke 4 in Fig.3, the user sketches the border of the ammeter with only one stroke, with adjacent start point and end point forming a selfloop. Therefore, simple graph is not powerful enough to model our problem, so we use a more general graph with multiple edges and selfloops allowed.

With this graph model, we propose our main algorithm: **Locality-Sensitive Special Sketch Symbol Detection (LS⁴D)**. Our algorithm is designed based on following observations: most of circuit component symbols are equipped with a closed border, as shown in Table 1 and [2], which are represented by cycles in graph models. Besides, strokes that form circuit components usually have closed connections in local spacial areas, which means that it is sufficient to only take strokes in the same local area into consideration. According to these two characteristics, we develop an algorithm that is good at detecting symbols with closed borders and name it **Locality-Sensitive Special Sketch Symbol Detection (LS⁴D)**. The main algorithm is shown in Algorithm 1, where...
Algorithm 1 General LS$^4$D

Require: New stroke $s$; Graph model $G < V, E >$;
Previous stroke set $S$; Previous cycle set $C$.

1. Segment $s$ into sub-strokes $s_{n+1}, s_{n+2}, \ldots, s_{n+m}$;
2. for $p = 1$ to $m$
3.   $\text{mark}(s_{n+p}) = -1$;
4.   Update $S$ and $G$ with above modeling methods;
5. if $\exists v \in C, $ satisfies $s_{n+p} \subseteq v$ then
6.   $c = c \cup \{s_{n+p}\}$;
7. else
8.   $S_L = \{s_i|d_{i,n+p} \leq loc\}$;
9.   $S_L = S_L - \{s_i|\text{marked}(s_i) \neq -1\}$;
10. $V_L = \{i|s_i \in S_L\}$;
12. $c = \{s_i|\exists j \in V_L, e_{i,j} \notin \text{cut}[G_L]\}$;
13. if $c = \emptyset$ then
14.   $C = C \cup \{c\}$;
15. end if
16. end if
17. $r \triangleq \text{classifier}(c$’s corresponding image);
18. $\text{mark}(s_j) = r, \forall s_i \in c$;

$s$ is new input stroke, $G$ denotes the graph model mentioned in previous sections, $V$ and $E$ denote vertex and edge set of $G$ respectively, $S$ denotes previous stroke set, $C$ denotes the set of cycles formed previously, $\text{mark}$ means the kind of component a stroke belongs to, and $\text{deg}$ means degree of a vertex.

General LS$^4$D. Firstly, we adopt a stroke segmentation step to process the new stroke in line 1 of Algorithm 1. In this step, we use both degree of the new stroke’s corresponding vertex and cycle borders as indicators to divide the stroke. To be specific, as soon as user paints the first pixel, a new vertex would be allocated for this new stroke. Then we keep tracking user’s nib. If it bumps into a stroke on a closed border, we would do segmentation here and start a new sub-stroke with a new vertex. It is worth noting that when a stroke collides to strokes other than this type, the algorithm would ignore it except that it happens on an endpoint of the new stroke or an existing stroke, to handle some crossing but non-joined wires. This kind of strategy makes crossing but not connected wires possible. In the segmentation process, when degree of the new stroke goes up to 2, the algorithm would also start a new sub-stroke, i.e., the degree of vertex corresponding to a new sub-stroke is at most 2. It makes sure that the algorithm would not miss any required cycle, which is helpful to extract and analyze all the cycles one by one.

Afterwards, we will focus on each sub-stroke, whose $\text{mark}$ is $-1$ initially.

Based on the common feature of our target symbols, i.e., a closed border, a natural idea is to divide the algorithm into two main branches to deal with the two cases respectively: the new stroke falls in a previously detected cycle and it is not in any cycle, corresponding to the condition statement in line 5 of Algorithm 1. We use notation $\notin$ to indicate that a stroke falls in a closed border. Obviously, if a new stroke is in a cycle, it needs to send all strokes in this cycle (including the cycle itself) into the classifier module. Even though some of the strokes have already been classified, we have to send some marked strokes into classifier again to ensure correctness of final classification results, since the content in it has changed. See line 6 of Algorithm 1.

