Ph.D. Dissertation

by

R. Sinan Tümen

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Doctor of Philosophy

in

Computational Sciences and Engineering

Koç University

August 24, 2015

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a doctoral dissertation by

R. Sinan Tümen

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

Assoc. Prof. T.Metin Sezgin (Advisor)

Assoc. Prof. Nafiz Arca

Assoc. Prof. Engin Erzin

Assoc. Prof. Berrin Yanıkoğu

Assoc. Prof. Yücel Yemez

Date: _____

ABSTRACT

Hardware supporting pen-based interaction have been around for a long time, however progress in efficient and intelligent processing of input has been lagging far behind. This is partly due to the complicated nature of the sketch recognition problem. Optimal sketch recognition is intractable even for moderate-sized sketches. Recent methods deal with the problem either by making simplifying assumptions or by adopting sub-optimal methods.

In this thesis, as an alternative to the sub-optimal methods, we describe an optimal and polynomial-time trainable framework for multi-domain sketch recognition. Our solution handles offline and interspersed sketches as well as online sketches, and it does not make assumptions about user input. Our unified framework is based on supervised machine learning techniques, graph theory, and dynamic programming. We apply the framework to two fundamental problems of bottom-up sketch recognition: stroke fragmentation and sketch segmentation.

Dynamic programming approach is directly applicable to the fragmentation of strokes and segmentation of ordered primitives. For other cases, such as offline and interspersed sketches, we introduce the *spatial serialization* concept to impose an order on the primitives. We propose different graph theoretic methods and coherence models to convert 2D points into an ordered set of primitives. We evaluate the accuracy and runtime of different serialization schemes on multiple datasets. For both fragmentation and segmentation problems, experiments show that the accuracy of the unified framework either matches with the state-of-the-art, or it surpasses them by a large margin.

ÖZETÇE

Kalem tabanlı etkileşimi destekleyen donanımlar uzun zamandır mevcut olmasına rağmen, bu sistemlerin girdilerinin etkin ve sağlıklı bir şekilde işlenmesi nispeten geri kalmış bir konudur. Bu durum kısmen çizim tanıma probleminin üssel karmaşıklığından kaynaklanmakta olup, bahse konu karmaşıklık orta büyüklükteki bir çizim için dahi bilinen yöntemler ile en iyi çözümü bulmayı imkansız hale getirmektedir. Mevcut çizim tanıma sistemleri, karmaşıklık problemini aşmak için ya basitleştirici varsayımlar kullanmakta, ya da en iyi olmayan çözümleri kullanıcıya sunmaktadır.

Bu tezde, çizim tanıma problemi için, mevcut metodların aksine en iyi tanıma çözümünü çokterimli bir karmaşıklıkta bulabilen, eğitilebilir bir çerçeve yaklaşım sunulmaktadır. Sunulan yaklaşım, çevrimiçi çizimlerin yanısıra, çevrimdışı ve herhangi bir sıra izlemeyen çizimler için de kullanılabilmekte, kullanıcı girdisi hakkında herhangi bir varsayımda bulunmamaktadır. Önerilen yaklaşım, denetimli öğrenme teknikleri, çizge teorisi ve dinamik programlama metodlarına dayanmaktadır.

Çalışmamızda çerçeve yaklaşım, aşağıdan yukarıya çizim tanıma probleminin iki temel alt problemi olan vuruşların parçalanması ve çizim bölütlenmesi problemlerine başarı ile uygulanmıştır. Çevrimdışı ve belirli bir çizim sırası izlemeyen çizimler için ise, uzamsal sıralama metodu geliştirilmiş, uzamsal sıralama için birden çok çizge algoritması ve birden falza bağdaşım ölçüsü tanımlamıştır. Önerilen metodların doğruluğu ve çalışma süreleri birden çok veri setinde denenmiş, deneyler sonucunda, geliştirilen yaklaşımın, bazı veri setlerinde bilinen en iyi çizim tanıyıcılar ile başabaş sonuçlar verdiği, bazılarında ise farklı derecede iyi sonuçlar ürettiği tespit edilmiştir.

ACKNOWLEDGMENTS

Firstly, I would like to thank my advisor Dr. Tevfik Metin Sezgin for the continuous support and guidance during my Ph.D. study. It was a chance to study in his supervision, and it was an honor to be among his first Ph.D. students.

I also would like to thank my thesis committee, Dr. Nafiz Arıca and Dr. Yücel Yemez, for their insightful comments and encouragement. I am also thankful to Dr. Berrin Yanıkoğlu and Dr. Engin Erzin for agreeing to be on my thesis jury and their valuable suggestions and comments.

I am also grateful to the members of Intelligent User Interfaces group at Koç University: Ayşe Küçükyılmaz, Çağlar Tırkaz, Çağla Çığ, Yusuf Sahillioğlu, Atakan Arasan, Şerike Çakmak, Cansu Şen, Banuçiçek Gürcüoğlu, Kemal Tuğrul Yeşilbek, Özem Kalay, Özgür Oğuz, Mustafa Emre Acer, Emrah Şamdan, for their collaboration and friendship.

I also would like to thank my colleagues at Turkish Navy Reseach Center Command: Ramis Akın, İldeniz Duman, Ekrem Serin, Hakan Tezcan, Kadir Alpaslan Demir, Remzi Akdağ, Bahadır Kürşat Polat, Emre Karaman, for their encouragement and support.

Last but not least, I would like to thank my beloved wife Canan and my lovely daughter İdil for consistent motivation and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

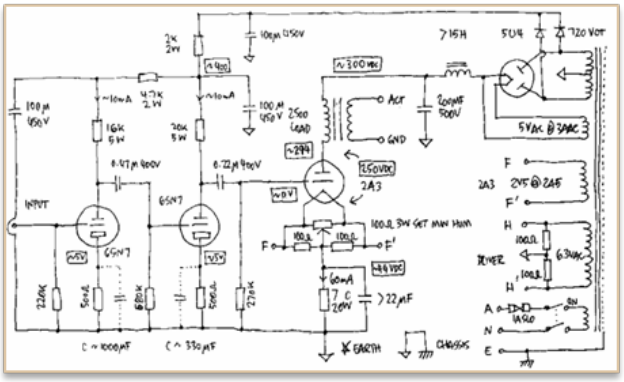
INTRODUCTION

Sketches are used naturally and ubiquitously in many different settings. Early design phases of engineering, planning of military operations and teaching in a classroom setting are typical scenarios for sketching. Figure-1.1 shows example sketches from several domains.

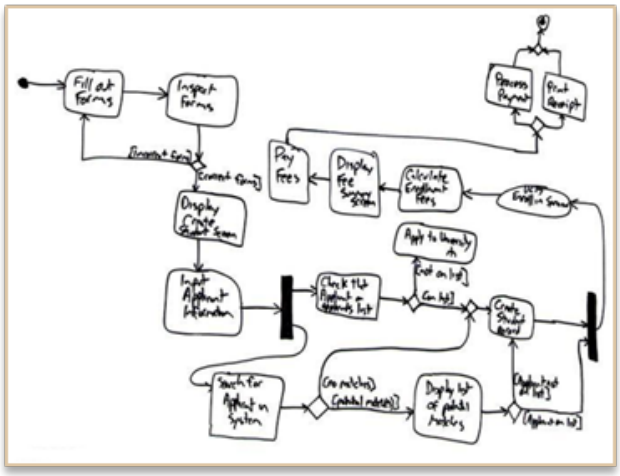
Sketches are used especially in working-memory extensive tasks. They provide a way to externalize 2D information to relieve the cognitive burden of holding and processing at the same time. In addition, they are also used to convey information to others. In human-to-human communication, sketches are common media for sharing information. In short, sketching is used mainly for two tasks: (1) Augmenting working memory, (2) Sharing ideas with others [Tversky, 2002].

In a common design scenario, we first sketch on a paper or on a board, then we translate it into computerized form using conventional menus and dialogs. The manual translation of sketches into computerized form is both redundant and time-consuming. Several studies attempt to automatically recognize the hand drawn sketches to make design process free from sketch translation. These efforts date back Sketchpad of Sutherland [Sutherland, 1964]. In recent years, sketch recognition regained the interest with the wide-spread availability of pen-based interfaces. However, sketches are still dead-pixels, due to lack of an accurate and efficient sketch recognition methods [Davis, 2007]. Development of a sketch recognition framework is a preliminary condition for the wide-spread usage of pen-based interfaces [Igarashi and Zeleznik, 2007].

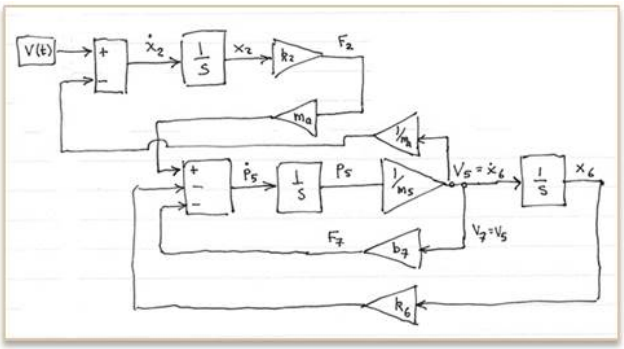
In this thesis, we focus on the development of a complete sketch recognition frame-



(a)



(b)



(c)

Figure 1.1: Example domains for sketch recognition: electrical circuit, UML and system block diagrams.

work. As an initial step, we first establish the terminology, and then describe the requirements for an ideal sketch recognition framework.

1.1 Terminology

Sketch is an informal freehand drawings consisting of a predefined set of symbols. The fixed vocabulary used in sketches allows us to represent the sketch by structural descriptions. A set of symbols in a domain is called as an *object*. For example, the resistor, capacitor and transistor are objects of electrical circuit domain.

Each object may in a sketch consists of multiple strokes. A *stroke* is set of points sampled between the pen-down and pen-up events. Objects that contains multiple strokes are called *multi-stroke objects*. Strokes that span multiple objects are called *multi-object strokes*. An object is interspersed, if its primitives are interspersed in the temporal order. In other words, the user starts to draw a new object, before finishing the current one.

Bottom-up approaches first splits the strokes into primitives.*Primitives* are simple geometric shapes such as lines and arcs. Primitives do not contain multiple objects and they are considered as the atomic entities that form the domain objects. *Fragmentation* of strokes into primitives results in a more expressive and manageable collection of points. Two primitives are siblings if they are parts of the same object.

The sketch recognition is the segmentation and the classification of a sketch. Segmentation is the task of grouping the primitives to form the domain objects. Classification is the task of determining which object each group of primitives represents. Segmentation produces k groups $G = g_1, g_2, ..., g_k$. Classification gives us the labels for each group $L = l_1, l_2, ..., l_k$.

1.2 Requirements for Sketch Recognition

The ultimate sketch recognition framework must meet certain criteria on accuracy, performance and adaptation.

- *Accuracy:* High accuracy is a fundamental requirement for a sketch recognition system. We measure the system's accuracy by comparing the recognized objects with the ground truth, which is generated by human annotators.

- *Runtime Efficiency:* A sketch recognition system must find an optimal solution in an acceptable time. A naïve brute force search for an optimal solution is impractical, since the number of recognition solutions increases exponentially with sketch size.

- *Adaptation to Different Domains:* A sketch recognition system must adapt to different sketch domains such as electrical circuits, mechanical diagrams, chemical structures and UML diagrams etc.

- *Adaptation to Sketching Styles:* The drawing styles vary across users. For instance, the level of messiness and noise are subject to change. A sketch recognition system must adapt to the styles of new users easily.

- *Freedom on Sketching Style:* Sketch recognition system must not constrain the user on his/her drawing style. Recognizer must adapt to the style of the user, not vice versa.

- *Offline/Online Mode:* Offline sketches do not have temporal information associated with ink points. The ultimate recognizer must also work in offline mode.

1.3 Overview of the Approach

To meet the requirements for an ideal sketch recognition system, we use two key methodologies throughout our framework: (1) Supervised learning for adaptability, and (2) Dynamic programming for efficiency. Machine learning methods automatically learns models that would require a tedious parameter tuning task if done manually. Dynamic programming approach reduces the complexity of search for optimal

solution to a polynomial time. We apply our approach to two fundamental problems of sketch recognition.

- *Fragmentation*: Fragmentation splits the strokes into manageable parts such as lines and arcs. A DP based solution is inherently applicable to the fragmentation problem.

- *Segmentation*: Segmentation groups the primitives into domain objects. If user draws objects sequentially, then DP based approach guarantees to find the optimal solution. Otherwise, we must transform the problem into a DP applicable form.

