Department: IEEE Computer Graphics and Applications
Editor: Torsten Möller, torsten.moeller@univie.ac.at

Segmentation and Recognition of Offline Sketch Scenes Using Dynamic Programming

Recep Sinan Tümen
Koç University

Metin Sezgin
Koç University

***Abstract*—**
Sketch recognition aims to segment and identify objects in a collection of hand-drawn strokes. In general, segmentation is a computationally demanding process since it requires searching through a large number of possible recognition hypotheses. It has been shown that, if the drawing order of the strokes is known, as in the case of online drawing, a class of efficient recognition algorithms become applicable. In this paper, we introduce a method that achieves efficient segmentation and recognition in offline drawings by combining dynamic programming with a novel stroke ordering method. Through rigorous evaluation, we demonstrate that the combined system is efficient as promised, and either beats or matches the state of the art in well-established databases and benchmarks.

SKETCH RECOGNITION is the task of identifying groups of ink representing individual domain objects in hand-drawn sketches. Work on sketch recognition makes a distinction between online and offline sketch recognition. In online recognition, sketching is viewed as a temporal process. It is assumed that the order in which the user puts the strokes on the drawing surface is known. Recognition treats sketches as raster images, or collections of strokes. The drawing order is unknown.

In either case, sketch recognition involves two intertwined processes: segmentation and symbol recognition. Typically the segmentation process enumerates alternative stroke groupings, while symbol recognition assigns labels to each group to form recognition hypotheses. The relative merit of the hypotheses is assessed based on how well they account for the ink on the drawing surface. Combinatorics of the grouping process prohibits considering all possible hypotheses. This is a primary challenge in sketch recognition. Existing algorithms attempt to address this challenge by invoking problem specific heuristics or by making

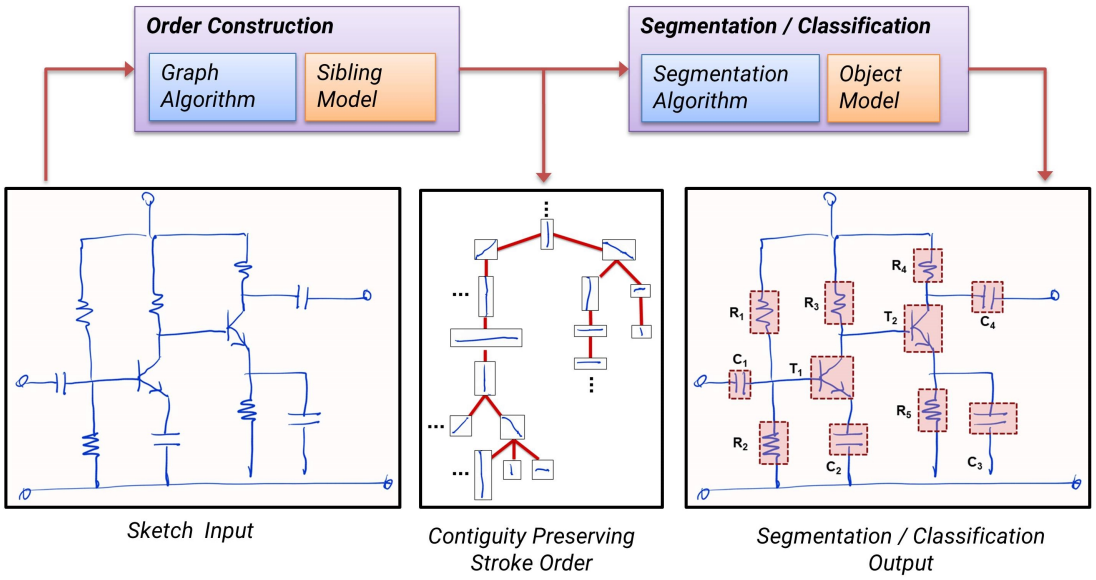


Figure 1: Our offline sketch recognition framework consists of two sequential modules. The first module, order construction, constructs a contiguity-preserving stroke order. The second module, segmentation/recognition, partitions and labels the domain objects using the produced stroke order.

simplifying assumptions about the input.

For example, offline recognition algorithms explore the search space of possible segmentations using heuristics that trade accuracy for speed in various ways (e.g., limit the spatial search radius, assume a maximum number of strokes per object, or perform greedy recognition). Unfortunately, these algorithms usually do not have optimality guarantees in the sense that they may fail to discover a better hypothesis for a sketch scene even if its superiority for could readily be verified with the symbol recognizers at hand.

Online recognition algorithms tackle the combinatorics of recognition using the temporal stroke ordering information available in online drawings. Evidence from the psychology and sketch recognition literature shows that people naturally use certain stroke orderings more often than others (e.g., drawing a stick figure is most likely to start with the head)[1]. Furthermore, people usually finish drawing one object before moving on to the next, hence strokes from individual objects come in temporally contiguous groups[2]. These regularities in online drawings, although not guaranteed to exist at all times, are significant from a recognition perspective because knowing the stroke ordering allows efficient recognition and segmentation based on dynamic programming [3], [4].

Unfortunately, stroke ordering information is entirely missing in offline drawings. Furthermore, sometimes temporal contiguity may be violated in online drawings (i.e., one may start drawing a new object before completing the current one). Hence, existing polynomial time algorithms are not applicable [4]. In this paper, we propose an efficient recognition algorithm that does not depend on having the drawing order. Our approach rests on two pillars. First is an algorithm that constructs stroke orderings from offline sketches such that strokes from distinct objects do not mix in order. In other words, the strokes appear in a contiguity-preserving order. Second is an algorithm for computing optimal segmentation using these stroke orderings.

Constructing contiguity-preserving stroke orderings from offline sketches is not trivial. It has not been studied in the literature. Our method is comprised of: (1) a trainable *sibling model*, and (2) a *graph algorithm* to construct an order. To construct ordering, we first assess the likelihood of pairs of strokes being part of the same object using the sibling model learned from data. This

produces a graph of sibling scores. Then, we construct contiguity-preserving stroke orderings by processing this graph. We present two approaches for training sibling models, and three methods for graph processing, making a total of six methods.

As an integral part of our order construction method, we also present an optimal and efficient algorithm that partitions ordered strokes into domain objects. Since our order construction methods produce total or partial stroke orders, our segmentation algorithm is able to digest both types of stroke orders. Segmentation evaluates if the overlapping sub-sequences of strokes form a domain object, and merges the best object hypotheses to form a coherent solution. In the evaluation of each stroke subset, we use several trainable object models. The order construction and segmentation algorithms that we present are trainable, hence they can adapt to different datasets, domains, and drawing styles. Therefore, there is no need for tedious and laborious parameter tuning for new sketch domains and drawing styles.

When combined, the order construction and segmentation algorithms form our offline sketch recognition framework. Fig. 1 shows the components of our framework. The order construction module is comprised of a sibling model and a graph algorithm. The segmentation / classification module contains our novel segmentation algorithm and trainable object model.

We have four key contributions. First, we establish that the intrinsic property which makes online sketches amenable to efficient segmentation is not stroke orderings. Stroke ordering information in online sketches is useful, because it conveniently allows building contiguity-preserving stroke groups. If such groupings can be extracted for offline sketches, efficient segmentation and recognition algorithms become applicable. Second, we present a method for constructing contiguity-preserving total and partial orderings of strokes for offline sketches. Third, we push the state of the art in segmentation forward by formulating a segmentation algorithm that works with totally ordered as well as partially ordered sets of contiguity-preserving strokes. Fourth, we combine the order construction and segmentation algorithms together to obtain a complete sketch recognition/segmentation framework. Through rigorous evaluation, we demonstrate that the combined system is efficient and accurate as promised.

RELATED WORK

In the literature, sketches are represented either as an image or as a sequence of strokes. When sketches are treated as an image, single object sketch recognition problem is easily solved using CNN-based architectures. Sketch-a-Net is the first CNN-based network designed for sketch recognition [5]. It is based on the AlexNet architecture, and adapted for sketch images. Its classification accuracy on TU-Berlin dataset surpasses human performance. When a sketch is considered as a sequence of strokes, temporal models such as RNNs become applicable for sketch recognition problem. Ha et all [6] proposed SketchRNN, an RNN-based variational autoencoder that can generate sketches. This work inspired subsequent work to use stroke sequences as input. Instead of pure image and temporal architectures, some architectures combine the temporal and image architectures. For instance, Wang et al. [7] use a two-branch networks for retrieval and recognition purposes. The image format of an input sketch is fed to a CNN branch and the corresponding stroke sequence is fed into a parallel RNN branch. A concatenation layer at last fuses the feature representations from the two branches.