On the other hand, if the new stroke is not in any cycle, the algorithm would find its nearby strokes, whose corresponding vertexes have a relatively short distance to the new vertex in the graph model, as shown in line 8 of Algorithm 1. Here we introduce a hyper parameter $\text{loc}$ to indicate the locality sensitivity of the algorithm. Only vertices with a distance no more than $\text{loc}$ to the new one are taken into account. Therefore, a higher $\text{loc}$ means a weaker locality sensitivity. If there is a need to consider those strokes that are relatively far from the new one, $\text{loc}$ should be higher. At the same time, the algorithm will consider less local information and more global information.

Since this branch satisfies the condition that the new stroke does not in any existing cycle, those marked strokes should not influence the segmentation and analysis of following strokes. Therefore, we do not take marked strokes into consideration and delete them from $S_L$ in line 9 of Algorithm 1. We then extract the induced subgraph $G_L$ for $S_L$’s corresponding vertex set $V_L$ in the current graph model $G$.

After this step, the algorithm would detect the target symbols only in this subgraph $G_L$ instead of the global graph, which reflects the local sensitivity of our algorithm. Subsequently, taking advantage of the closed border, we only need to examine whether there is a cycle in $G_L$. When a cycle is found, the algorithm can separate out the border strokes of target sketch symbols, and thus complete the detection tasks. In line 12 of Algorithm 1, the algorithm would find the cut-edge of $G_L$, which is equal to finding a cycle as shown by the following theorem[1].

Theorem 1. An edge is a cut-edge if and only if it is not contained in any cycle.

Considering that some users may draw the inner-frame content of a symbol at first and the border afterwards, in line 14 of Algorithm 1, we also add strokes which are enclosed in the extracted border, i.e., the corresponding strokes of the cycle.
Without omission, LS users tend to use relatively less strokes to finish drawing a circuit sketch, according to our observation and user study, LS Fine-tuned appendix section. Proof of above theorem and conclusion can be found in the out any error report, i.e., set $loc = 1$. In other words, we only take those strokes that directly adjacent to the new stroke into consideration. Then the time complexity of the key operation of $LS^4D$ for each sub-stroke is $O(1)$ and total time complexity is $O(|S|)$, which outperforms previous state-of-the-art techniques [15][8][26][13]. Detailed analysis of time complexity can be found in the appendix section. Since the degree of a new input sub-stroke’s corresponding vertex is at most 2, the algorithm can only find cycles with length no more than 3. Unfortunately, through our survey, some users still prefer to draw a resistor with 4 strokes, corresponding to the left, up, right, and bottom borders respectively. In this case, the basic $LS^4D$ algorithm would fail to detect this kind of resistor, as shown in Fig.4(a). Therefore, we introduce the concept of rectangular closure and utilize it to fine-tune the basic $LS^4D$ algorithm to solve this problem. To calculate the rectangular closure of a set of strokes, firstly, we need to find the smallest rectangle which can enclose all the strokes in the set, i.e., the boundary coordinates of the rectangle is the maximum and minimum values of the strokes in the set on axes $x$ and $y$ ($x_{max}, x_{min}, y_{max},$ and $y_{min}$). Subsequently, we find all the strokes that are in the rectangle calculated in the last step. The set of these strokes is the rectangular closure of the original stroke set. In other words, we expand the original $S_L$ through a rectangular rule and use expanded $S_L$ instead of the original one for the following steps. Formally, the following two rectangular closure calculation steps are inserted after line 8 of Algorithm 1 to obtain fine-tuned $LS^4D$:

$$R \triangleq \min\{\text{Rect} | \forall s_i \in S_L, s_i \subseteq \text{Rect}\},$$  \hspace{1cm} (1)

$$S_L = \{s_i | s_i \in R\}. \hspace{1cm} (2)$$

As shown in Fig.4(b), the problem can be solved efficiently after we use the rectangular closure.