Fragmentation and segmentation are similar. In fragmentation, we first split the strokes into segments and combine them into primitives. In segmentation, we further combine the fragmented primitives into domain objects. We use a dynamic programming approach for both stroke fragmentation and sketch recognition problems.

1.3.1 Dynamic Programming

The dynamic programming approach is similar to *divide-and-conquer* approach, but the latter method is applicable, when subproblems do not overlap. The problems with overlapping subproblems and additive cost functions conforms the optimal substructure property. The dynamic programming approach is applicable, when the problem exhibits an optimal substructure.

The primitive fragmentation problem intrinsically exhibits optimal substructure property. The segmentation problem with sequential drawing assumption also conforms the optimal substructure. However, offline and interspersed sketches must be transformed to a DP applicable form. We introduce and apply spatial serialization for transformation.

1.3.2 Spatial Serialization

Spatial serialization is the task of generating a synthetic ordering on the primitives that is independent of temporal information. In generating the primitive order, it uses only spatial properties of the primitives. In a DP applicable serialization, the primitives of each object is connected to each other. After a correct serialization is generated, the DP based approaches become applicable for the optimal recognition.

Spatial serialization uses pairwise coherence estimations and well-known graph algorithms to create a correct serialization. In the coherence estimations, we use the PairIDM and supervised machine learning methods. After we create a serialization, we generate many object hypotheses, and evaluate each hypothesis with supervised learning methods.

1.3.3 Hypotheses Evaluation

The dynamic programming based approach generates many primitive/object hypotheses. We can evaluate the quality of each hypothesis using either a rule-based [Hammond and Davis, 2005] or probabilistic [Alvarado and Davis, 2006] approach. The rule-based system requires a tedious task of rule generation and parameter tuning. As an alternative, a probabilistic approach calculates a matching score between an object template and a hypothesis using machine learning techniques.

We can calculate a probabilistic matching score for an object hypothesis based on the properties of its primitives and their relations, or we can treat the hypothesis as a raster image and estimate a matching score based on the features extracted from the image patch. The fixed-length features makes image-based method well suited for the standard three-step machine learning pipeline, where first feature vectors are extracted from a set of example images, then classifiers are trained using these features, and finally new sketches are classified using the trained models.

Throughout the thesis, we follow the probabilistic approach in the matching score calculation. In the final step of fragmentation and recognition, we combine the costs of object/primitive hypotheses to find the optimal solution.

1.4 Thesis Roadmap

The next chapter formulates the fragmentation problem, describes our implementation and gives analysis for the fragmentation model. Chapter three introduces segmentation problem and spatial serialization concept, generalizes the formulation for fragmentation and gives several insights and analyses for the segmentation model. Chapter four gives a review of the stroke fragmentation and sketch recognition literature. Chapters five summarizes our main contributions. We conclude with future work.

Chapter 2

LITERATURE REVIEW

The methods in the literature consider the stroke fragmentation and sketch recognition as separate problems. In contrast, we present a unified approach to both problems.

2.1 Stroke Fragmentation

Many methods have been suggested for stroke fragmentation. In general, these methods can be split into two categories. The first group of methods find fragmentations by detecting corners, others compute fragmentations by computing piecewise approximations of given digital curves. In both cases, existing methods employ sets of carefully designed heuristics to compute fragmentations.

2.1.1 Corner Detection Algorithms

Corner detection algorithms mainly rely on the maxima in curvature and the minima in speed to detect corners. However, speed and curvature data contain substantial amounts of noise due to digitization and imperfect motor control of the hand, which must be filtered out before detecting corners. To address this issue, many authors have attempted to determine optimal smoothing parameters, and then locate corners. To distinguish small scale curvature variations that are due to noise from the larger variations due to real corners, earlier work on corner detection employed multiscale approaches.

Recent corner detection algorithms still employ different forms of speed and curvature metrics to locate corners [Wolin and Hammond, 2008, Xiong and Jr., 2010]. For example, the Shortstraw algorithm [Wolin and Hammond, 2008] declares points

with straw values less than some threshold as corners, where straw for a point is defined to be the euclidean distance between its right and left neighbors. Istraw [Xiong and Jr., 2010], an extension to Shortstraw, also uses straw values, but it also takes advantage of speed features.

Istraw and Shortstraw are rule based approaches that rely on local stroke features. Their parameters have to be manually tuned in order to adapt them to different users, contexts and primitive types. This requirement is their biggest disadvantage, because although they can achieve high accuracy rates in datasets for which their parameters have been tuned for, they perform poorly on other databases unless they go through a tedious iterative manual tuning cycle. An exception is Classyseg [Herold and Stahovich, 2011a], which utilizes machine learning. However, Classyseg relies on local decisions to build a corner set, and does not address the issue of finding a globally optimal solution.

2.1.2 Piecewise Approximation Algorithms

Piecewise approximation algorithms generally identify an initial set of segments, then they split or merge them based on a set of heuristics. For instance, Yu et al.[Yu, 2003] use mean shift clustering for noise reduction, then they approximate the stroke with arcs, lines and ellipses using visual features. The Sort-Merge-Repeat (SMR)[Wolin et al., 2009] algorithm merges the initial segments based on the fit errors.

There are also piecewise fragmentation methods using dynamic programming. For example, Deufemia et al.[Deufemia and Risi, 2005] use dynamic programming to match a given stroke to templates. Liu et al. [Yin et al., 2005] use dynamic programming with a deterministic cost function. Kolesnikov [Kolesnikov, 2012] uses dynamic programming within a minimum description length (MDL) setup. These methods are either limited to predefined templates or they use deterministic cost functions, which do not meet the adaptability requirements of the fragmentation problem. Unlike these methods, we learn primitive level models from data.

Speedseg is another recent piecewise approximation algorithm. After identifying a set of initial segments, it merges and splits the segments using a set of improved heuristics, however it relies on preset parameters [Herold and Stahovich, 2011b]. By contrast, our method finds piecewise approximations via dynamic programming combined with an adaptive cost function, rather than relying on preset thresholds or templates.

2.2 Sketch Segmentation

We can categorize sketch recognition methods by their level of abstraction: (1) Primitive-level, (2) Object-level. Primitive-level methods explore the properties of primitives and their relations to match them to the object templates. Object-level methods generate sets of object hypotheses, assess their quality and combine them into a coherent solution. Generally, sketch recognizers are not purely primitive or object level, but they are generally a combination of both. However, in essence, they are either primitive-level or object-level. According to this categorization, we can classify our work as an object-level method.

2.2.1 Primitive-Level Methods

Primitive-level methods classify primitives and their relations to form the domain objects. For example, Hammond's top-down approach [Hammond and Davis, 2006] detects the domain objects using object templates. A template is composed of rules, which are either the properties of a primitive or relations between primitives. In this approach, a rule-based system determines whether the given primitives satisfy a certain rule. After all rules of an object template are satisfied, the corresponding domain object is detected, and its primitives are removed from universal set. The weaknesses of this method are its strong dependence on the correct primitive fragmentation, and its requirement for a tedious parameter tuning. Hammond's method implicitly uses stroke ordering in detection and it constrains the object size.

Rule-based systems with hard constraints do not perform well, because sketches

are generally so noisy that they cannot be described with hard-constraints. For this reason, Alvarado's method [Alvarado and Davis, 2006] uses shape grammars, but instead of using hard constraints, it constructs a probabilistic knowledge base and assesses the fulfilment of a template using Bayes nets. It uses stroke ordering to build the hypotheses incrementally. We can classify Alvarado's method as a greedy method based on best match. It does not guarantee a globally optimal and coherent solution for the segmentation problem.

Incremental nature of sketching makes temporal models applicable for sketch recognition. Sezgin et al [Sezgin and Davis, 2008] models the temporal patterns between primitives using Dynamic Bayesian Network. DBN is a generalization of Hidden Markov Models with additional random variables. In this method, additional random variables determine the object boundaries, type of the objects and the interspersing. Sezgin's method uses probabilistic temporal templates, which is learned from the training data.

Some methods in the literature use the learned distance metrics/labels between strokes for clustering. For example, Stahovich et al [Stahovich et al., 2014] first classify each stroke and stroke pair based on object types and stroke relations. Then, they form the object clusters by chaining the strokes according to the stroke relations. Delaye et al[Delaye and Lee, 2015] propose another clustering approach. They first compute a distance metric between pairs of primitives, and then they perform a Single Linkage Agglomerative Clustering (SLAC) and apply a threshold on the linkages to form the domain objects. They train both distance metric and threshold using the training data. Both methods make use of temporal information.

2.2.2 Object-Level Methods

Object-level methods generate a set of object hypotheses, and combine them into a coherent solution. For instance, Shilman et al [Shilman and Viola, 2004] formulate the sketch recognition as an optimization problem, and perform a spatially bounded search to detect objects in a proximity graph. They formulate the combined seg-

mentation cost as an additive function, and use a hashing mechanism to re-use the calculated symbol costs. The method limits the number of primitives and the distances between them. The approach is purely spatial, and it does not use stroke ordering.

Some object-level methods apply dynamic programming to determine the optimal segmentation. They also formulate the overall cost function as a combined cost of individual objects. For example, Arandjelovic et al [Arandjelović and Sezgin, 2011] assume that the user draws the objects sequentially and apply 1D dynamic programming for segmentation. Feng et al [Feng et al., 2009] relax the constraint for sequential drawing and they apply 2D dynamic programming for segmentation. However, the degree of interspersing is limited in this approach. The user is required to redraw the sketch, if sibling nodes have more than two non-sibling nodes in the given sequence of primitives.

Some other methods apply structured prediction methods to sketch recognition. For example, Ouyang's method [Ouyang and Davis, 2011] first creates a Conditional Random Field (CRF) over primitives and object candidates. It generates the object candidates using a spatially and temporarily bounded search around the given primitive. It then performs approximate Loopy Belief Propagation on CRF. Finally, it combines the object hypotheses into a coherent solution using the agglomerative clustering approach. It uses stroke ordering, and puts constraints on the number of strokes per object and the maximum size of the objects.

Chapter 3

STROKE FRAGMENTATION

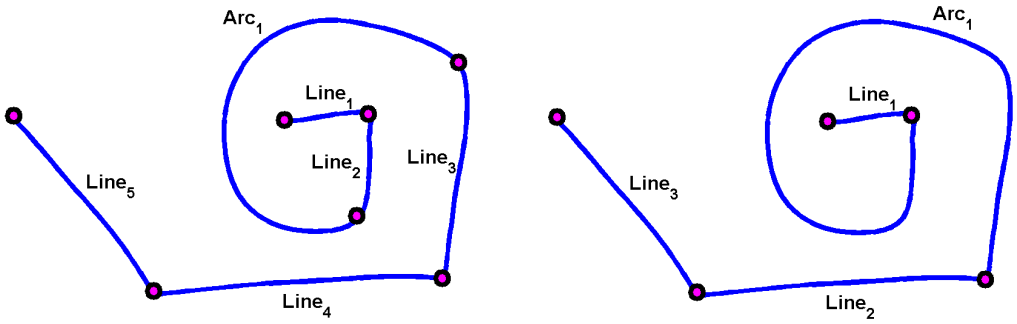
Fragmentation splits the strokes into a set of geometric primitives such as lines and arcs. It is a necessary step in sketch recognition, because, several objects can exist in a single stroke. If we combine strokes instead of primitives to form the domain objects, then the solution may result in an under-segmentation.

A primitive must be atomic, i.e., it cannot be further split. In our case, this means that it contains only points of one object. In addition, the fragmentation must produce the smallest number of primitives possible, because, the complexity of segmentation increases exponentially with the number of primitives. Generally, a single line or an arc does not span multiple objects. As a result, to guarantee the atomicity and constraint on the number of primitives, we can define line and arc as the primitive set. However, primitives may vary across domains.

Besides sketch recognition, many computer graphics applications require approximation of digital curves using sets of prespecified geometric primitives. For example, computer aided design and free-hand modeling applications require fitting NURBS to digital curves ([Adi et al., 2010], [Orbay and Kara, 2011]). In all these applications, the goal is to find the best way of representing a digital curve using geometric primitives that collectively produce an accurate approximation.

A fragmentation system must meet the general accuracy, efficiency and adaptability requirements of a sketch recognition system. In addition, it must adapt to different fragmentation preferences.What constitutes a correct fragmentation for a given stroke can be highly subjective. For example, two users may potentially prefer different fragmentations for the same stoke (e.g. Figure-3). Accordingly, a fragmentation system should be able to adapt to different preferences. In addition, the set of primitives

allowed for building fragmentations may vary across domains and applications. For example, in a scenario that involves hand-drawn circuit diagram recognition, a set of primitives that includes helices may be of use in order to aid the recognition of inductor symbols. Therefore, the set of primitives used in generating fragmentations should be easy to customize.