Deep learning methods generally require large-scale datasets. The Tu-Berlin, Google QuickDraw, SPG, SketchSeg are commonly used by deep learning methods. These datasets contain a single object per sketch. For this reason, studies utilizing these datasets either focus on the recognition of a single object, or they decompose a single object into semantically valid parts [8]. In this work, we focus on the multi-object sketch segmentation instead of single object recognition and decomposition. Our datasets contain multiple objects. Each sketch in the datasets is unique in terms of object types, number of objects and spatial occurrences.

For the multi-object sketch segmentation, Ouyang et al. [9] first construct a Conditional Random Field (CRF) over primitives and object candidates and generate the object candidates using a spatially and temporarily bounded search around the given primitive. Then, they perform approximate Loopy Belief Propagation on the

CRF. Finally, they combine the object candidates into a coherent solution using the agglomerative clustering algorithm. Their method relies on both temporal and spatial contiguity assumptions. Stahovich et al. [10] first classify each stroke and stroke pair, then, form the object clusters by chaining the strokes according to these labels. Delaye et al. [11] propose a clustering algorithm and compute a distance metric between pairs of primitives. Then, they perform a Single Linkage Agglomerative Clustering (SLAC) and apply a threshold on the linkages to form the domain objects.

Taking advantage of the sparsity of sketches, several methods try to reduce the complexity of object level sketch segmentation. For instance, Arandjelovic et al. assume that siblings are contiguous in the drawing order and apply 1D dynamic programming for segmentation [4]. Using a dynamic programming algorithm, they find an optimal segmentation in $O(n^2)$ under the temporal contiguity assumption. Feng et al. relax the contiguity assumption and they apply 2D dynamic programming for segmentation [3]. However, the degree of interspersing between siblings is limited in this algorithm. Our framework extends the ability to compute optimal sketch recognition solution in polynomial time from online to offline sketches, by taking advantage of sketch specific properties.

If temporal information is not available, we use an offline solution for segmentation. As an earlier example of an offline method, Shilman formulates the offline sketch recognition as an optimization problem and performs a spatially bounded search to detect objects on a proximity graph [12]. Shilman's method formulates the combined segmentation cost as an additive function and uses a hashing mechanism to reuse the calculated symbol costs. It constrains the distances between siblings and it is based on the spatial contiguity assumption. Several offline recognition systems use domain-specific rule-based approaches for offline recognition. For instance, Wu et al.[13] reduce the number of object candidates using a heuristic called "shapeness estimation", which is based on the observation that most of the hand-drawn shapes have well-defined closed boundaries. Some recent offline sketch recognition applications also use the shapeness estimation method [14]. However, in many domains, hand-drawn shapes may not have closed boundaries. Flow2Code presents an offline sketch recognition application [15]. In this application, users draw their flowcharts directly on paper, take a photo of the flowchart and their offline recognition system generates and executes the resulting code.

State-of-the art multi-object sketch segmentation methods generally formulate object level sketch segmentation as a detection problem. For instance, in ImageCLEFdrawnUI 2020 task [16], the winner segmentation method uses Faster R-CNN with Resnet-50 to segment sketch images of user interfaces. However, sketch recognition is different from object detection in photographic images. Sketch data is highly sparse. It allows us to find the optimal solution efficiently, instead of brute force search.

Stroke ordering methods have been proposed for animating drawings of artistic sketches and handwriting recognition. For example, Fu et al.[17] find a reasonable stroke order that look plausible to a human viewer. They formulate the ordering of strokes as finding a Hamiltonian path on a graph encoding rule-based characteristics of strokes. In this work, instead of finding a plausible stroke order, we find a stroke order that preserves the contiguity of object strokes.

APPROACH

In the sketch recognition literature, problem statement and terminology differ across methods. For this reason, we first present the terminology and conventions.

Terminology

A *sketch* is a freehand drawing consisting of a predefined set of symbols, which we refer to as *domain objects*. For example, the wire, resistor, capacitor, and transistor are objects of the sketch in Fig. 1. A sketch consists of multiple *strokes*, which are set of points sampled between pen-down and pen-up events. Each object in a sketch may consist of multiple strokes, or a stroke may span multiple objects. A *sibling* is pair of strokes that resides in the same domain object. The *stroke order* is a total or partial order of strokes. A stroke order preserves the contiguity if it connects all siblings to each other. We represent a total order

and partial order by a chain or a tree respectively. In our framework, direction of the order is not a consideration as long as it preserves the sibling contiguity. We use the term "stroke order" to draw a connection to online drawing order.

Sketch recognition is generally composed of three sequential processes: fragmentation, segmentation, and classification.

Fragmentation splits the strokes into geometric *primitives*, such as arcs and lines. A primitive is a simple geometric shape that do not span multiple objects. Fragmentation converts a sketch S into an unordered set of primitives, $S = p_1, p_2, ..., p_n$. The sketch in Fig. 1 consists of line primitives. In this study, for simplicity, we assume that the input sketch is already fragmented into primitives. We use the term *primitive* and *stroke* interchangeably to refer the primitives.

Segmentation finds the disjoint sets of primitives constituting the domain objects. It produces k groups, $G = g_1, g_2, ..., g_k$. There exist exponentially many possible primitive groupings, or segmentation solutions for a single sketch, and only a single solution complies with the ground truth, which we refer to as the *optimal segmentation*.

Classification determines the types of objects that each primitive group represents, and it assigns labels $L = l_1, l_2, ..., l_k$ to each primitive group. Classification is a well-studied topic in machine learning literature. In this study, we segment and classify each primitive group at the same time in the segmentation/classification module.

Problem Statement

The purpose of segmentation is to discover the optimal partitioning of sketch into domain objects. A segmentation is optimal, if it is fully compliant with the human-annotated objects in the sketch. For a given sketch and its segmentation G, we may assess its optimality in terms of $P(O|G_i)$, which is the likelihood of object candidate G_i in G constituting a valid sketch object.

$$P(\mathcal{O}|G) = \prod_{G_i \in G} P(O|G_i) \tag{1}$$

where $\mathcal{O}$ denotes the segmentation optimality, and O denotes that G_i corresponds to a valid sketch object. Then, we can define the optimality criterion for the segmentation as follows:

$$\begin{aligned} G_{opt} &= \arg\max_{G \in \mathcal{G}} \; P(\mathcal{O}|G) \\ &= \arg\max_{G \in \mathcal{G}} \; \prod_{G_i \in G} P(O|G_i) \end{aligned} \tag{2}$$

where $\mathcal{G}$ represents the set of all possible segmentation solutions.

To avoid the numerical underflow of probabilities, we replace the probability term with the log probability:

$$G_{opt} = \arg\min_{G \in \mathcal{G}} \; -\sum_{G_i \in G} log \; P(O|G_i) \tag{3}$$

Given the cost function in Eq.3, we can attempt to enumerate all possible segmentation solutions $\mathcal{G}$ of a given sketch S, and declare the segmentation with the smallest cost as optimal. However, exhaustive enumeration of all possible groupings is not feasible, because $\mathcal{G}$ grows exponentially with the number of primitives in the sketch.

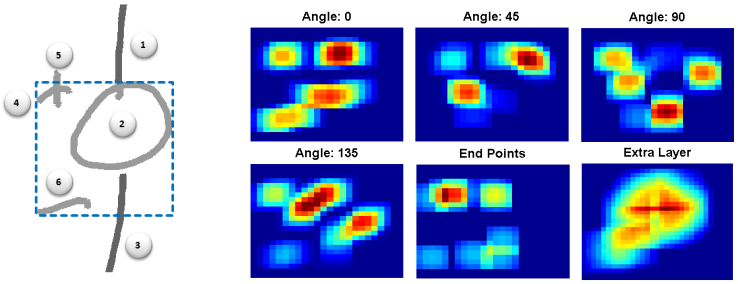
In this study, instead of an exhaustive search for an optimal solution, we segment a given sketch using a two-stage recognition framework as illustrated in Fig. 1. We explain the details of each module and their components in the next section.

Stroke Order Construction

The construction of a contiguity-preserving stroke order is the key component of our sketch recognition framework. It consists of two sequential processes: (1) Sibling score estimation (2) Construction of the stroke order.