Analysis Algorithms
Detection for Other Symbols. The $LS^4D$ algorithm can detect those sketch symbols with closed border efficiently. However, in the scene of circuit sketch recognition, symbols of power supply and capacitor are not equipped with this feature. Thus, $LS^4D$ fails to detect sketch of these symbols and we need some other mechanisms to tackle this problem. In this paper, we continue using our graph model to find these symbols.

As shown in Table 1, symbols of power supply and capacitor consists of two unconnected strokes. In addition, strokes belonging to these components usually have only one wire connected. This is unlike those strokes belonging to the wires, since both ends of wires are connects by components in a normal circuit graph. In other words, while detecting these symbols, we can pay attention to vertices...
with degree 1, and verify their distances between each other as well as stroke lengths to decide whether they belong to power supply, capacitor, or just noise.

Connection Relationship Analysis. With fine-tuned LS^4D algorithm and a simple mechanism to locate power supplies and capacitors, we can detect and locate all the circuit components as well as divide all the strokes according to the components they belong to. Now we can take advantage of these previous results to analyze their relationships between each other and convert user’s input strokes into format needed by future calculation on the circuit diagram to implement our sketch analysis module. Firstly, all strokes remaining a −1 mark would be viewed as wires in circuit diagrams. Due to the flexibility of wires in the circuit graph, it is equivalent to directly use a stroke’s corresponding vertex in our dual graph model to represent a node in the actual circuit graph. Therefore, edges between marked strokes and strokes with a −1 mark describe relationships between circuit components and wires perfectly.

Circuit State and Attribute Calculation. This part introduces the circuit state and attribute calculation module of our system. After extracting the connect relationships between all the wires and components, the last step is to calculate the state and attribute of the circuit graph. The basic idea is to construct a set of equations according to electrical laws like Kirchoff’s circuit laws [4] and Ohm’s law [5]. Specifically, we can use nodes converted from strokes with −1 (wire) mark to build equations for electric current with Kirchoff’s current law, use cycles from positive pole of power supply to negative pole to build equations for voltage with Kirchoff’s voltage law, and use detected circuit components to build equations with Ohm’s law. Due to the forms of these equations, the set of equations constructed above can be converted into a linear system of equation. Therefore, we can consider using Gauss elimination method [22] to solve it. As long as conditions are sufficient, the rank of the matrix is no less than the number of unknown quantities and the solver can find the solution. The details of the solving process is not the core aspects of this paper.

Here, we have built a complete circuit sketch recognition and analysis prototype system and implemented all the related algorithms.

4 EXPERIMENTS AND USER STUDY

Low-Level Implementing Details

To begin with, the sketch interface in our system is an electronic handwriting screen with 1920 × 1080 resolution. Similar to a down-sample process, we view the screen as a 640 × 360 checkerboard, where each cell is a 3 × 3 pixel area. And a 640 × 360 map is used to denote each cell is occupied by which stroke. Initially all the entries of this map are empty and an entry would be set as current stroke number if user’s nib touches on a pixel belonging to this cell. This map is convenient to update our graph model and the down-sample process makes sure that this map does not consume two much memory.

On the other hand, in real cases, since sketch is inaccurate, stroke might not connected exactly with the endpoints being connected. In other words, there may be a subtle gap between two semantically connected points. Based on this observation, we relax the criterion of detecting inter-stroke collisions. As long as the distance between two endpoints is less than 5 cells, i.e., 15 pixels, these two endpoints would be viewed as joint. This value is set based on most users’ sketch habits in order to minimize the error rate from this aspect.