(a) (b)

Figure 3.1: An example stroke with more than one valid fragmentations.

Unfortunately, none of the existing fragmentation techniques meets all of the requirements listed above. In this section, we describe DPFrag, a stroke fragmentation algorithm which is designed to meet all five of these requirements.

Our algorithm has two main components: (1) A dynamic programming method, which reduces the exponential runtime complexity of naïve search to $O(n^2)$, in terms of stroke length; (2) An adaptive cost function, which enables DPFrag to adapt to different sets of primitives, contexts and user styles. Our evaluation with five different datasets shows that our method meets the stated requirements, and outperforms the state of the art in terms of accuracy.

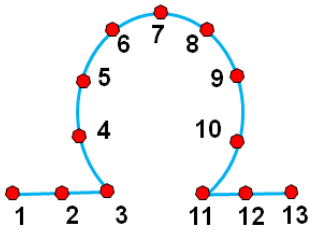
3.1 Problem Formulation

Primitives are simple geometric shapes such as lines and arcs. Primitives do not span multiple objects and they are considered as the atomic entities to form the domain

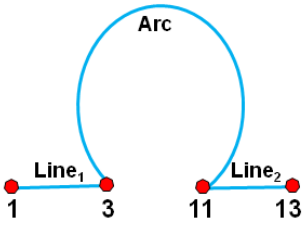
objects. *Fragmentation* of strokes into primitives results in a more expressive and manageable collection of points.

3.1.1 Optimal Fragmentation

A fragmentation F of a stroke is an ordered partition of the stroke points into nonempty sets. A fragmentation can be expressed in terms of a set of corner points $C = \{c_1, c_2, ..., c_n\}$. We denote a fragmentation as $F = \{P_{c_1:c_2}, P_{c_2:c_3}, ..., P_{c_{(n-1)}:c_n}\}$. We also index the primitives using subscripts such as P_i. In the context of Figure-3.2, a plausible fragmentation is $F = \{P_1, P_2, P_3\} = \{P_{1:3}, P_{3:11}, P_{11:13}\}$.



(a)



(b)

Figure 3.2: A fragmentation example: (a) shows the raw stroke points (b) shows the correct fragmentation $F = \{P_{1:3}, P_{3:11}, P_{11:13}\}$ for the stroke. Three primitives – two lines and an arc – are generated as a result of the fragmentation.

Given a fragmentation F, the fragmentation optimality $\mathcal{O}$ can be assessed by checking how well its subsets look like individual primitives:

$$P(\mathcal{O}|F) = \prod_{P_i \in F} P(O|P_i) \tag{3.1}$$

where $\mathcal{O}$ denotes the fragmentation optimality, and O denotes the event of being part of the optimal fragmentation. Then, the optimality criterion for the fragmentation problem can be written as:

$$\begin{aligned} F_{opt} &= \arg\max_{F \in \Gamma} \; P(\mathcal{O}|F) \\ &= \arg\max_{F \in \Gamma} \; \prod_{P_i \in F} P(O|P_i) \end{aligned} \tag{3.2}$$

where Γ represents the set of all possible fragmentations. To avoid the numerical underflow of probabilities, we replace the probability term with log probability:

$$F_{opt} = \arg\max_{F \in \Gamma} \; \sum_{P_i \in F} log \; P(O|P_i) \tag{3.3}$$

Then, a cost function $Cost(F)$, which is minimized for optimal solutions can be defined as:

$$Cost(F) = -\sum_{P_i \in F} log \; P(O|P_i) \tag{3.4}$$

The optimality criterion can be restated in terms of our cost function $Cost(F)$:

$$\begin{aligned} F_{opt} &= \arg\min_{F \in \Gamma} \; Cost(F) \\ &= \arg\min_{F \in \Gamma} \; -\sum_{P_i \in F} log \; P(O|P_i) \end{aligned} \tag{3.5}$$

The calculation of the term $P(O|P_i)$, which is the likelihood of a segment P_i being a part of the optimal segmentation, will be explained next.

3.2 DPFrag Implementation

Our fragmentation framework consists of two modules: The first module, **classifier learning**, allows us to learn optimal fragmentation settings and produces the

Figure 3.3: The schematic view of the classifier learning module. DPFrag classifier, which is used to estimate $P(O|P_i)$, is produced by the classifier learning module.

DPFrag classifier, which is used to estimate $P(O|P_i)$. The second module, **stroke fragmentation**, takes the DPFrag classifier and computes optimal fragmentations of new strokes using dynamic programming. We now describe the details of the classifier learning and stroke fragmentation modules.

3.2.1 Classifier Learning

A given stroke segment can be interpreted in three ways:

C1 : The segment forms a single primitive, no more, no less.

C2 : The segment covers a part of a primitive.

C3 : The segment contains multiple primitives.

We refer to these cases as primitive, subprimitive and multiprimitive cases. We train a multiclass classifier for estimating which one of these three cases matches a given stroke segment. This classifier, which is called the DPFrag classifier in the rest of this document, allows us to estimate the term $P(O|P_i)$ in Equation-3.5. The classifier training module consists of several stages, which are illustrated in Figure-3.3.

Segment Extraction and Training Set Generation

To train the DPFrag classifier, we need examples of primitive (C1), subprimitive (C2) and multiprimitive (C3) classes. Consecutive sequences of segments in an annotated stroke provide us with instances of these three classes as illustrated in Figure-3.4. To extract segments, we first use the well-known Douglas-Peucker algorithm to partition a given stroke. Then, we feed all connected subsets of the stroke segments into the feature extraction and label assignment steps.

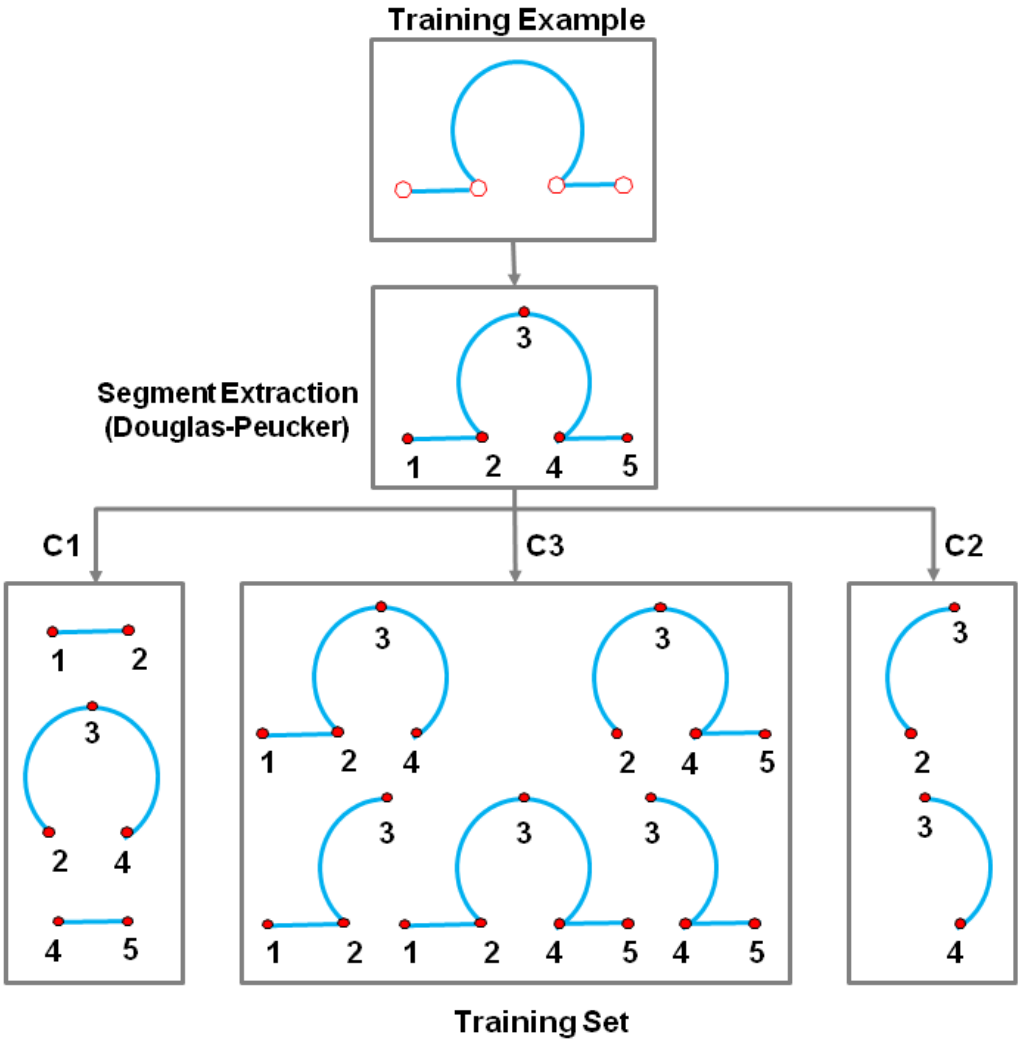


Figure 3.4: The demonstration of training set generation. Given a stroke annotated by a human annotator, we generate the examples of C1, C2 and C3 using the consecutive segments of the stroke. Human-annotated corners for the training example are marked with circles, and segments extracted by the Douglas-Peucker algorithm are marked with filled dots.

The Douglas-Peucker algorithm recursively divides a curve into short linear segments until each point on the stroke falls within at least ϵ distance away from the line connecting its left and right segment points. We automatically set ϵ such that no segment in the training set contains multiprimitives. This is a very conservative choice of threshold, hence the probability of missing corners in the test data due to a large ϵ is negligible.

Feature Extraction

The DPFrag classifier deals with primitives, hence we use features that encode the shape, curvature and speed properties of primitives. Some of the features that we use are based on prior work on fragmentation, while others have been designed by us.

Feature	Description
1, 2	$\Delta_{avg}(P_{c_i:c_j})$ and $\Delta_{max}(P_{c_i:c_j})$ for a line fit.
3	$\min(\Delta_{avg}(P_{c_{i-}:c_j}), \Delta_{avg}(P_{c_i:c_{j+}}))$ for a line fit.
4	$\min(\Delta_{max}(P_{c_{i-}:c_j}), \Delta_{max}(P_{c_i:c_{j+}}))$ for a line fit.
5, 6	$\Delta_{avg}(P_{c_i:c_j})$ and $\Delta_{max}(P_{c_i:c_j})$ for an ellipse fit.
7	$\min(\Delta_{avg}(P_{c_{i-}:c_j}), \Delta_{avg}(P_{c_i:c_{j+}}))$ for an ellipse fit.
8	$\min(\Delta_{max}(P_{c_{i-}:c_j}), \Delta_{max}(P_{c_i:c_{j+}}))$ for an ellipse fit.
9	Chord length.
10	Euclidean distance between start and end points.
11	Width of the bounding box.
12	Height of the bounding box.
13	The number of segments

Table 3.1: Shape-based features for a segment from c_i to c_j. The shorthands $\Delta_{max}(P_{c_i:c_j})$ and $\Delta_{avg}(P_{c_i:c_j})$ represent the maximum and average least square fit errors for the stroke segment from c_i to c_j. Points c_{i+}, and c_{i-} refer to the closest points to c_i on either side as found by the Douglas-Peucker algorithm.

Shape-based Features Visual appearance is the most distinctive property of primitives, hence, we mainly use fit-based features to characterize the similarity between the hand-drawn ink and a set of idealized geometric templates such as lines and elliptical arcs. Measuring the similarity to these templates enables us to discriminate the primitive, subprimitive and multiprimitive cases as follows:

C1: If the stroke segment represents a primitive, then the fit-based error for a specific template is expected to be lower than the multiprimitive case. The concatenation of a new segment to either end point of the sequence increases the fit-based error significantly.

C2: If the segment is a subprimitive, then the fit-based error for a specific template is expected to be lower than the multiprimitive case. Furthermore, extending the stroke segment on either end should not to increase the fit error substantially.

C3: If the segment consists of multiple primitives, we would expect it to have a large fit error for all primitive templates.

Table-3.1 lists a complete description of our shape-based features designed based on the properties of the three classes as stated above.