Sibling Score Estimation

We use two different sibling models to estimate the likelihood of a primitive pair being a sibling. The first is a simple model based on the spatial distance between primitives, and the second model is a regression model with spatial features of primitives. In this study, we do not use temporal sibling models, because temporal information is not available in offline sketches.



(a) (b)

Figure 2: PairIDM feature for p_2 and p_6 of the voltage source object.

Distance Model:
The distance model assumes that siblings are spatially close to each other. It estimates the sibling scores as follows:

$$C_d(p_i, p_j) = 1 - \frac{d(p_i, p_j)}{D_{max}} \tag{4}$$

where $d(p_i, p_j)$ is the minimum Euclidean distance between p_i and p_j, and D_{max} is the diagonal length of the sketch's bounding box.

Regression Model:
The regression model uses PairIDM feature, which we created by modifying the original Image Deformation Model (IDM)[18], to estimate sibling likelihood. To extract the original IDM feature, the strokes are converted into a scaled raster image. Then, the image is filtered in different directions (0, 45, 90, 135 degrees). In addition, a smoothed image of the stroke end points is generated. Finally, all filtered images are stacked into a feature vector. In PairIDM, we extend the IDM with an extra layer, which consists of a scaled and smoothed image of the given primitive pair. In the final pair representation, original IDM features encode the context, and new pair layer encodes the spatial layout of the primitive pair. Fig. 2 shows the PairIDM layers for p_2 and p_6. Our PairIDM implementation consists of six images of 9x9 pixels. We refer the reader to the original paper[18] for the details of IDM feature.

After we extract PairIDM features, we apply a PCA transformation and train a Support Vector Regression (SVR) model with a Radial Basis Function (RBF) kernel. In the training of the regression model, we use sibling and non-sibling samples in the training set. We optimize feature extraction and regression model parameters by a grid search using a 5-fold cross validation.

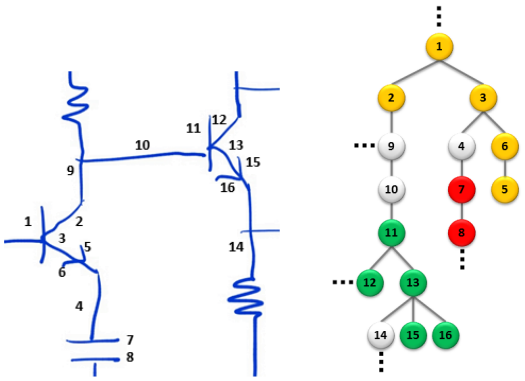
Construction of the Stroke Order
A stroke order can be represented as a chain or a tree, where the nodes represent the strokes and edges represent the order relation. In a contiguity-preserving stroke order, the sibling nodes are connected to each other over a non-blocking path.

In a contiguity-preserving stroke order, the sum of sibling scores is maximized. Instead of a maximization of the scores, we apply a reversed order min-max normalization to the sibling scores and find partial and total orders of strokes by score minimization. Minimization allows us to use standard algorithms such as MST and TSP in order construction.

Partial Order
A partial order may be represented by a spanning tree. Kruskal's algorithm builds a partial stroke order that preserves the contiguity MST between strokes.

If the scores of siblings are greater, then siblings come before non-siblings in the descending score order, and siblings are added to the spanning tree before non-siblings. Hence, when scores favor siblings over non-siblings, Kruskal's algorithm creates a contiguity preserving stroke order. Fig. 3 shows a sample sketch and partial order represented by a tree.

Total Order
A total order may be represented by a Hamiltonian path that visits every node in the graph. If the edges between siblings always have a higher scores, than a TSP solution builds up a total order that preserves the contiguity between siblings.



(a) (b)

Figure 3: A portion of a sketch and the partial order of its strokes. The siblings of each object in the drawing are connected to each other, forming sub-trees. The dots on the tree represent the extension points.

Graph Algorithm Implementations

In summary, if sibling estimates favor siblings over non-siblings, then MST or TSP solutions construct a contiguity-preserving stroke order. In this study, we use three different graph algorithms to construct the stroke order.

- *Kruskal's Solution to MST*: The algorithm's complexity is $O(n^2\ log(n))$ for a fully connected graph.
- *Exact Solution to TSP*: Finding an exact solution to the TSP problem is NP-complete. However, recent TSP solvers have tolerable run-times up to a few hundred nodes. In our implementation, we use Concorde TSP solver.
- *Controlled Random Search Solution to TSP*: We also use a genetic algorithm based solution to the TSP problem. The complexity of the algorithm is $O(n^2)$. The number of iterations is a constant and cost calculation at each iteration requires $O(n^2)$ operations.

Segmentation & Classification

Given a stroke order, the segmentation/classification module iteratively builds G_{opt} based on Eq.3. For each sub-problem, it first calculates an optimal solution using a bottom-up approach. Then, it stores the best solution for each sub-problem. Finally, it backtracks the solutions to sub-problems and combines them to produce G_{opt}. We can solve the segmentation problem using dynamic programming because the problem exhibits the optimal substructure property of dynamic programming. The dynamic programming lets us find an optimal solution G_{opt} in polynomial time. However, it is not possible to apply a dynamic programming unless an optimal substructure property holds for the problem. For this reason, we first prove that the search for G_{opt} in an ordered set of strokes exhibits the optimal substructure property according to Eq.3.

Suppose that an optimal segmentation splits the tree at a set of boundary primitives P_B. For instance, $P_B = \{p_5, p_{11}, p_{22}, p_{23}\}$ for transistor object in Fig. 3(b). Then, the segmentation of sub-trees rooted at each element of P_B must also be optimal. Because, otherwise, a less costly segmentation of the sub-trees could have been substituted in, to obtain a better segmentation, which is a contradiction. Thus, we can build an optimal solution by solving the sub-problems using dynamic programming based on the optimality criterion in Eq.3.

We can define the cost of G_{opt} recursively in terms of optimal solutions to the sub-problems. If there is no primitive, which is the trivial case, the cost is zero. Assuming that segmentation of a sub-tree with root p_j splits the tree at a set of boundary primitives P_B and it creates the object G_j, the cost of the segmentation is:

$$C[j] = \left(\sum_{p \in P_B \text{ of } G_j} C[p] \right) - log\ P(O|G_j) \quad (5)$$

Thus, our recursive definition for the minimum cost is:

$$C_{opt}[j] = \min_{\text{for each subtree of } p_j} (C[j]) \quad (6)$$

Instead of computing the solution by recurrence, we can iteratively compute the optimal cost using a bottom-up approach. We estimate the term $P(O|G_j)$ in Eq.3 using a trainable object model.

Object Model

$P(O|G_i)$ is the likelihood that primitive group G_i constitutes a valid domain object. We define

$P(O|G_i)$ as the joint probability of three random variables:

$$\begin{aligned} P(O|G_i) &= P(T, V, D|G_i) \\ &= P(V|G_i)\ P(D|V, G_i)\ P(T|V, D, G_i) \end{aligned} \tag{7}$$

where

- $P(V|G_i)$ is the probability that given primitive group constitutes a valid object candidate.
- $P(D|V, G_i)$ is the probability that given valid object candidate represents a symbol.
- $P(C|V, D, G_i)$ is the probability that given symbol is an instance of a particular object class.

Each random variable is designed to assess the quality of G_i. We train three probabilistic classifiers to estimate each term in Eq.7.

- *Validity Classifier:* The validity classifier estimates $P(V|G_i)$, which is the probability of a set of primitives constituting a valid object candidate. It checks whether the given set of primitives is valid using simple features such as the number of primitives and the size of the bounding box. It produces binary results, which we use to filter the invalid object candidates. It is a one-class classifier trained with positive examples. We do not apply a set of hard-coded constraints for object candidates, but our validity classifier learns the constraints from the training data.
- *Object/Non-object Model:* The second model uses IDM with a PCA transformation to estimate $P(D|V, G_i)$, which is the probability that the given valid object candidate G_i is a domain object. To generate the training instances, we use the valid object candidates in the training, which are detected by the validity classifier.
- *Symbol Classifier:* The third model estimates the $P(T|V, D, G_i)$, which is the probability that the given object candidate is an instance of a specific object type. To train the multi-class classifier, we extract IDM feature from each object in the training set. The symbol classifier does not contain a reject class because object/non-object model filters the non-object primitive groups.