In addition, we employ the stroke segmentation approach in [12] besides our original segmentation method to handle this problem, to handle cases that users draw multiple components with one stroke. At last, there are only the following restrictions or concerns to users’ input:

- the hyper-parameter loc is set to 1 as default to reduce computational cost and users need to finish a component with a closed border in at most 4 strokes (a higher loc can relax this restriction and when loc = inf this restriction would not exist);
- a distance less than 5 cells, i.e., 15 pixels would be viewed as being connected. Users need to control the distance to make it equal to a value more than that when they want a pair of disconnected endpoints and less than that when they want a connected pair;
- a detail should be noted that when two wires cross with each other, the system would not view them being connected and users can divide one wire into two or put a dot on the crossing point to achieve connecting, which is consistent with common drawing standards.

Compared with previous arts like [15][8][26][13], the methods and system in this work do not require users to present their strokes as a specific order, or maintain the whole screen clean at any time as shown in following user study.

Circuit Component Classifier

Sample Source: The training and test samples are collected from 11 users, including on-the-job and retired teachers, college students, and middle and high school students, so that the final model is oriented to people with different sketching habits, and classification accuracy can be as high as possible. Altogether, we collect about 200 hand-drawn samples for each type of circuit component with a closed border, and the total number of samples is over 1600. The details of our data
Table 2: Details of our data set and test results

<table>
<thead>
<tr>
<th>Component Type</th>
<th>Total # of Samples</th>
<th>Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light Bulb</td>
<td>174</td>
<td>90.9%</td>
</tr>
<tr>
<td>Resistor</td>
<td>205</td>
<td>98.0%</td>
</tr>
<tr>
<td>Ammeter</td>
<td>184</td>
<td>92.9%</td>
</tr>
<tr>
<td>Voltmeter</td>
<td>181</td>
<td>92.3%</td>
</tr>
<tr>
<td>Diode</td>
<td>228</td>
<td>94.6%</td>
</tr>
<tr>
<td>Transistor</td>
<td>242</td>
<td>91.4%</td>
</tr>
<tr>
<td>Motor</td>
<td>236</td>
<td>92.2%</td>
</tr>
<tr>
<td>Buzzer</td>
<td>235</td>
<td>95.2%</td>
</tr>
</tbody>
</table>

set are shown in Table 2. We randomly divide all samples of each type of component into a training set, a cross-validation set, and a test set according to the 7 : 2 : 1 ratio.

Data Preprocessing: In order to consider a variety of different sketch styles as well as the equivalence of horizontal and vertical drawing of some components, we perform data augmentation on the training samples [24], as follows:

- for diodes, transistors, light bulbs, resistors, and buzzers, carry out the horizontal and vertical symmetry transformations on the images, as well as 90, 180, and 270 degrees rotation transformations;
- for diodes, buzzers, and resistors, carry out 1.2, 1.5, and 1.8 times compression transformations horizontally and vertically;
- for motors, ammeters and voltmeters, carry out the horizontal symmetry transformation.

The final training set consists of all original images and transformed images, with over 6500 samples in total.

Training and Testing Details: We use LeNet-5 [6] as our basic network architectures. Different from the original network, we use dropout with a probability of 0.5 for the first two fully connected layers to avoid overfitting [27] and replace original ReLU activation functions with LeakyReLU [29]. We adopt Adam [18] ($\alpha = 10^{-3}, \beta_1 = 0.9, \beta_2 = 0.99$) as optimizer, with a batch normalization strategy (batch size is 2). After each epoch, the model is tested on the cross-validation set. In the end, our model achieves a 99.9% accuracy rate on the training set, a 99.7% accuracy rate on the cross-validation set, and an overall accuracy rate of 93.7% on the test set after the entire training. The accuracy rates of each type of component are shown in Table 2.

User Study

Recognition Accuracy Test. We convey user study on 17 users to test our algorithms and system. Each user draws 6 circuit diagrams provided by us from official tests for high school students (part of users also drew some other samples designed by themselves). The result is shown in Fig.5. We collect altogether 158 samples, with 11 anomalous analysis. The overall analysis accuracy is 93.04% and all the exceptions fall in the user restrictions mentioned above. Note that users were not informed about these restrictions before the test. Numerically, the average accuracy is higher than previous state-of-the-art approaches in [8], [13] and [26].