Curvature and Speed based Features The utility of curvature and speed based features has been cited extensively in the previous work on stroke fragmentation. The features that we use are listed in Table-3.2. The features were designed to aid the classification of stroke segments into C1, C2 and C3 by noting the following about the curvature and speed characteristics of segments:

C1: If the stroke segment constitutes a primitive, the segment is likely to have corner characteristics (maxima in curvature and minima in speed) at its end points.

C2: If the segment is a subprimitive, only one of the end points carries corner characteristics.

C3: If the segment contains more than one primitive, the segment is likely to have additional points with corner characteristics in addition to the end points.

Feature	Description
14	Maximum absolute curvature at the start of the segment.
15	Minimum speed at the start of the segment.
16	Minimum straw value at the start of the segment.
17	Maximum absolute curvature at the mid-points of the segment.
18	Minimum speed at the mid-points of the segment.
19	Minimum straw value for the mid-points of the segment.
20	Maximum absolute curvature at the end of the segment.
21	Minimum speed at the end of the segment.
22	Minimum straw value at the end of the segment.

Table 3.2: Speed and curvature features used in DPFrag.

To estimate the curvature at a given point, we use the curvature formula from analytical geometry:

$$\text{Curvature}_\text{i} = \frac{\dot{x_i}\ddot{y_i} - \ddot{x_i}\dot{y_i}}{(\dot{x_i}^2 + \dot{y_i}^2)^{3/2}} \tag{3.6}$$

where (x_i, y_i) is the coordinates of the i$^{\text{th}}$ point. The speed at a given point is estimated using finite differences:

$$\text{Speed}_\text{i} = \frac{d_{i+1} - d_{i-1}}{t_{i+1} - t_{i-1}} \tag{3.7}$$

where d_i is the chord length between the first and the i$^{\text{th}}$ point of the stroke, and t_i is the time-stamp of the i$^{\text{th}}$ point.

To get a better estimate of the curvature and speed at the ends of the segment, we pick points within $\epsilon/2$ radius of the segment end points as illustrated in Figure-3.5. We compute the curvature maxima and speed minima within these regions. A radius

Figure 3.5: To get reliable curvature and speed features, we compute the curvature extrema separately in three regions: two circular regions with radius of $\epsilon/2$ around start and end points of the stroke segment and a third region between these two circular regions.

of $\epsilon/2$ guarantees that the support regions for computing the curvature and speed features remain disjoint.

Label Assignment

After features are extracted for all contiguous stroke segments, they are labeled as primitives (C1), subprimitives (C2), or multiprimitives (C3) automatically based on manual fragmentations of a human annotator as illustrated in Figure-3.4. If the annotation of a given stroke segment contains corners only at both ends, then it is an instance of primitive (C1). If it contains only one corner at an end point, it is a subprimitive (C2). Otherwise it is an instance of the multiprimitive (C3).

Training of the DPFrag Classifier

Using examples of the three classes (C1, C2, C3), we train a probabilistic multiclass support vector machine (SVM) with a radial basis function kernel. The output of the SVM classifier is posterior probability estimate of the class labels. We use the LIBSVM toolbox to learn pairwise probability estimates by maximizing a likelihood function using the training data and predicted labels. Following the optimization, LIBSVM computes multiclass probabilities by combining pairwise class probabilities. We train the classifier via 5-fold cross validation, and use it to estimate $P(O|P_i)$ for

a given stroke segment.

3.2.2 Fragmentation of Stroke Instances

After the DPFrag classifier is trained, it can be used to estimate the cost of any given fragmentation. One can attempt to enumerate all possible fragmentations Γ of a given stroke, and declare the fragmentation with the smallest cost as the optimal solution. However, because the number of fragmentations grows exponentially, exhaustive enumeration is not feasible. Fortunately, the optimal substructure property of dynamic programming holds for our problem and it allows us to use dynamic programming to search for optimal fragmentations efficiently.

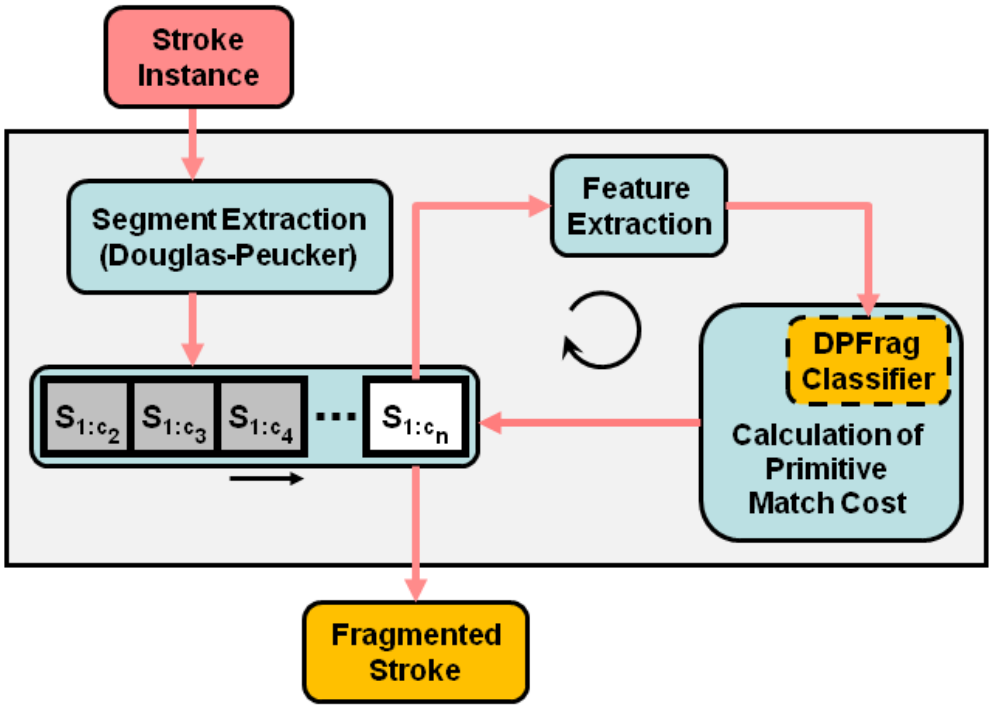


Figure 3.6: Schematic view of the stroke fragmentation module. The primitive matching costs are calculated and combined efficiently by the fragmentation module.

A problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to subproblems. As a proof of optimal substructure property for the fragmentation problem, suppose that an optimal fragmentation

for a stroke segment splits the segment at point j. Then, the fragmentation of $P_{1:j}$ within this optimal fragmentation must also be an optimal fragmentation. Because, otherwise, a less costly fragmentation of $P_{1:j}$ could have been substituted in, to obtain a better segmentation, which is a contradiction. This shows that any optimal solution contains within it optimal solutions to subproblems, and the optimal substructure property holds for the fragmentation problem with the optimality criterion in Equation-3.5. Thus, we can build an optimal solution by solving subproblems using dynamic programming.

Next, we define the cost of an optimal solution recursively in terms of optimal solutions to the subproblems. If there is no segment, which is the trivial case, the cost is zero. Assuming that fragmentation of $P_{1:k}$ splits the segment at the point j where $j < k$, the cost of optimal fragmentation is :

$$Cost[k] = Cost[j] - log\ P(\ O|P_{j:k}\ ) \tag{3.8}$$

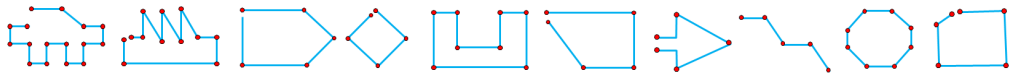
Thus, our recursive definition for the minimum cost is :

$$Cost[k] = \begin{cases} 0 & \text{if } k = 0 \\ \underset{1 \le j < k}{min}(Cost[j] - log\ P(\ O|P_{j:k}\ )) & \text{if } k > 0 \end{cases} \tag{3.9}$$

Instead of computing the solution by recurrence, we can iteratively compute the optimal cost using a bottom-up approach. This allows us to solve the fragmentation problem in $O(n^2)$, which is far less than the exponential complexity of naïve search methods. The stroke fragmentation module, and illustration of how it is connected to the DPFrag classifier is shown in Figure-3.6.

3.3 Evaluation

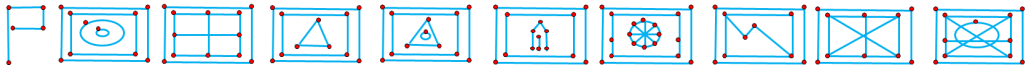
We have conducted several experiments to evaluate the accuracy, performance and adaptability capabilities of DPFrag. We evaluated our algorithm on datasets with diverse properties, including datasets containing geometric shapes that are not associated with a specific domain (*out-of-context datasets*), and datasets containing sketches of symbols with an associated meaning in a real domain (*in-context datasets*).



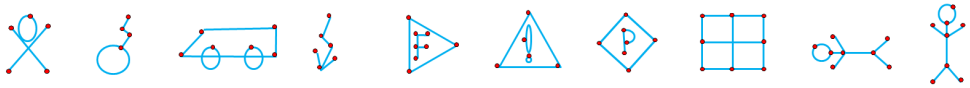
(a) Shortstraw

(b) IStraw

(c) Speedseg



(d) COAD



(e) NicIcon

Figure 3.7: Vector drawings of the shapes from out-of-context and in-context datasets.

3.3.1 Out-of-Context Datasets

The standard out-of-context datasets that we used for evaluation are the ShortStraw, Istraw, and Speedseg datasets. Each dataset contains symbols designed specifically to evaluate its corresponding fragmentation algorithm with the same name (Short-Straw, Istraw, Speedseg). The Shortstraw dataset is composed of polyline strokes. Istraw and Speedseg shapes contain line and arc primitives. Representative symbols from the three datasets are shown Figure-3.7. For detailed information on the datasets, we refer the reader to their respective papers [Wolin and Hammond, 2008, Xiong and Jr., 2010, Herold and Stahovich, 2011b]. Evaluation with these standard out-of-context datasets allow us compare the performance of DPFrag with the state of the art.

3.3.2 In-Context Datasets

The in-context datasets used in our evaluation include the well-known Course of Action Diagrams (COAD) and NicIcon datasets. Symbols in these datasets were not collected for evaluating specific fragmentation algorithms, so they allow us to test our algorithm with more realistic inputs. This is an important point to make, because the accuracy and performance on in-context datasets have not been reported in earlier work on stroke fragmentation. However, accuracy for in-context datasets is what really matters, because they are more representative of how strokes look like when people draw symbols from real domains, rather than focusing on reproducing simple strokes with very specific constructions.

The COAD dataset consists of symbols that are drawn on maps while planning military operations. A complete list of COAD symbols can be found in the US Army Field Manual 101-5-1. The NicIcon dataset contains symbols designed for emergency management applications. Representative symbols from both datasets are shown in Figure-3.7. Both datasets can be fragmented using line and elliptical arc primitives. Since these datasets contain excessive number of examples, we annotated and used random subsets of these datasets in our evaluation. Table-3.3 lists the number of manually annotated strokes in each dataset.

	Strokes	Symbols	Types	Subjects	In-Context
ShortStraw	600	600	11	15	No
Istraw	400	400	10	10	No
COAD	1507	400	20	8	Yes
NicIcon	1204	400	14	32	Yes
Speedseg	2311	2311	10	14	No

Table 3.3: The properties of datasets used in the evaluation of DPFrag.

3.3.3 Fragmentation Accuracy

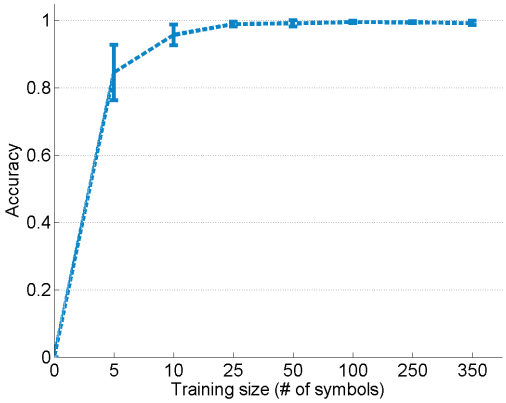
Fragmentation accuracy is the most important evaluation criterion used to measure the utility of a fragmentation algorithm. To measure the accuracy of DPFrag for each dataset, we use a randomly selected subset of the dataset for training DPFrag, and the rest for testing its accuracy. Figure-3.8 shows the average accuracy of DPFrag for different training data sizes over 10 randomized runs. The error bars represent the range of one standard deviation for recorded accuracies of 10 repeated measurements. As seen in the figure, DPFrag reaches excellent accuracies with as few as 25-50 annotated examples in all datasets.