We train all models using LIBSVM implementation of Support Vector Machine (SVM) with RBF kernel. We optimize parameters by a grid search with 5-fold cross validation. We estimate $P(O|G_j)$ according to Eq. 7.

Simultaneously with the segmentation, the symbol classifier estimates the labels associated with each primitive group G_i.

EVALUATION

To evaluate our sketch recognition framework, we conducted several experiments on three datasets from different domains. In all experiments, we trained sibling and object models with the same training instances and tested the recognition accuracy on the test dataset. We used two different cross-validation schemes: 5x2cv and 5x10cv. We prefer 5x2cv scheme for statistical tests because it has an acceptable type-I error. We also present the results of 5x10cv to compare the results with those of other methods in the literature.

In the experiments, in addition to six stroke order construction methods, we use two additional methods for evaluation purposes: an upper-bound and a baseline method.

Upper-Bound Method

The Upper-Bound method constructs a valid primitive chain using the sibling scores based on human annotations and exact TSP solution. If we assign binary sibling scores according to the object annotations, then we may end up with a trivial stroke order which connects the distant objects. To avoid such an easily recognizable stroke ordering, we add a normalized distance element C_d of Eq.4 to the sibling scores based on the ground- truth. We set the sibling score of a primitive pair p_i and p_j as follows:

$$C_u(p_i, p_j) = \begin{cases} 1 + C_d, & \text{p}_\text{i} \text{ and } \text{p}_\text{j} \text{ are sibling} \\ C_d, & \text{otherwise} \end{cases} \tag{8}$$

Sibling scores estimated by Eq.8 guarantee the construction of a contiguity-preserving stroke order. The order also emulates a common drawing style, in which the user continues with the next closest object after finishing the current one.

Table 1: Properties of the stroke order construction methods

	Graph Algorithm	Sibling Model	Complexity	Optimality
Upper-Bound	Exact TSP	Annotated Values	$O(n^2\ 2^n)$	Yes
MST	Kruskal	Regression	$O(n^2\ log(n))$	Yes
TSP	Exact TSP	Regression	$O(n^2\ 2^n)$	Yes
TSPGA	CRS[1]	Regression	$O(n^2)$	No
MSTD	Kruskal	Distance	$O(n^2\ log(n))$	Yes
TSPD	Exact TSP	Distance	$O(n^2\ 2^n)$	Yes
TSPDGA	CRS[1]	Distance	$O(n^2)$	No
Closest-Primitive	Greedy	Distance	$O(n)$	No

Table 2: Statistics of the datasets

	EC	FC	FA
#Participants	10	35	25
#Sketches	110	419	300
#Object Types	10	6	3
Total #Objects	3216	6128	4314
#Strokes	3233	14804	7449
#Primitives	6726	14804	7449
Avg.#Objects	29	14	14
Avg.#Strokes	29	35	25
Avg.#Primitives	61	35	25

Closest-Primitive Method

The baseline method, which we refer to as Closest-Primitive, first adds a random primitive to the sibling chain, and then it continues to grow the chain with the closest primitive. The Closest-Primitive method is greedy, and it uses a distance-based sibling model. It may produce a contiguity preserving stroke order if objects do not contain branches and gaps between siblings. Table 1 presents a comparison among these different stroke order construction methods.

Datasets

It is challenging to construct a contiguity-preserving stroke order for sketches with cyclic connections. For this reason, we intentionally select three datasets with an excessive number of cycles. All three datasets are publicly available.

- *Analog Electrical Circuit Dataset*: The EC dataset contains 110 sketches of Analog Electrical Circuits by 10 users. The wire, resistor, capacitor, ground, battery, transistors, voltage and current sources are the objects in the EC dataset. It is an element of Experimental Test Corpus of Hand Annotated Sketches (ETCHA) [19].
- *Flow-chart Dataset*: The FC dataset contains 419 sketches of flow-charts by 35 users [20]. The process, decision, data, connector, terminator, and arrow are the objects in the FC dataset.
- *Finite Automata Dataset*: The FA dataset contains 300 finite-automata sketches by 25 users [21]. The state, final state, and arrow are the objects in FA dataset.

We present representative sketches from three datasets in Fig. 4 along with their statistics in Table 2.

Accuracy Metrics

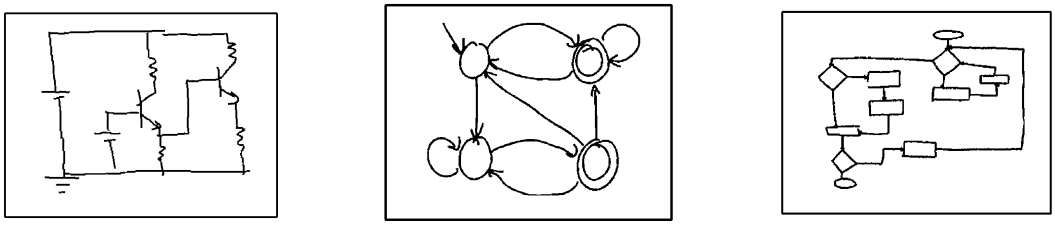
Several methods have been proposed for the evaluation of sketch recognizers in the literature. Some methods determine the accuracy by comparing the number of points in the objects [3]. Others calculate the ratio of overlap for the bounding boxes [9]. In this study, unless stated otherwise, we apply the all-or-nothing (AoN) accuracy metric, where the number of points must be exactly the same with the ground-truth. In the AoN accuracy, the recognition output is considered incorrect even if a single primitive does not comply with human annotations. In the accuracy comparison, we also use intersection over union (IoU), because methods in the literature adopt the same metric for evaluation.

In accuracy reports, we first calculate the accuracy separately for each object type and then report their average. Averaging over object types helps us avoid the dominance of the frequent types in accuracy reports. For example, wires constitute almost half of the EC dataset. Thus, accuracy averaging without an object categorization of types may be misleading.

Sibling Connectivity

We can segment and recognize an object correctly only if all its siblings are connected in the given stroke order. The ratio of the recognizable

[1]Controlled Random Search, (a.k.a Genetic Algorithm)



(a) (b) (c)

Figure 4: Representative sketches from the EC, FA and FC datasets.

objects after stroke order construction gives us the sibling connectivity, which is an upper bound for recognition recall. We calculate the sibling connectivity for an individual object class as the ratio of correctly ordered objects to all objects of the same class. Then, we report the overall sibling connectivity by averaging the connectivity values calculated for each object class.

Segmentation/Recognition Accuracy

To calculate the segmentation and recognition accuracy metrics on dataset D, we first perform the segmentation and recognition using method M, and then we calculate the precision, recall and F1 scores for each object type in the dataset. Then, we report their average as the accuracy of a method M on dataset D.

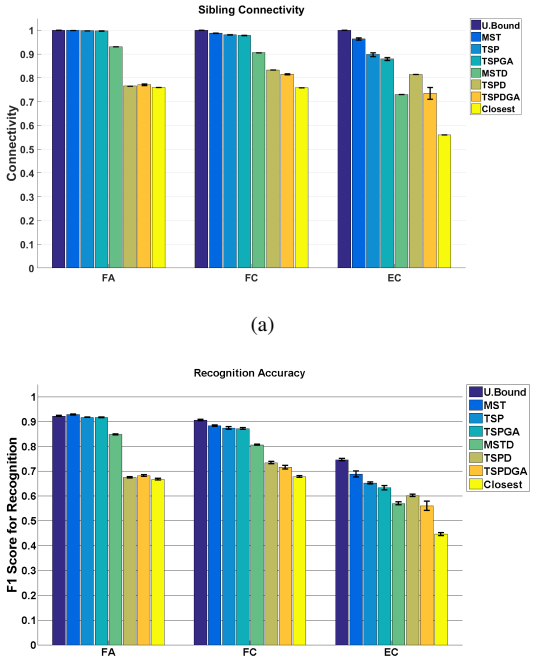
Accuracy Evaluation

Fig. 5 and Table 3 show the average sibling connectivity and F1 scores for recognition. Results reveal that the ranks of different stroke order construction methods are similar for all datasets. In addition, sibling models based on PairIDM regression outperform the distance based models. For a more sound analysis, we conduct several statistical tests to determine the effect of stroke order construction methods on recognition accuracy.