For specific instances, it turns out that our algorithm and system can not only face regular circuit examples from physics text books, but also handle messy input with some noises and complex problems, as shown in Fig.6 and Fig.7. Here we hide the input interface to make a clean panel. There are three types of exceptions. The first one is due to default loc value. When loc = 1, theoretically LS4D can only find cycles with length at most 3 if the rectangular closure is not introduced and if users draw a component with strokes more than that number, there could be a small loc exception. Fortunately, this can be solved by using a higher loc value and sacrifice some running efficiency. The second is about the setting of connecting pixel error tolerance mentioned above. We set a 15-pixel threshold to decide whether two strokes are connected. In these exception cases, users represented disconnecting with a distance smaller than 15 pixels and represented connecting with a distance higher than that value. Setting this value more reasonably is a meaningful research topic that is not the main focus in this paper. And the third type is about misclassification from the classifier.

User Satisfaction. We also carried out a satisfaction evaluation based on a 5-point Likert scale. The rating results are shown in Fig.8, most rating results concentrate on level 4.5 ∼ 5.0. Some representative comments are listed below:

- I think the system has high practical value and it covers almost all points of electricity in middle and high school. I am satisfied with the accuracy and efficiency of this recognition and analysis system, and look forward to more helpful functions in teaching such as...
highlighting and blackboard-writing (from a physics teacher);
- The most valuable use of this system for me is to check my homework answers efficiently because it supports sketch and I can draw circuits freely on it as convenient as drawing on a scratch paper. It can also help me calculate solutions in real time and develop my understanding on electrical laws. Thanks to this system, I am no longer afraid of circuit diagram problems. (from a high school student).

Running Time Experiment
In this part, we use different loc values to evaluate the average per-stroke running time of our LS^4D algorithm using a uniform standard circuit shown in Fig.2. We sample the setting of loc from 1 ~ 6 and 10. The corresponding results are shown in Fig.9. Under default loc value 1, we get a latency less than 60µs per stroke on average on a modern workstation with an Intel-9980HK CPU. To a general trend, the running time and loc shows a linear relationship. The results indicates that the key steps in our algorithm only take roughly 100µs even though loc is as high as 10 and users can hardly sense the system delay. As a result, 100% of users are satisfied with real-time performance of our system.

5 CONCLUSION AND FUTURE WORKS
This paper proposes a method to model users’ drawn strokes with vertices in graphs, and proposes an efficient and reliable locality-sensitive algorithm to detect and localize symbols that have closed borders through information of cycles. The algorithm has solid theoretical foundation. The circuit recognition and analysis system based on it produces 93.04% accuracy on a user experiment with 158 samples. In this process, we publish the source code of our algorithm, model, and data set used by sketch circuit component classifier, which facilitates researches in the field of sketch recognition, especially for circuit sketch recognition.

Since general LS^4D algorithm works for all the symbols with a closed border, it also can be applied to other type of sketch, like sketches of flow chart and UML diagram. In the future, we would like to extend the system to a wider range of applications, e.g., to build it as a real educational software that can be deployed in classrooms, and to apply the algorithm into systems of flow chart and UML diagram recognition and analysis.

6 APPENDIX
In this section, we supplement some materials to demonstrate the correctness of our main algorithm, i.e., Graph-based
Proof of Algorithm Correctness

Proof of Theorem 2:

Proof. We use mathematical induction to prove that there is no any cycle with length no more than $2 \times \text{loc} + 1$ at any time step.

Initially, it is obvious that when there is only one stroke or one vertex, the theorem is true. Assume that the conclusion is true for all $n > 1$, where $n$ denotes the size of vertex set of graph $G < V, E >$, i.e., $|V|$. Suppose the algorithm is currently ready to process $n + 1$th sub-stroke $s$ (here we only consider stroke that does not in any existing cycle) and the corresponding vertex is $v$.