Table-3.4 compares DPFrag with the state of the art methods in the literature. Reported DPFrag accuracies are the average accuracies of 10-time repeated trials, in which a set of randomly selected examples are used for the training the DPFrag classifier and accuracies are obtained on the rest of a dataset.

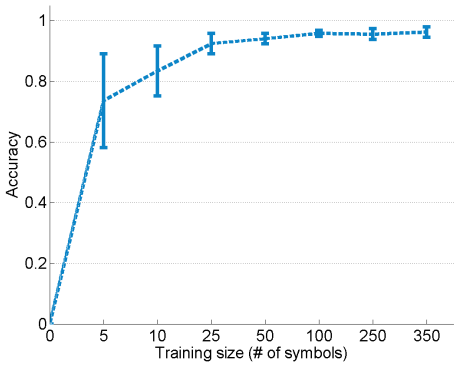
	Datasets				
	Shortstraw	IStraw	Speedseg	COAD	NicIcon
DPFrag (25 training examples)	**0.99**	0.92	0.77	**0.95**	0.78
DPFrag (50 training examples)	**0.99**	0.94	**0.87**	**0.95**	**0.82**
DPFrag (350 training examples)	**0.99**	**0.96**	**0.89**	**0.97**	**0.84**
IStraw	**0.99**	**0.96**	0.72	0.82	0.24
Sort-Merge-Repeat	0.97	0.78	-	-	-
Speedseg(User Agnostic)	0.86	-	0.78	-	-
Classyseg(User Agnostic)	-	-	0.75	-	-

Table 3.4: Accuracy comparison with other methods. Only the authors of IStraw made their algorithm available for testing. The Speedseg and Classyseg algorithms are not publicly available, hence we only list accuracies reported in the literature. The number of examples used by DPFrag is listed in parentheses.

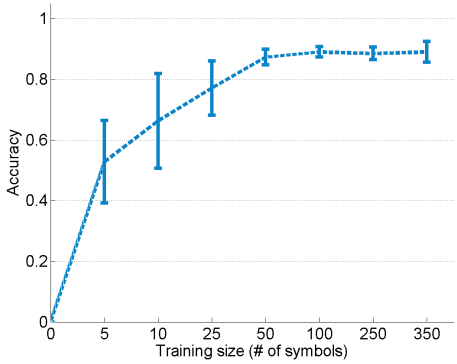
The first thing to note in Table-3.4 is that DPFrag either matches the performance



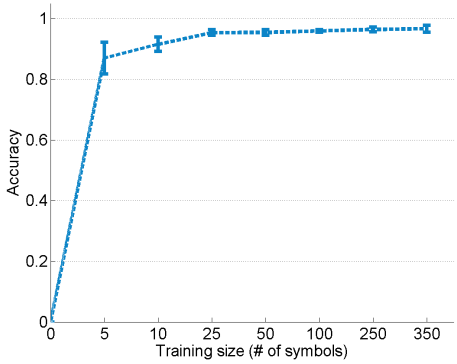
(a) Shortstraw



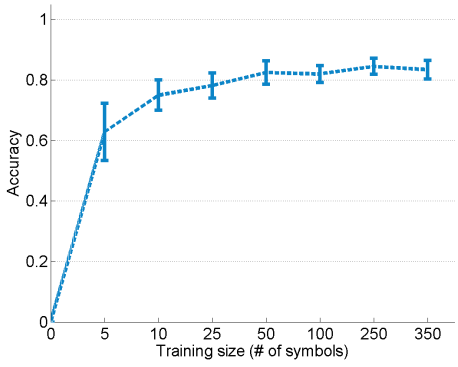
(b) IStraw



(c) Speedseg



(d) COAD



(e) NicIcon

Figure 3.8: Learning curves of DPFrag for each dataset. Error bars indicate the range of one standard deviation. DPFrag reaches its best performance with at most 50 examples.

of the state of the art (e.g. IStraw), or surpasses the state of the art with a wide margin (e.g. differences up to 17-60% with Shortstraw, Speedseg, Classyseg). The superior performance of DPFrag is especially pronounced when compared to the state of the art in new datasets. In particular, as seen in the case of IStraw, existing methods are highly tuned for their respective datasets, and they cannot generalize when they are given data from a different dataset. DPFrag, on the other hand, can generalize to new datasets with only a handful of training examples.

Inability to adapt to different datasets is a common problem encountered by methods that need hand tuning. Although they may achieve reasonable accuracies on datasets for which their parameters have been tuned, they generally fail in others. The problem is further aggravated by the fact that manual parameter tuning is hard, especially when there are too many parameters to tune. As an example, Speedseg has 12 free parameters, and it does worse than DPFrag in its own dataset. Its performance on the Shortstraw dataset is also lower than expected.

It should be noted that using machine learning alone does not result in a better performance unless it takes global features into account and searches for a globally optimal solution. This is illustrated by the case of Classyseg, which classifies the candidate corner points using local features. Its accuracy is also below the accuracy of DPFrag on the Speedseg dataset. Boosted performance of DPFrag shows the utility of using global features and searching for a global optimum in the space of all fragmentations, rather than relying on local decisions.

3.3.4 Fragmentation Speed

In order to facilitate real-time interaction, fragmentation algorithms should be efficient and fast. DPFrag is particularly well suited for real-time processing of strokes, because it can start processing a stroke even before it is completed by virtue of the optimal substructure property of our dynamic-programming-based approach. In Figure-3.9, we show the average time spent for processing strokes with varying number of segments. The times are discounted by the time spent for drawing each stroke. Negative

values in this figure indicate cases where DPFrag actually computes fragmentations faster than the users draw.

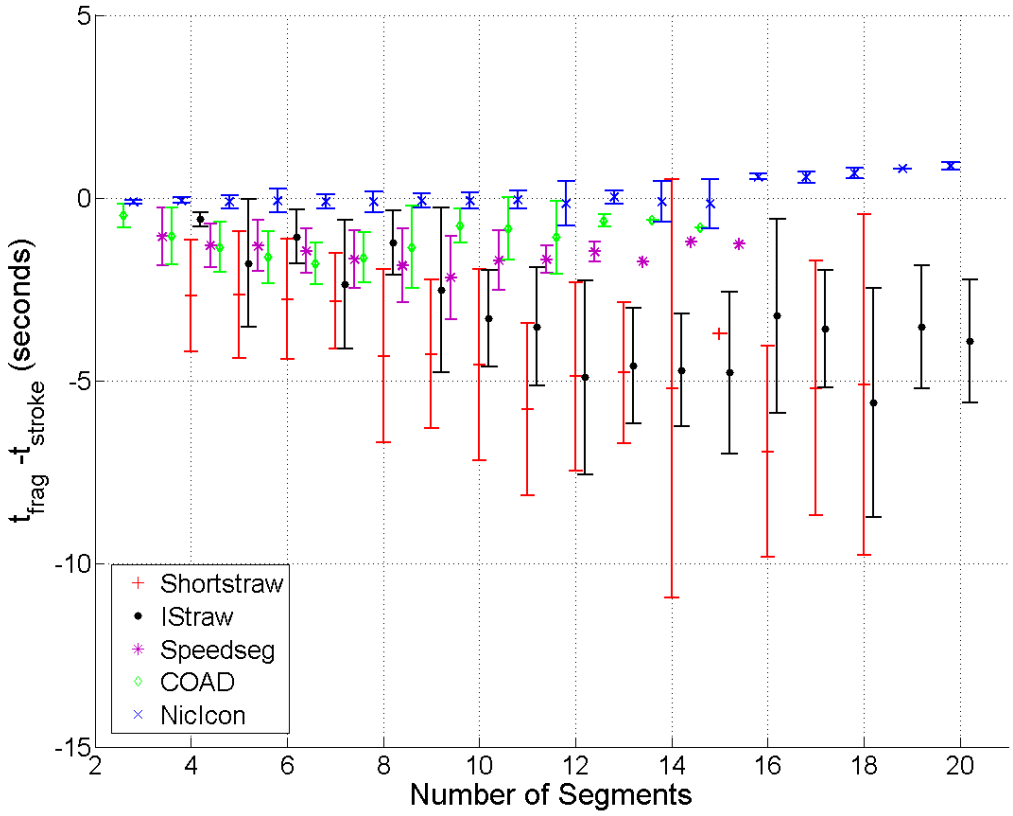


Figure 3.9: The net average difference between the time spent for processing and drawing strokes. Negative values indicate cases where DPFrag actually computes fragmentations faster than the users can draw. Error bars indicate one standard deviation.

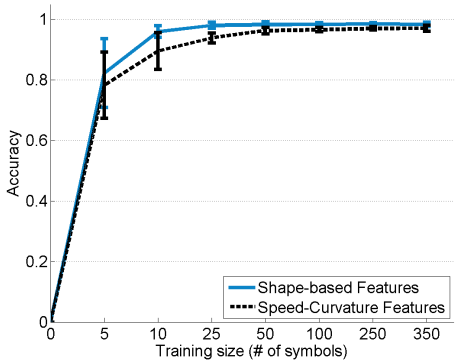
The speed measurements were carried out on a standard laptop using a non-optimized MATLAB implementation of DPFrag, hence in practice there is substantial room for improving the speed.

3.3.5 Utility of Feature Subsets

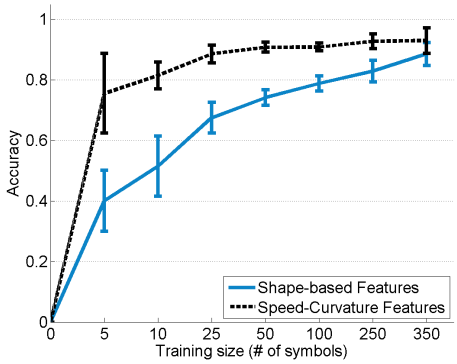
Fragmentation algorithms use speed, curvature and shape based features extensively in many forms. However, their utility in fragmentation has not been studied. In

addition to reporting fragmentation accuracies for feature level combination of all feature subsets, we also assess the utility of feature subsets by comparing fragmentation accuracies when DPFrag classifier is trained with individual feature subsets. The fragmentation accuracies for speed-curvature and shape based features are shown in Figure-3.10. As seen here, shape-based features outperform the speed and curvature based features in all datasets with the exception of the IStraw dataset. Superiority of shape based features is more pronounced in the in-context datasets, where strokes may have more corners with subtle drops in speed.

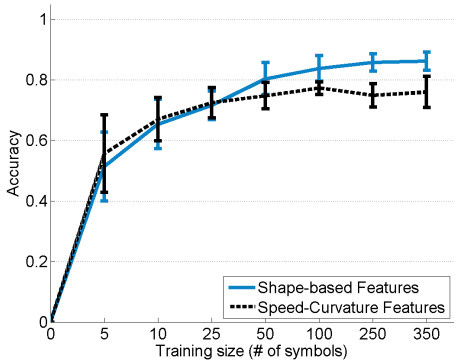
Feature level combination of feature subsets always results in a boosted fragmentation accuracy for all five datasets. Complimentary nature of shape, speed and curvature features makes it possible to achieve excellent performance with only a few training examples.



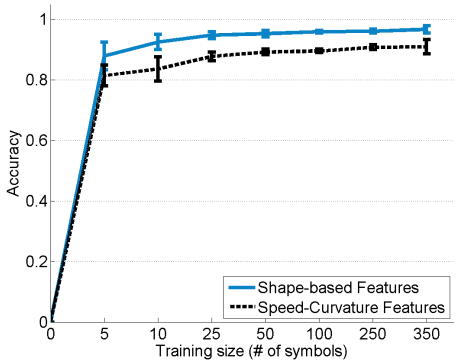
(a) Shortstraw



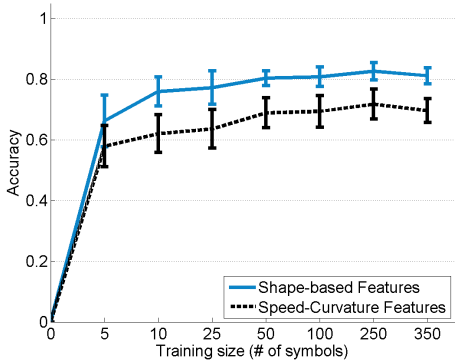
(b) IStraw



(c) Speedseg



(d) COAD



(e) NicIcon

Figure 3.10: Fragmentation accuracy of shape and speed/curvature based features for all five datasets. For clarity of presentation, curves representing accuracies obtained by all features are omitted. Those figures were already presented individually in Figure-3.8.