Evaluation of Stroke Order Construction Methods

The Friedman test indicated a significant difference between the recognition accuracies produced by different stroke order construction methods, $\tilde{\chi}^2(7)=20.00$, $p=0.006$. A Nemenyi post hoc test also indicated that the MST and the Upper-Bound

[2]U.Bound method uses a perfect stroke ordering, and it is used only for evaluation.



(a)

(b)

Figure 5: Average F1 scores obtained by a 5x10cv across repetitions. Error bars show one standard deviation of accuracies.

method produced significantly higher accuracy than that of baseline, $p<0.05$. In addition, the baseline method recognition produced significantly lower accuracy than those of all other methods, $p<0.05$.

Evaluation of Sibling Models

Wilcoxon signed rank test revealed a statistically significant median increase in the recognition accuracy (*Mdn*=0.108) when methods used the regression model (*Mdn*=0.857), instead of a distance model (*Mdn*=0.677), Z= -2.666, p= 0.008.

Table 3: Averages of sibling connectivity and recognition F1 scores across repetitions of 5x10cv

	Connectivity			F1 Score(AoN)			F1 Score (IoU = 0.5)		
	EC	FA	FC	EC	FA	FC	EC	FA	FC
U.Bound[2]	**1.00**	**1.00**	**1.00**	**0.75**	0.92	**0.91**	**0.84**	0.94	**0.94**
MST	0.96	**1.00**	0.99	0.69	**0.93**	0.88	0.79	**0.95**	**0.94**
TSP	0.90	**1.00**	0.98	0.65	0.92	0.87	0.78	0.94	0.93
TSPGA	0.88	**1.00**	0.98	0.63	0.92	0.87	0.78	0.94	0.93
MSTD	0.73	0.93	0.91	0.57	0.85	0.81	0.71	0.89	0.88
TSPD	0.81	0.77	0.83	0.60	0.67	0.73	0.73	0.70	0.82
TSPDGA	0.74	0.77	0.82	0.56	0.68	0.72	0.69	0.71	0.79
Closest-Primitive	0.56	0.76	0.76	0.45	0.67	0.68	0.56	0.70	0.78

Table 4: F1 score comparison for EC, FA and FC datasets.

Method	EC	FA	FC
MST (IoU = 0.5)	**0.79**	0.94	0.94
TSP (IoU = 0.5)	0.78	0.94	0.93
Faster R-CNN (IoU = 0.5)	0.61	**0.97**	**0.96**
MST (IoU = 0.90)	**0.74**	0.94	0.92
TSP (IoU = 0.90)	0.71	0.93	0.91
Faster R-CNN (IoU = 0.90)	0.50	**0.96**	**0.95**
MST (IoU = 1.0)	**0.69**	**0.93**	**0.88**
TSP (IoU = 1.0)	0.65	0.92	0.87
Faster R-CNN (IoU = 0.97)[3]	0.22	0.83	0.76

Table 5: Accuracy comparison with online methods. Table shows the recall values for the FA and FC as they are reported.

Method	FA (IoU = 1.0)	FC (IoU = 1.0)
MST	0.93	**0.88**
TSP	0.92	0.87
Bresler[21]	0.94	0.84
Carton[22]	-	0.79
Lemaitre[23]	-	0.75
Delaye[11] [4]	**0.97**	0.77

Evaluation of Graph Algorithms

Friedman test revealed that recognition accuracy was significantly different, $\tilde{\chi}^2(2)=7$, $p=0.030$. Post hoc analysis with Bonferroni correction also revealed a significant difference for MST ($Mdn=0.816$) and TSPGA ($Mdn=0.690$) ($p=0.028$), but MST-TSP and TSP-TSPGA pairs were not significantly different.

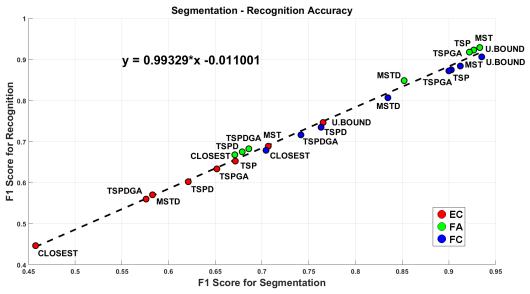
Component based Error Analysis

Each component in our framework contributes to the recognition errors to a varying extent. Fig. 6 shows the relation between the input and output accuracy of the segmentation and recognition. Figure-6(a) shows that the segmentation and recognition accuracies are nearly equal to each other for all datasets, which means that objects are generally classified correctly if they are correctly segmented. Fig. 6(b) reveals a linear relationship between stroke order construction and segmentation accuracy. Recognition accuracy increases linearly with stroke order construction accuracy for all datasets in a similar trend. Figure-6(b) also shows that segmentation is the common source for recognition errors. The error introduced by the stroke order construction and recognition is negligible when compared to the errors of the segmentation. Since DP based algorithm is proven to be optimal, we can attribute the segmentation error to the object model.

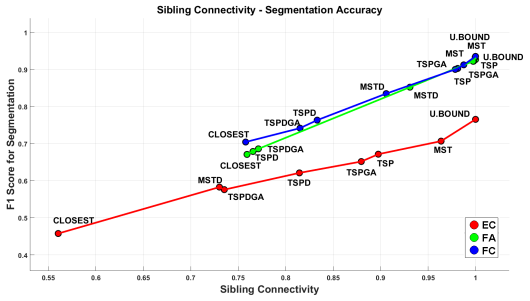
Accuracy Comparison

In ImageCLEFdrawnUI 2020 task, Faster R-CNN with Resnet-50 came out as the most accurate object detection method for UI sketches. The same method was also applied to flowchart domain with success by Julca-Aguilar et al. [24]. Table 4 compares Faster R-CNN with our MST and TSP solutions for the EC, FA and FC datasets. Based on its success on several sketch recognition tasks, we choose Faster R-CNN for accuracy comparison.

[3]Faster R-CNN method is a pure raster method. It is not aware of the exact primitive geometries. Consequently, the accuracy is approximately 0%. for IoU=1.0 setting. To be fair, we report its accuracy with IoU=0.97 instead of IoU=1.0.



(a)



(b)

Figure 6: The interaction plots for accuracies in segmentation and recognition. In all datasets, segmentation and recognition accuracies are nearly equal to each other.

In the experiments with Faster R-CNN, we randomly choose 80% of the sketches as the training set, and remaining 20% as the test set. When IoU ratio is set to 0.5 and 0.9, Faster R-CNN have higher accuracy than MST and TSP in FC and FA datasets. However, when overlap criteria becomes more strict, our methods surpass the accuracy of Faster R-CNN in all datasets. These results show that TSP and MST is a proper choice for applications which are sensitive to the exact locations and dimensions of the objects in the sketch.

Training of deep networks generally requires large-scale datasets. Faster R-CNN's accuracy on EC may benefit from an increase in the training dataset. However, when the size of FA and FC made equated to the size of EC (110 sketch samples by a random selection), the accuracy degrades on the order of one percent. As shown in Table 4, our method produces accurate results with all datasets.

Arrow R-CNN[25] extends the Faster R-CNN object detection system with an arrow keypoint predictor. Arrow R-CNN is a domain specific object detector that works in the domains where arrows are used to connect objects. It produces 93% recall FC dataset using a domain specific knowledge. However, it is restricted to sketches that use arrows as connectors.

Thus far, we compared our work with object detectors. However, we note that object detection is fundamentally different from sketch parsing or segmentation. A sketch segmentation algorithm outputs a coherent solution that labels all primitives with a single label. The detection output does not need to be coherent from the segmentation perspective. A single primitive may belong to multiple objects or some primitives may be out of all detected objects.

Sketch segmentation methods evaluated on the FC and FA dataset are generally online. They make use of either stroke ordering or temporal information in recognition. We compare our offline recognition framework with these methods in Table 5. Studies indicate a substantial increase in the recognition accuracy when sketch recognizers use temporal information [4]. For example, Delaye [11] reports a 25% increase in the recognition accuracy, when the stroke ordering is used on the FC dataset. However, notably, the accuracy of our purely offline methods either matches with the online state of the art methods or surpasses them. We report the average accuracy either obtained by 5x2cv or 5x10cv. In selecting the cross-validation type, we base our decision on the evaluation method of the other method.