Since we only consider those strokes with $\text{mark} = -1$, we can temporarily delete those vertexes whose corresponding strokes do not satisfy this property and get the induced subgraph $G'$. Determined by the operation of $LS^4 D$, we send strokes whose corresponding vertexes are on a cycle to classifier. According to the inductive hypothesis, there is no cycle with length no more than $2 \times \text{loc} + 1$ in $G$. As $G'$ is a sub-graph of $G$, this hypothesis also applies to $G'$. Therefore, all the edges are either cutting edges, or on one or more cycles with length more than $2 \times \text{loc} + 1$ in graph $G'$.

Besides, due to our stroke segmentation method and pre-treatment operation on the stroke, the degree of a new sub-stroke’s corresponding vertex is at most 2, i.e., $\deg(v) \leq 2$. Obviously when $\deg(v) = 1$ or $\deg(v) = 0$, the new vertex is certainly not on any cycle so the inductive hypothesis is still true in these two cases. Therefore, if there are cycles with length no more than $2 \times \text{loc} + 1$ in graph $G' + v$, these cycles must contain vertex $v$ at the same time and $\deg(v) = 2$. The algorithm would consider vertexes whose distance to the new vertex is at most $\text{loc}$ as long as the length of a cycle is no more than $2 \times \text{loc} + 1$, it would be sent to the classifier and get marked. In this way, the new vertex must be marked and there are less vertexes that remain unmarked. Then the induced graph derived by these unmarked graph $G''$ is a subgraph of $G'$.

Since there is no cycle with length no more than $2 \times \text{loc} + 1$ in $G'$, the same thing is also applied to $G''$. Hence, it is correct that there is not any cycle with length no more than $2 \times \text{loc} + 1$ at any time step for $n + 1$th sub-stroke and then the algorithm would detect and mark all the cycles with length no more than $2 \times \text{loc} + 1$, i.e., the recall rate is 100%.

Proof for Precision Rate:

Proof. Decided by the algorithm itself, our algorithm never sends a set of non-cycle strokes into the classifier. In other words, when we send a found cycle of the graph model into classifier, it is also a loop from the perspective of strokes, which can form the closed border of a symbol. Thus, the detection report of our algorithm is reliable.

In summary, $LS^4 D$ will detect all the required cycles accurately without omission. Since the target objects of the algorithm is symbols with closed borders, it can detect all the satisfied symbols correspondingly.

Analysis of Time Complexity

Due to the uncertainty of the number of sub-strokes, we take a single sub-stroke, instead of a whole input stroke as the unit for our time complexity analysis. According to the above analysis, the degree of a new vertex is at most 2. And if we set $\text{loc} = 1$, the algorithm can detect cycles whose length is at most 3, i.e., the new vertex and its two adjacent vertexes. Therefore, the computational cost of the key step, which is finding cycles, is constant level $O(1)$.

However, when we set $\text{loc} > 1$, in the worst case, the algorithm has to consider all the vertexes of marked strokes, since there is always a way of constructing the graph to push all the vertexes in the graph have the distance at most 2 to the new vertex. A typical example is a star graph. In this case, the time complexity is $O(|V| + |E|)$. Since there is no cycle in the previous graph, and the degree of the new vertex is at most 2, we have $O(|V|) \sim O(|E|)$. The time complexity in the worst case is $O(|V|)$, which is no worse than the state-of-the-art method [13].

Fortunately, the worst case mentioned above is an extreme case. In application, users’ input strokes usually have high spatial locality and this bad event happens rarely according to our user study. In conclusion, our algorithm has only constant time complexity level $O(1)$ when we set the highest local-sensitivity. And the time complexity would raise if we set a higher $\text{loc}$ value. It would be $O(|V|)$ in the worst case and it is decided by the local sensitivity, i.e., $\text{loc}$, in normal cases. The results about time complexity in application can be found in Fig.9.

REFERENCES