Chapter 4

SKETCH SEGMENTATION

Segmentation partitions a sketch into the domain objects. To find an optimal segmentation, we must completely explore the solution space. However, similar to the fragmentation, the search among all possible solutions is not tractable. To overcome the problem, some recent approaches use approximate inference methods such as Loopy Belief Propagation [Ouyang and Davis, 2011, Alvarado and Davis, 2006]. Others rely on assumptions about input. For example, Wu et al [Wu et al., 1999] assume that each symbol will be drawn independently and that the user will say the name of the symbol while drawing it. Kara et al [Kara and Stahovich, 2004] assume that the diagram consists of shapes linked by arrows. The dynamic programming (DP) based methods generally assume that the user draws objects sequentially, one after another [Arandjelović and Sezgin, 2011].

If the user draws objects sequentially and the stroke order information is available, then DP based approaches guarantee to find an optimal solution in polynomial time. But, sequential drawing assumption does not hold for offline and interspersed sketches, which are common in reality. The temporal information is not available for offline sketches, and the user does not follow the sequential drawing assumption in interspersed drawing.

In this chapter, for the offline and interspersed cases, we propose the concept of *spatial serialization*, which relaxes the assumptions of DP based approaches by imposing an order on the primitives. Spatial serialization is independent of temporal information, and it uses only spatial information to generate a primitive order. After finding a correct serialization, the DP based approaches become applicable for the optimal recognition of offline and interspersed sketches.

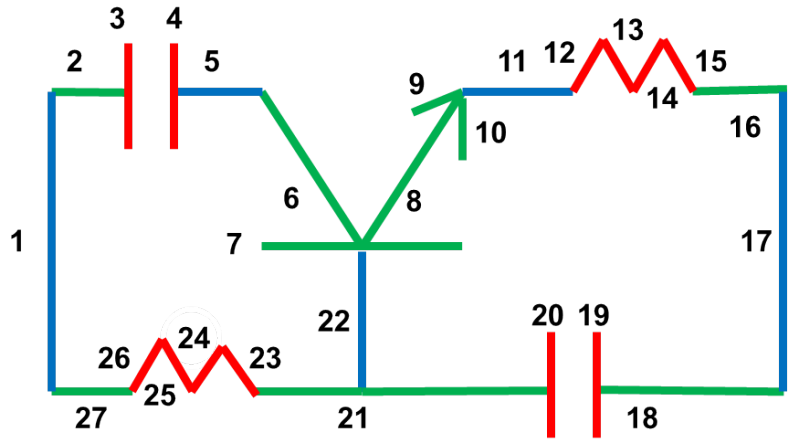


Figure 4.1: Primitives of each object must stay connected in a correct serialization. The serialization in the figure is correct, because the primitives of each object are connected to each other.

As illustrated in Figure-4.1, the primitives of each object must stay connected in a correct serialization. However, it is not necessary to determine each object to generate a correct serialization. In the next section, we show that we can build a correct serialization based on the pairwise coherence of primitives. For the sake of simplicity, the pairwise coherence can be estimated based on the distance between primitives. But, for a more accurate result, we must estimate the coherence using a more complicated model.

If we estimate the coherence correctly, it is possible to cast the serialization problem into the well-known "Traveling Salesman Problem (TSP)". In TSP, given a list of cities and the distances between each pair of cities, we are required to find the shortest route that visits each city exactly once. In the serialization problem, the distances between cities correspond to the incoherence between pairs of primitives. Instead of using TSP with exponential complexity, we can also generate a correct serialization by generating a partial order with a "Minimum Spanning Tree (MST)" solution. Similar to TSP chain, MST tree can also be segmented efficiently and optimally using dynamic programming.

We propose two different coherence models and three different graph algorithms

for spatial serialization. We conduct experiments to compare the accuracies of the different combinations of coherence models and graph algorithms. The results of the experiments show that the accuracy difference between MST and online methods is negligible. In addition, in terms of efficiency, TSP provides a balanced solution in terms of run-time and accuracy.

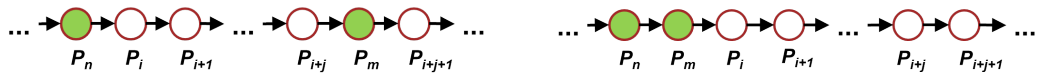
In the rest of the chapter, we first formulate the serialization and the segmentation problems. Then, we introduce different graph algorithms and coherence models for serialization. Finally, we report the comparison of different serialization schemes. We conclude with related work and discussions.

4.1 Problem Statement

The input to segmentation is an unordered set of primitives $S = p_1, p_2, ..., p_n$. Segmentation produces k groups $G = g_1, g_2, ..., g_k$. Classification gives us the labels for each group $L = l_1, l_2, ..., l_k$.

Given an unordered set of primitives, we first estimate coherence values between each pair of primitives, then we generate a spatial serialization. Finally, we segment and classify the domain objects. In the next section, we describe the serialization and the recognition problems formally.

4.1.1 Formulation for Serialization



(a) (b)

Figure 4.2: Suppose that p_n and p_m are siblings, primitives of the same object. If coherence values are estimated correctly, then the cost of H_1 in Figure-4.2(a) is higher than the cost of H_2 in Figure-4.2(b).

To formulate the serialization problem, we represent a sketch as a fully connected undirected graph, where each node represents the primitives, and each edge represents

the coherence between a primitive pair.

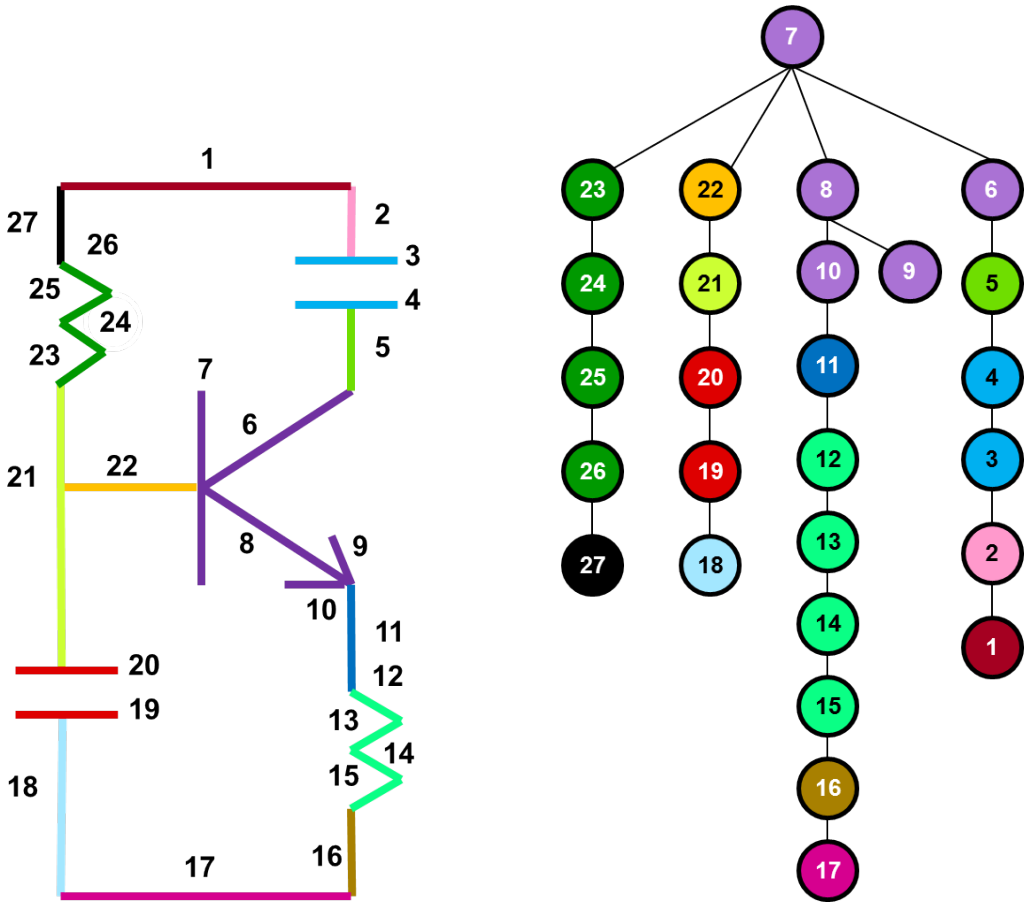
In this representation, a serialization corresponds to a Hamiltonian path (a path that passes through each vertex once). Since coherence graph is fully connected, it contains $n!$ different Hamiltonian paths. Among $n!$ serializations, several serializations are correct. In correct serializations, sibling nodes are connected each other. The Hamiltonian path with the minimum cost also corresponds to a correct serialization. We can easily prove the statement by contradiction:

Consider the Hamiltonian paths H_1 and H_2 in Figure-4.2. Suppose that H_1 is the Hamiltonian path with the minimum cost, but it separates the primitive p_m from its siblings. Another Hamiltonian path H_2 has the same topology, except p_m is connected to its siblings, and neighbors of p_m are connected to each other. The cost of H_2 is less than H_1, because the edge from p_m to its siblings has lower cost than the edges between non-sibling nodes. New connections of p_m's neighbors cannot increase the cost, because, in H_1, they are connected with p_m, which is not their sibling. Consequently, H_2 has a lower cost than H_1, which is a contradiction to the assumption that H_1 is the Hamiltonian path with the minimum cost.

As a result, we conclude that siblings cannot be interleaved in the Hamiltonian path with the minimum cost. In graph theory, finding the Hamiltonian path with the minimum cost is known as the Travelling Salesman Problem (TSP), which is NP-complete.

More generally, we can use a tree structure to represent an order, where parents have precedence over their children. However, a tree represents a partial order. If two nodes do not have a parent-child relation, then they can be ordered arbitrarily. In this form, the siblings must form a connected subgraph that corresponds to a correct serialization. There exist several trees corresponding to a correct serialization. The minimum spanning tree (MST) also corresponds to a correct serialization.

An MST can be created using The Kruskal's algorithm. It starts with an empty set, and creates the MST by iteratively adding the cheapest edge to this set. If the added edge forms a cycle, it is discarded, and the process continues with the next



(a) (b)

Figure 4.3: Siblings of each object must form a connected subtree in the MST.

cheapest edge. In the coherence graph, the sibling edges are always cheaper than non-sibling ones. Consequently, sibling edges are added to the set before non-sibling ones. Besides, it is always possible to form a connected graph without introducing a cycle. If siblings can be connected without introducing a cycle and the edges of sibling nodes are added before those of non-siblings, then the resulting MST corresponds to a correct serialization. An MST solution is illustrated in Figure-4.3. The run-time of Kruskal's algorithm on a complete graph is $O(n^2\ log(n))$, where n is the number of nodes in the graph.

As a result, we conclude that if coherence values are estimated correctly, then it is possible to generate a correct serialization using either the MST or the TSP solution.

4.1.2 Formulation for Segmentation

Given a segmentation G, the segmentation optimality $\mathcal{O}$ can be assessed in terms of the subsets of G :

$$P(\mathcal{O}|G) = \prod_{G_i \in G} P(O|G_i) \tag{4.1}$$

where $\mathcal{O}$ denotes the segmentation optimality, and O denotes the event of being a part of the optimal segmentation. Then, the optimality criterion for the segmentation can be written as:

$$\begin{aligned} G_{opt} &= \arg\max_{G \in \mathcal{G}} \ P(\mathcal{O}|G) \\ &= \arg\max_{G \in \mathcal{G}} \ \prod_{G_i \in G} P(O|G_i) \end{aligned} \tag{4.2}$$

where $\mathcal{G}$ represents the set of all possible segmentations. To avoid the numerical underflow of probabilities, we replace the probability term with the log probability:

$$G_{opt} = \arg\min_{G \in \mathcal{G}} \ -\sum_{G_i \in G} log\ P(O|G_i) \tag{4.3}$$

Given the cost function in Equation-4.3, 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 is not feasible, because the number of possible segmentations grows exponentially with the number of primitives. Through serialization, we transform $\mathcal{G}$ into a tractable form, for which the substructure property of dynamic programming holds. The optimal substructure property allows us to use dynamic programming to search for G_{opt} efficiently and optimally. We prove that segmentation of the tree exhibits the optimal substructure property according to Equation-4.3. We omit the proof for the segmentation of a chain, because a chain is a special instance of a tree.