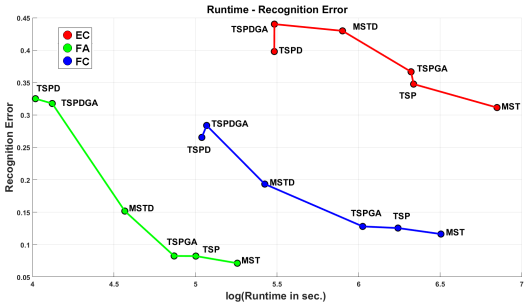


Figure 7: The trade-off between run-time and recognition accuracy. Recognition error and runtime values are the averages of 5-times repeated 10-fold cross-validation (5x10cv).

[4]Delaye et al. do not report recall per object type. Instead, they report the overall recall.

Run-time Comparison

In order to facilitate real-time interaction, a sketch recognizer should be efficient and fast. In this section, we analyze the contribution of stroke order construction and segmentation to the run-time.

- *Stroke Order Construction:* The complexity of order construction is different for each stroke order construction method. The complexity of an exact TSP solution is exponential, while the complexity of MST is $O(n^2\ log(n))$. CRS solution to TSP has a complexity of $O(n^2)$, but it is not an exact solution to TSP.
- *Segmentation/Classification:* The complexity of the segmentation depends on the structure of the stroke order because the number of sub-trees rooted at a specific node varies with the structure of stroke order. If the tree is a chain (similar to the TSP solution), then the number of sub-trees is $O(n^2)$. For MST based solutions, the worst case is a star topology, in which the number of sub-trees grows exponentially with the number of vertices.

As illustrated in Fig. 7, the recognition accuracy of the stroke order construction methods and their total run-time exhibit a trade-off. The optimal choice for a stroke order construction method differs across applications.

The training of Arrow R-CNN and Faster R-CNN on FC dataset takes about 20 hours on GPU. Inference on a single image takes about 100 ms on GPU and 3 seconds on a CPU[25]. For the same FC dataset, depending on the order construction method, training of our methods takes maximum of 2 hours on a CPU. Segmentation and recognition run-times depends on order construction methods, which are detailed in Fig. 7.

LIMITATIONS & ASSUMPTIONS

- Fig. 8 shows some failure cases for our algorithm. Fig. 8(a) shows that two wires on the right-hand side are have incorrectly been segmented as a capacitor. Similarly, arrow of the final state's self loop in Fig. 8(b) has been misclassified as a state symbol. One possible solution to address such cases would be to use context features in object models.
- We assume that input sketches do not have any accidental and overdrawn strokes. Our datasets do not contain accidental and overdrawn strokes. However, in practice, accidental and overdrawn strokes must be eliminated in a pre-processing step, before the sketch is fed into order construction.
- In this work, we focus on the efficiency of finding the optimal solution. For simplicity, we use traditional SVM-based models with image-based features. The accuracy of both sibling and object models can be improved by deep models trained on large-scale datasets.
- The run-time our of framework depends on the vertex degree of the sibling graph. In MST solutions, run-time efficiency degrades, when the vertex degree increases. As a solution, we may limit the maximum vertex degree by applying a degree-constrained spanning tree algorithm. The trade-off between run-time and accuracy can be used to determine the an optimal vertex degree.
- Experiments show that if location and dimension errors are tolerable, we may resort to object detection methods such as Faster R-CNN.

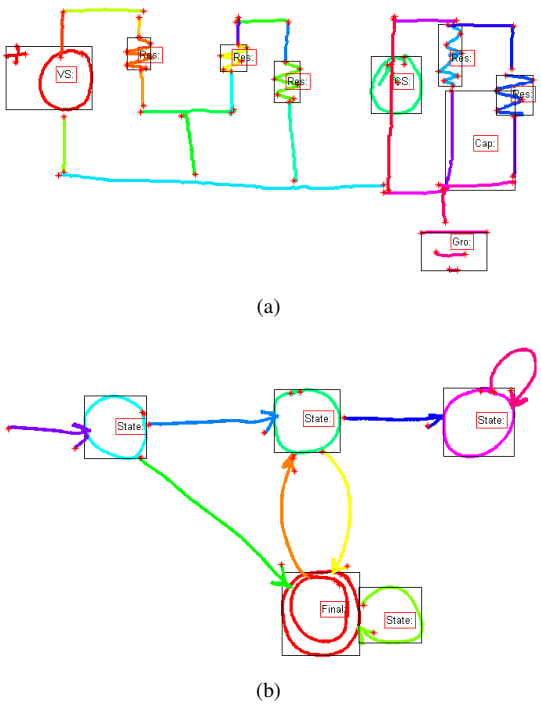


Figure 8: Recognition failures of our method. Context features may help eliminating segmentation and classification errors.

Object detection methods result in a reasonable accuracy for 0.5 IoU ratio criteria. However, if the application requires finding the exact strokes comprising the objects, our framework provides an efficient way finding the optimal solution with high degree of overlap.

DISCUSSION & FUTURE WORK

We presented an offline sketch recognition framework inspired by online recognition methods' ability to use temporal stroke orders. We addressed the absence of temporal information by constructing contiguity-preserving stroke orderings. Contiguity-preserving stroke orders are extracted using two subsystems: (1) A sibling model trained to assess if two primitives are likely to be part of the same object instance, (2) A graph algorithm that constructs partial/total orders using the sibling scores. We further demonstrated that the stroke orders produced by these components can be combined within a carefully crafted recognition scheme using dynamic programming.

Through our work, we also introduced three important problems that are worthy of further research on their own right. In particular, we see the following topics to attract attention as individual threads of research: (1) improved sibling models for assessing if pairs (or more generally tuples) of primitives are likely to be part of the same object, (2) alternative methods to construct total stroke orders representing single drawing orders, and (3) methods to construct partial orders representing multiple drawing orders.

As illustrated in Fig. 7, various combinations of the sibling models and the graph algorithms put us on different points in the accuracy-speed space. There also appears to be a clear trade off between speed and accuracy. Finding other combinations of improved models that give us better and richer ways of trading off speed and accuracy remains as an open question requiring further research.

The ability to produce contiguity-preserving stroke orders comes as a convenient side product of our work. This is a significant contribution on its own right, and is likely to find many practical uses. For example, it can be used to synthesize animations depicting plausible drawing orders for offline sketches, very much in the sprit of [17].

Finally, analysis of our framework has revealed that the majority of the recognition errors are due to the lack of highly accurate isolated object recognizers. This highlights the importance of accurate discrimination of objects from non-objects.

REFERENCES

1. P. Van Sommers, *Drawing and cognition: Descriptive and experimental studies of graphic production processes.* Cambridge University Press, 1984.
2. T. M. Sezgin and R. Davis, "Sketch recognition in interspersed drawings using time-based graphical models," *Computers & Graphics*, vol. 32, no. 5, pp. 500–510, Oct. 2008.
3. G. Feng, C. Viard-Gaudin, and Z. Sun, "On-line hand-drawn electric circuit diagram recognition using 2d dynamic programming," *Pattern Recognition*, vol. 42, no. 12, pp. 3215–3223, 2009.
4. R. Arandjelović and T. M. Sezgin, "Sketch recognition by fusion of temporal and image-based features," *Pattern Recognition*, vol. 44, no. 6, pp. 1225–1234, 2011.
5. Q. Yu, Y. Yang, Y.-Z. Song, T. Xiang, and T. Hospedales, "Sketch-a-Net that Beats Humans," in *Procedings of the British Machine Vision Conference 2015*, no. i. British Machine Vision Association, jan 2015, pp. 7.1–7.12.
6. D. Ha and D. Eck, "A neural representation of sketch drawings," in *International Conference on Learning Representations*, 2018.
7. F. Wang, S. Lin, H. Wu, H. Li, R. Wang, X. Luo, and X. He, "Spfusionnet: Sketch segmentation using multi-modal data fusion," in *2019 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2019, pp. 1654–1659.
8. K. Kaiyrbekov and M. Sezgin, "Deep stroke-based sketched symbol reconstruction and segmentation," *IEEE Computer Graphics and Applications*, vol. 40, no. 1, pp. 112–126, 2020.
9. T. Y. Ouyang and R. Davis, "Chemink: a natural real-time recognition system for chemical drawings," in *Proceedings of the 16th international conference on Intelligent user interfaces*, ACM. New York, NY, USA: ACM, 2011, pp. 267–276.
10. T. F. Stahovich, E. J. Peterson, and H. Lin, "An efficient, classification-based approach for grouping pen strokes into objects," *Computers & Graphics*, vol. 42, pp. 14–30, 2014.
11. A. Delaye and K. Lee, "A flexible framework for online document segmentation by pairwise stroke distance learning," *Pattern Recognition*, vol. 48, no. 4, pp. 1193–1206, 2015.

12. M. Shilman and P. Viola, "Spatial recognition and grouping of text and graphics," in *Proceedings of the First Eurographics conference on Sketch-Based Interfaces and Modeling*, Eurographics Association. Geneve, Switzerland: Eurographics Association, 2004, pp. 91–95.
13. J. Wu, C. Wang, L. Zhang, and Y. Rui, "Offline sketch parsing via shapeness estimation," *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2015-January, pp. 1200–1207, 2015.
14. M. Bresler, D. Průša, and V. Hlaváč, "Recognizing off-line flowcharts by reconstructing strokes and using on-line recognition techniques," *Proceedings of International Conference on Frontiers in Handwriting Recognition, ICFHR*, pp. 48–53, 2017.
15. J. I. Herrera-Camara and T. Hammond, "Flow2Code: From hand-drawn flowcharts to code execution," *Proceedings - Sketch-Based Interfaces and Modeling, SBIM 2017 - Part of Expressive 2017*, 2017.
16. D. Fichou, R. Berari, P. Brie, M. Dogariu, L. D. Ştefan, M. G. Constantin, and B. Ionescu, "Overview of ImageCLEFdrawnUI 2020: The detection and recognition of hand drawn website uis task," in *CLEF2020 Working Notes*, ser. CEUR Workshop Proceedings. Thessaloniki, Greece: CEUR-WS.org <http://ceur-ws.org>, September 22-25 2020.
17. H. Fu, S. Zhou, L. Liu, and N. J. Mitra, "Animated construction of line drawings," *ACM Trans. Graph.*, vol. 30, no. 6, pp. 133:1–133:10, 2011.
18. T. Y. Ouyang and R. Davis, "A visual approach to sketched symbol recognition," in *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, ser. IJCAI'09, San Francisco, CA, USA, 2009, p. 1463–1468.
19. M. Oltmans, C. Alvarado, and R. Davis, "Etcha sketches: Lessons learned from collecting sketch data," *Making Pen-Based Interaction Intelligent and Natural*, vol. 1, pp. 134–140, 2004.
20. A.-M. Awal, G. Feng, H. Mouchere, and C. Viard-Gaudin, "First experiments on a new online handwritten flowchart database," in *IS&T/SPIE Electronic Imaging*, International Society for Optics and Photonics. Bellingham, WA, USA: SPIE, 2011, pp. 78 740A–78 740A.
21. M. Bresler, T. Van Phan, D. Prusa, M. Nakagawa, and V. Hlavác, "Recognition system for on-line sketched diagrams," in *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, IEEE. Washington DC, USA: IEEE, 2014, pp. 563–568.
22. C. Carton, A. Lemaitre, and B. Couasnon, "Fusion of statistical and structural information for flowchart recognition," in *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, IEEE. Washington DC, USA: IEEE, 2013, pp. 1210–1214.
23. A. Lemaitre, H. Mouchere, J. Camillerapp, and B. Coüasnon, "Interest of syntactic knowledge for on-line flowchart recognition," in *Graphics Recognition. New Trends and Challenges*. Berlin, Heidelberg: Springer, 2013, pp. 89–98.
24. F. D. Julca-Aguilar and N. S. Hirata, "Symbol detection in online handwritten graphics using faster R-CNN," *Proceedings - 13th IAPR International Workshop on Document Analysis Systems, DAS 2018*, pp. 151–156, 2018.
25. B. Schäfer and H. Stuckenschmidt, "Arrow r-cnn for flowchart recognition," in *2019 International Conference on Document Analysis and Recognition Workshops (ICDARW)*, vol. 1, 2019, pp. 7–13.

Recep Sinan Tümen graduated from the Turkish Naval Academy in 2003. He received his MS degree from Johns Hopkins University in 2007 and his Ph.D degree from Koç University in 2015. He is currently affiliated with Turkish Navy Research Center. His research interests include intelligent user interfaces and machine learning. Contact him at stumen@ku.edu.tr.

Metin Sezgin is currently an associate professor at Koç University, Istanbul, and directs the Intelligent User Interfaces (IUI) Laboratory. He received the Graduate (summa cum laude with honors) degree from Syracuse University in 1999. He received the M.S. and Ph.D. degrees from the Massachusetts Institute of Technology in 2001 and 2006. He subsequently joined the University of Cambridge as a postdoctoral research associate, and held a visiting researcher position at Harvard University in 2010. His research interests include intelligent human–computer interfaces and HCI applications of machine learning. He is particularly interested in applications of these technologies in building intelligent penbased interfaces. He is a member of IEEE. Contact him at mtsezgin@ku.edu.tr.

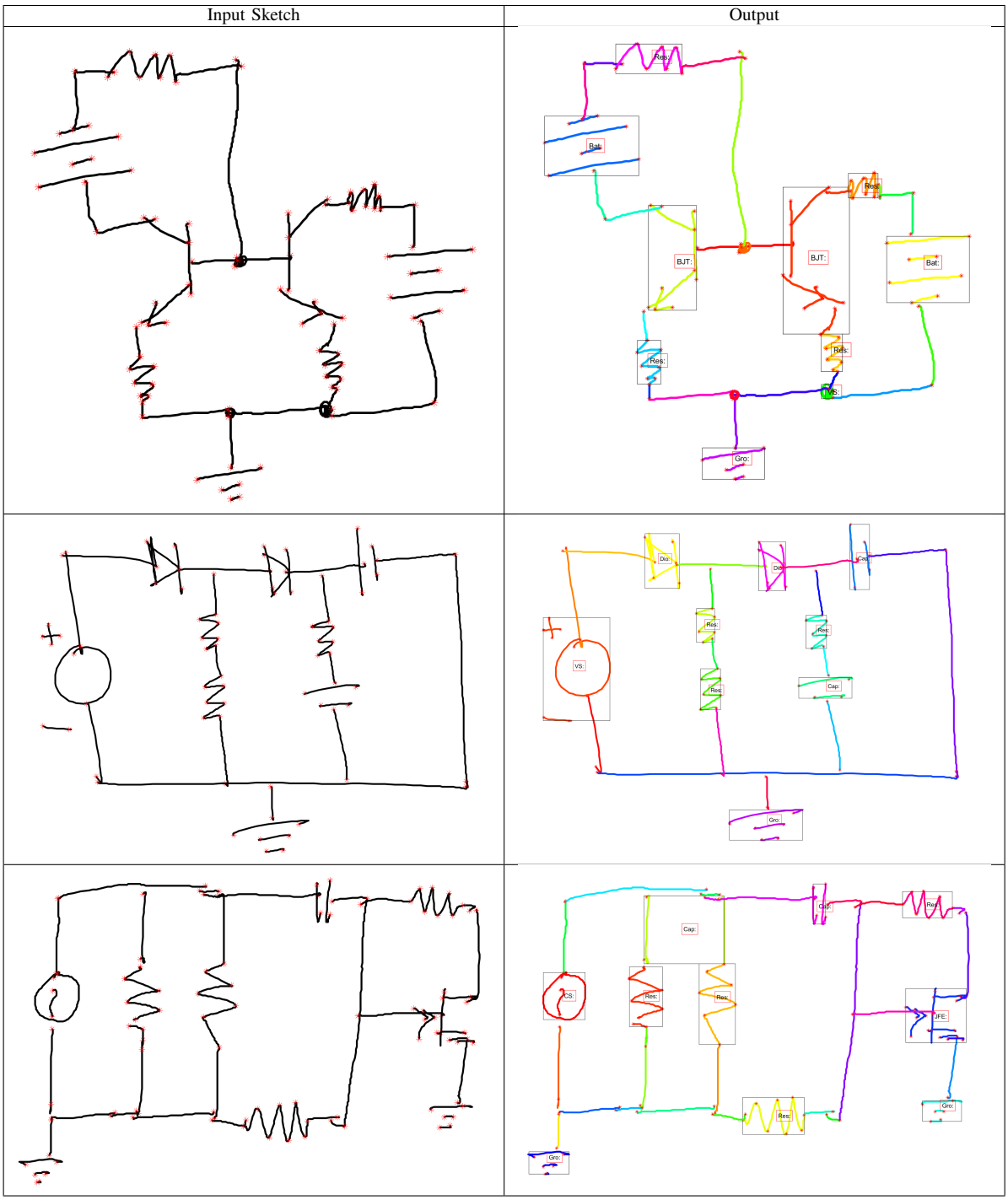


Table A.1: Sample recognition results of the MST method in EC dataset. The wires are not labeled to keep the figures simple. Start and end point of each primitive are labeled with a red star.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MCG.2021.3069863, IEEE Computer Graphics and Applications