A problem exhibits an optimal substructure if an optimal solution to the problem contains within it optimal solutions to the subproblems. Suppose that an optimal segmentation splits the coherence tree at a set of boundary primitives P_B. For instance, $P_B = \{p_5, p_{11}, p_{22}, p_{23}\}$ for MST in Figure-4.3(b). Then, the segmentation of subtrees rooted at each element of P_B must also be optimal. Because, otherwise, a less costly segmentation of the subtrees could have been substituted in, to obtain a better segmentation, which is a contradiction. This shows that any optimal solution contains within it optimal solutions to the subproblems, and the optimal substructure property holds for the segmentation problem according to the optimality criterion in Equation-4.3. Thus, we can build an optimal solution by solving subproblems using dynamic programming.

Next, we define the cost of an optimal solution recursively in terms of optimal solutions to the subproblems. If there is no primitive, which is the trivial case, the cost is zero. Assuming that segmentation of a subtree 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 its optimal segmentation is:

$$C[j] = \left(\sum_{p \in P_B \text{ of } G_j} C[p] \right) - log\ P(O|G_j) \tag{4.4}$$

Thus, our recursive definition for the minimum cost is:

$$C[j] = \begin{cases} 0 & j = 0 \\ \min\limits_{G_j \text{ subtree of } p_j} \left(\left(\sum_{(p \in P_B \text{ of } G_j)} C[p] \right) - log\ P(O|G_j) \right) & j > 0 \end{cases} \tag{4.5}$$

Instead of computing the solution by recurrence, we can iteratively compute the optimal cost using a bottom-up approach. The calculation of the term $P(O|G_j)$, which is the likelihood of a primitive group being a part of the optimal segmentation, in Equation-4.3 will be explained in the next section.

4.2 Implementation

Our framework consists of three modules: The first module, **serialization**, creates a non-cyclic tree of primitives. The second module, **segmentation**, partitions the serialized primitives into domain objects. The last module, **classification**, labels the segmented objects. An overview of the framework is presented in Figure-4.4.

4.2.1 Serialization Module

In the serialization module, we first estimate the coherence between each pair of primitives, which requires $O(n^2)$ operations. Then, we assign the estimates to the edges of the coherence graph as weights, and we apply a MST or TSP solution to the coherence graph. Finally, we select a root node, and orient the coherence graph from root to the leaves. The output of the serialization module is an acyclic spanning graph. In the final serialization, ideally, each node is connected to its siblings over a non-blocking path.

The serialization module consists of two components: a coherence model and a graph algorithm. The **Coherence model** produces the coherence estimates for primitive pairs. The **graph algorithm** generates the acyclic spanning graph based on the coherence estimates.

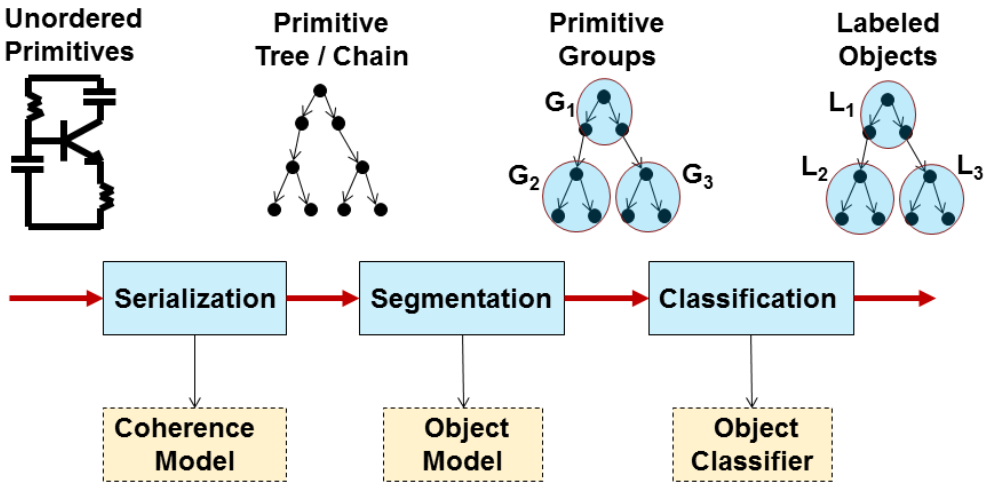
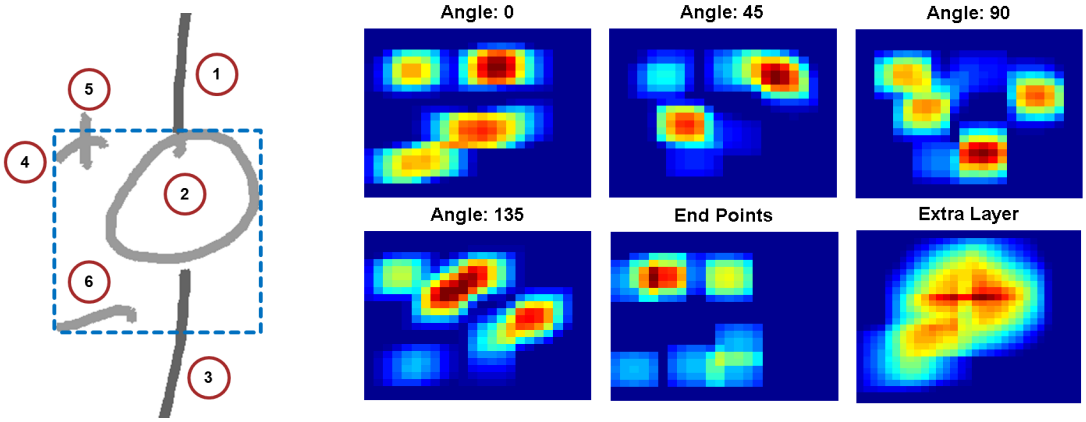


Figure 4.4: The first module, serialization, creates a fully connected coherence graph, and then it generates a serialization by applying a TSP or MST solution. It uses the coherence model to estimate the coherence between the primitive pairs. The second module, segmentation, partitions the serialization tree into the domain objects. It uses the object model to estimate $P(O|G_j)$. The third module, classification, is responsible for labeling the primitive groups. It uses the type classifier to estimate the label for the segmented shapes.



(a) (b)

Figure 4.5: PairIDM feature for p_2 and p_6 of the voltage source object.

Coherence Model

The coherence model estimates the likelihood of a given pair being siblings. In this study, we use two different coherence models. The first is a simple model based on the Euclidean distance between primitives, and the second model is a regression model based on image features.

- **Distance Model:** The distance model assumes that siblings are spatially close to each other, and estimates the coherence as the normalized minimum distance between primitives.

$$C_d(p_i, p_j) = 1 - \frac{d(p_i, p_j)}{D_{max}} \tag{4.6}$$

  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 the PairIDM feature, which is a modified version of the original IDM, to estimate the coherence between primitive pairs. In the original Image Deformation Model (IDM)[Ouyang, 2009], the strokes are first converted into a raster image. Then, the image is filtered in different directions (0, 45, 90, 135 degrees). Additionally, a smoothed raster image of the stroke end points is added to the filtered images. Finally, all five filtered images are stacked into a feature vector.

  In the extraction of PairIDM feature, in addition to original IDM layers, we also generate a smoothed image of the isolated primitive pairs. The extra layer provides an encoding for the spatial layout of the primitive pair, while the original IDM layers capture the spatial layout of their context. Figure-4.5 shows the PairIDM layers for primitives p_2 and p_6. Our specific implementation for PairIDM consists of six images of 9x9 pixels.

The training set of the regression model is composed of features extracted from sibling and non-sibling primitive pairs in training sketches. After we extract features, we apply a PCA transformation for the dimension reduction. With transformed features, we train a Support Vector Regression (SVR) with RBF kernel. We optimize feature extraction and regression model parameters with a 5-fold cross validation on the training set.

Graph Algorithms

We use three different graph algorithms for serialization.

- **Kruskal's Solution to MST**: The Kruskal's algorithm starts with an empty set, and creates the MST by iteratively adding the cheapest edge to this set. If the edge forms a cycle, the process continues with the next cheapest edge. The algorithm's complexity is $O(n^2\ log(n))$ for the fully connected coherence graph.

- **Exact Solution to TSP**: The exact solution to TSP problem is NP-complete. However, the state-of-the art TSP solvers has acceptable run-times up to a few hundred nodes. In our implementation, we use Concorde TSP solver, which is regarded as the fastest among currently available TSP solvers.

- **Controlled Random Search Solution to TSP**: We also use a genetic algorithm based solution to TSP problem. The complexity of the algorithm is $O(n^2)$. The number of iterations are limited, but the cost calculation at each iteration requires $O(n^2)$ operations.

4.2.2 Segmentation Module

Given a primitive serialization, the segmentation module iteratively builds G_{opt} based on Equation-4.3. For each subproblem, it first calculates the optimal solution using a bottom-up approach. Then, it stores the best solution for each subproblem. Finally, it backtracks the solutions to subproblems and combines them to produce G_{opt}.

Object Model

$P(O|G_i)$ is the likelihood that primitive group G_i is part of the optimal segmentation. To be the part of the optimal segmentation, G_i must be a *valid domain object of a specific type*. As a result, we can define $P(O|G_i)$ as the joint probability of three random variables:

- **H:** The event of being a valid object hypothesis.
- **D:** The event of being an object in the given domain.
- **T:** The type of the object.

We express the joint probability as follows:

$$P(O|G_i) = P(T, H, D|G_i) = P(H|G_i)\ P(D|H, G_i)\ P(T|H, D, G_i) \tag{4.7}$$

where each variable is designed to assess the quality of each G_i hierarchically.
We train three different classifiers for each term in Equation-4.7.

- **Validity Classifier:** The validity classifier estimates $P(H|G_i)$, which is the posterior probability of a given primitive group being a valid hypothesis. It checks whether the given hypothesis is valid using simple features such as the number of primitives and the size of the bounding box. It produces binary results, and we use it as a filter to reduce the load on the other classifiers [Tax, 2015]. It is a one-class classifier trained with positive examples. In contrast with the current methods in the literature, we do not apply a set of hard-coded constraints for object hypotheses, but our validity classifier learns the constraints from the training data.
- **Object/Non-object Model:** The second model uses the IDM feature with a PCA transformation to estimate $P(D|H, G_i)$, which is the posterior probability

that the given valid hypotheses G_i is a domain object. To generate the training instances for the model, we segment them using the validity classifier. This trial reveals the negative instances that are failed to be filtered by the validity classifier. In the training of the model, we use the negative instances from the segmentation trial and the domain objects in the training set, as the negative and the positive instances respectively.

- **Type Classifier:** The third model estimates the $P(T|H, D, G_i)$, which is the posterior probability that the given hypothesis is of a specific object type. To train the multi-class classifier, we extract the IDM feature from the objects in the training set. The type classifier does not contain a reject class, because, the object/non-object model filters the non-objects.

We train all of the models using LIBSVM implementation of SVM with RBF kernel. We train each model with parameters optimized by 5-fold cross validation, and estimate $P(O|G_j)$ according to Equation-4.7.

4.2.3 Classification Module

Given a segmentation G, Classification module creates the labels L associated with each primitive group. In our implementation, we do not use a separate object classifier for classification. Instead, we use the type classifier of the object model.

4.3 Evaluation

To evaluate the proposed methods, we conducted experiments on three datasets from different domains. In all experiments, we trained both the coherency and the object models with the same training instances, and tested the recognition accuracy on the rest of the dataset. We used two different cross validation schemes: 5x2cv and 5x10cv. We prefer 5x2cv scheme for statistical tests, because 5x2cv scheme is reported to have acceptable type-I error in the comparison of the supervised machine learning methods

[Dietterich, 1998]. In other types of accuracy evaluations, we also present the results of 5x10cv.

4.3.1 Serialization Schemes

In the experiments, we evaluate six different serialization schemes, which are the combinations of three different graph algorithms and two different coherence models. Table-4.1 presents a comparison between these different serialization schemes. Besides, we also use two additional methods for evaluation purposes: Upper-Bound method and Closest-Primitive method.

- **Upper-bound Method:** The Upper-Bound method creates a correct serialization using the ground-truth coherence values and the exact TSP solution. However, if we assign binary coherence values according to the ground-truth, we may end up with a serialization, which connects the distant objects. To avoid such an easily recognizable serialization and to emulate an online sketching, we add a normalized distance element C_d to the ground-truth. We assign the following coherence to the primitive pair p_i and p_j:

$$C_u(p_i, p_j) = \begin{cases} 1 + C_d, & \text{if } \mathrm{p_i} \text{ and } \mathrm{p_j} \text{ are siblings} \\ C_d, & \text{otherwise} \end{cases} \tag{4.8}$$

  Coherence values estimated by Equation-4.8 are guaranteed to create a perfect serialization. This serialization is also compatible with a common drawing style, in which the user continues with the next closest object after finishing the current one.