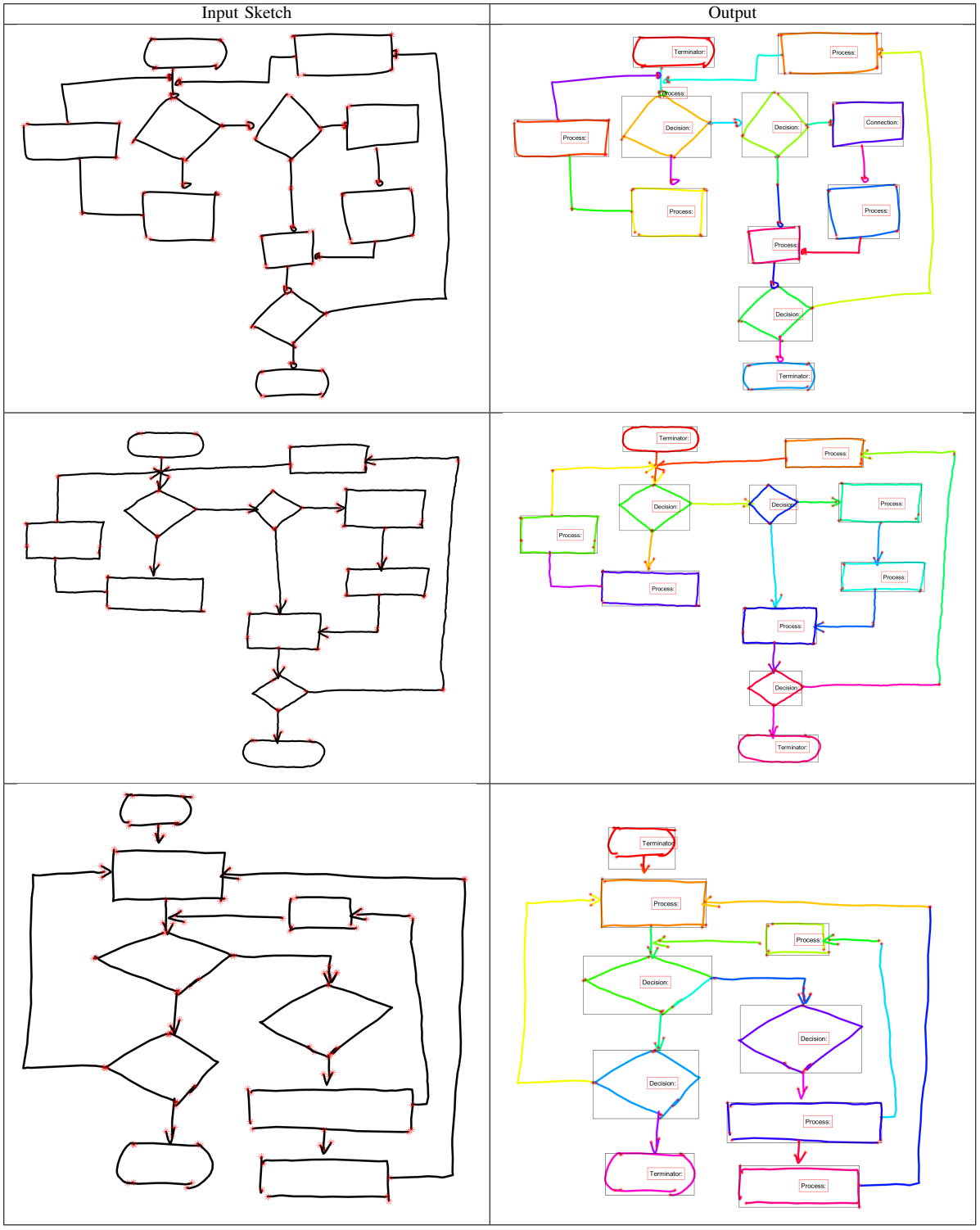


Table A.2: Sample recognition results of the MST method in FC dataset. The arrows are not labeled to keep the figures simple. Start and end point of each primitive are labeled with a red star.

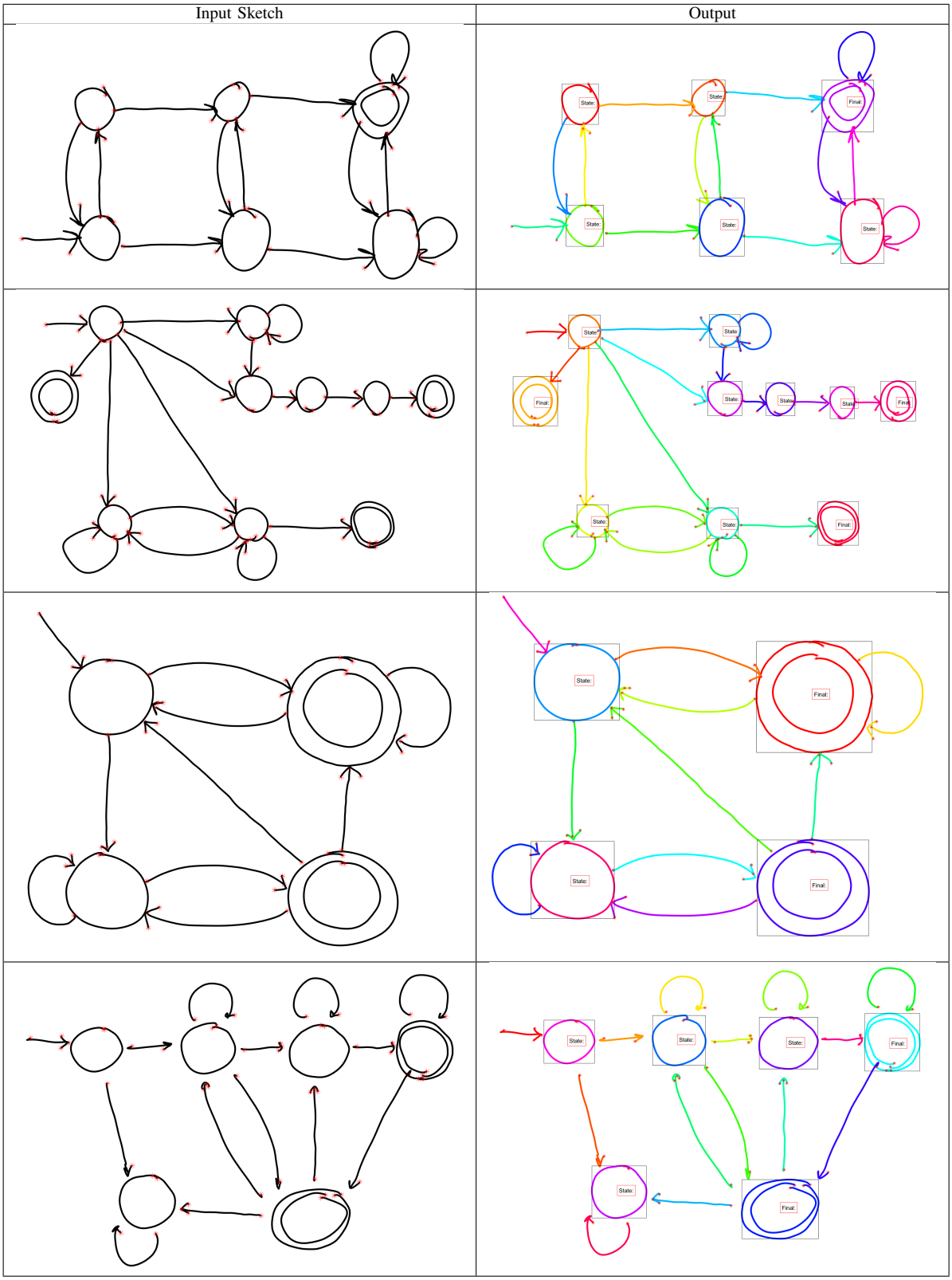


Table A.3: Sample recognition results of the MST method in FA dataset. The arrows are not labeled to keep the figures simple. Start and end point of each primitive are labeled with a red star.