- **Closest-Primitive Method:** The Closest-Primitive method first adds a random primitive to the serialization chain, then it continues with the closest primitive. The Closest-Primitive method is greedy, and it uses a distance-based co-

herence model. It theory, it produces correct serializations if objects do not contain branches and gaps between siblings.

	Algorithm	**Coherence**	**Complexity**	**Optimal**
Upper-Bound	Exact TSP	True 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	Euclidean	$O(n^2\ log(n))$	Yes
TSPD	Exact TSP	Euclidean	$O(n^2\ 2^n)$	Yes
TSPDGA	CRS[1]	Euclidean	$O(n^2)$	No
Closest-Primitive	Greedy	Euclidean	$O(n)$	No

Table 4.1: Properties of serialization schemes.

4.3.2 Datasets

Sketches with cyclic connections are challenging for serialization. For this reason, we intentionally select three datasets with an excessive number of cycles. All three datasets are publicly available.

- **Analog Electrical Circuit Dataset**: EC dataset contains 110 Analog Electrical Circuit sketches by 10 users. The objects in EC dataset are wire, resistor, capacitor, ground, battery, transistors, voltage and current sources. The dataset is an element of Experimental Test Corpus of Hand Annotated Sketches (ETCHA)[Oltmans et al., 2004].

- **Flow-chart Dataset**: FC dataset contains 419 flow-chart sketches by 35 users [Awal et al., 2011]. The objects in the dataset are process, decision, data, connector, terminator and arrow.

	Electrical	**Flowchart**	**Automata**
#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

Table 4.2: Statistics of the datasets.

- **Finite Automata Dataset**: FA dataset contains 300 finite-automata sketches by 25 users [Bresler et al., 2014]. The objects in the dataset are state, final state and arrow.

We show representative sketches of EC, FC and FA datasets, along with their statistics in Figure-4.6 and Table-4.2 respectively.

4.3.3 Accuracy Metrics

Several schemes are proposed for the evaluation of sketch recognizers. Some methods determine the accuracy by comparing the number of points in the objects [Feng et al., 2009]. Others calculate the ratio of overlap for the bounding boxes [Ouyang and Davis, 2011]. 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 and the bounding boxes must be identical with the ground-truth. In AoN, the detection is false, even if a single primitive mismatches with human-annotations. In accuracy comparisons, we also use %50 bounding box overlap metric of Pascal challenge (BB %50), because, the methods in the literature adopts the same metric in evaluations. In BB %50, a de-

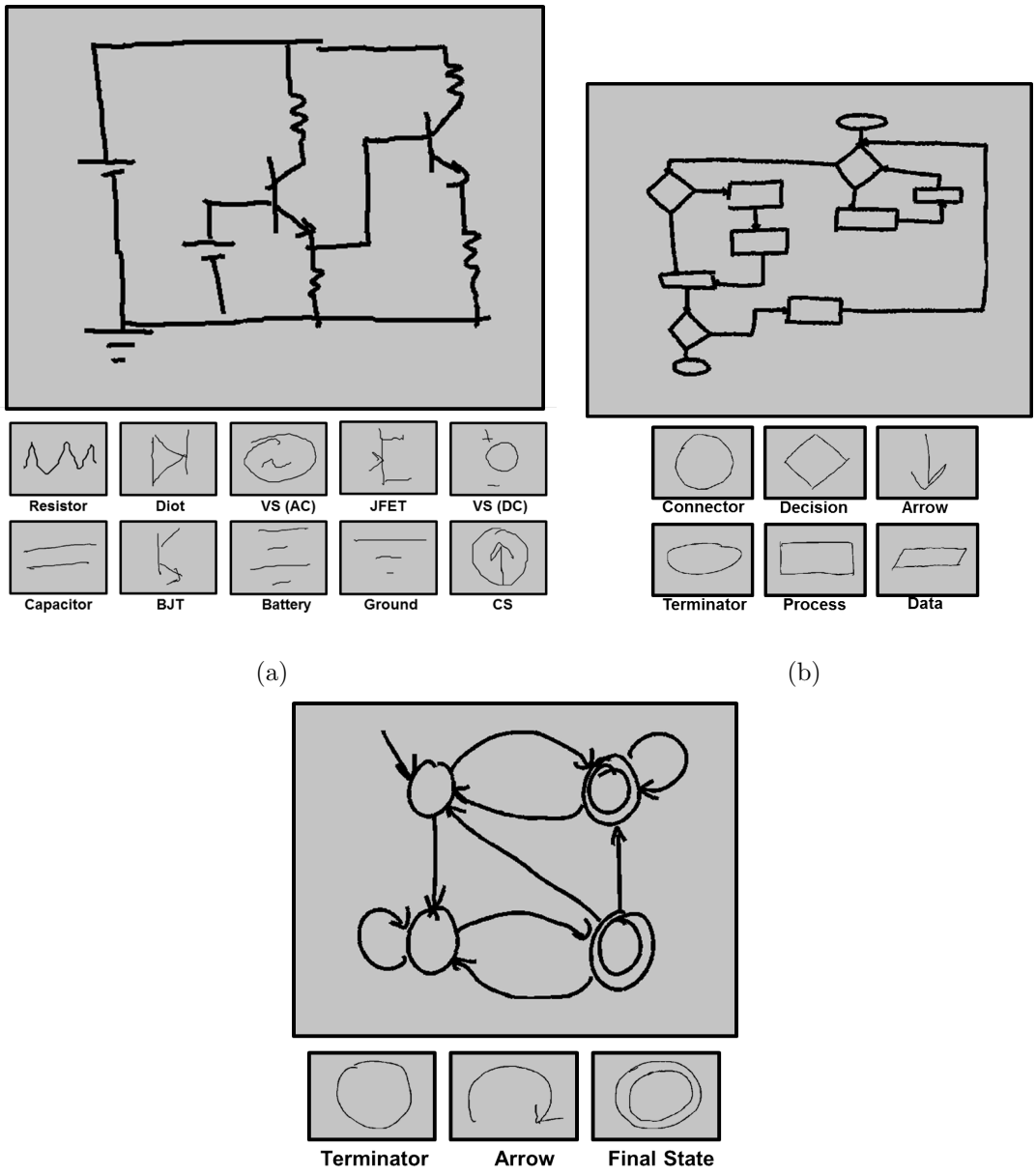


Figure 4.6: Representative sketches from EC, FA and FC datasets.

tection is correct, if the 50% area of its bounding box overlaps with the ground-truth object.

In accuracy reporting, 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 constitutes almost half of the dataset. Thus, cumulative averaging of accuracy metric without a categorization of types is simply misleading.

- **Serialization Accuracy:** An object is serialized correctly, if all its primitives form a connected subgraph in the coherence graph. The serialization accuracy is always an upper bound for the segmentation and recognition recall. An object cannot be correctly segmented, if it is not correctly serialized. We always report AoN accuracy for serialization.

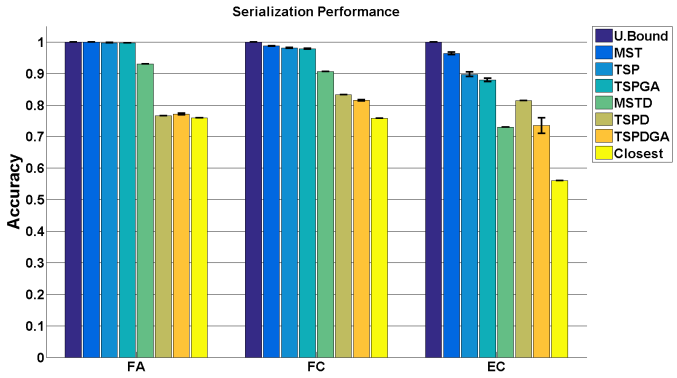
- **Segmentation/Recognition Accuracy:** To calculate the accuracy metrics on dataset D, we first perform the segmentation and recognition using method M, then we calculate the precision, recall and F1 scores for each object type in the dataset. We report their average as the accuracy of M on D.

4.3.4 Accuracy Evaluation

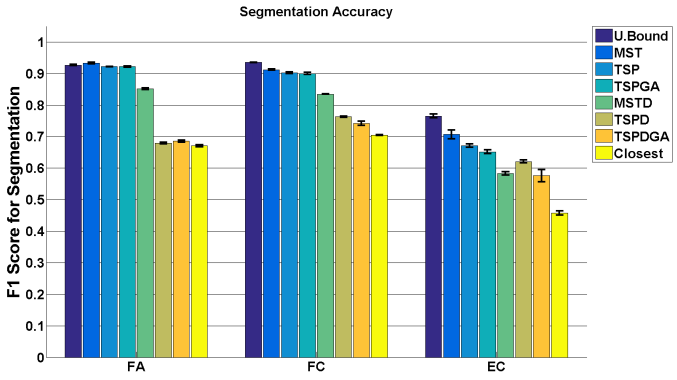
Figure-4.7 and Table-4.3 show the averages of serialization accuracies and F1 scores of segmentation/recognition. Inspection of the Figure-4.7 points out a strong correlation among the order of accuracies of different datasets. It also shows that regression based serialization schemes outperform the distance based schemes. However, the order of the graph algorithms depends on the coherence model. For a more sound analysis, we conduct several statistical tests to reveal the effect of different serialization methods on accuracy.

Accuracy Comparison on Multiple Datasets

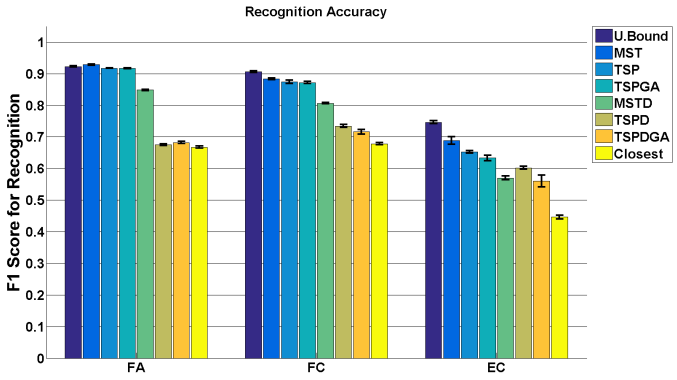
To ensure the maximum independence between accuracy samples, we use 5x2cv in the accuracy comparison of different serialization schemes [Dietterich, 1998]. In the analysis, we refrain from using parametric tests such as 3 (dataset) x 8 (method) mixed design ANOVA, because the inspection of the accuracy values shows that the



(a)



(b)



(c)

Figure 4.7: Average F1 scores obtained by a 5x10cv across repetitions. Error bars show one standard deviation of accuracies.

	Ser.Acc.			F1 Score(AoN)			F1 Score(50% BB)		
	EC	FA	FC	EC	FA	FC	EC	FA	FC
U.Bound	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 4.3: Averages of serialization accuracy and recognition F1 scores across repetitions of 5x10cv.

assumptions of parametric tests fail. Variance stabilization transformations, such as arcsine-square-root, also do not change the results of the assumption checks.

Violation of test assumptions is commonly encountered while comparing different machine learning algorithms, and non-parametric tests are suggested for such cases [Demšar, 2006]. For this reason, we used Friedman non-parametric test, which is a non-parametric equivalent of repeated measures ANOVA, to compare the accuracies of different methods on multiple datasets. The Friedman test indicated that different serialization methods result in statistically significantly different recognition accuracies, $\tilde{\chi}^2(7)=20.00$, $p=0.006$. A Nemenyi post hoc test also indicated that scores of the MST and the Upper-Bound method were statistically significantly higher than the scores of Closest-Primitive method, $p<0.05$. In addition, the recognition accuracy of Closest-Primitive method was statistically significantly lower than all proposed methods, $p<0.05$.

Figure-4.8 shows a critical difference diagram for the recognition accuracies. The critical difference diagram displays the mean rank of the set of methods over datasets. The cliques of methods with statistically similar performance are connected by a bar.

	Recall (AoN)			Recall (50% BB)		
	EC	FA	FC	EC	FA	FC
U.Bound	0.76	0.93	0.90	0.86	0.95	0.94
MST	0.70	0.93	0.88	0.81	0.95	0.94
TSP	0.66	0.92	0.87	0.79	0.94	0.93
TSPGA	0.64	0.92	0.87	0.78	0.94	0.93
MSTD	0.57	0.87	0.79	0.70	0.91	0.86
TSPD	0.60	0.72	0.71	0.72	0.74	0.78
TSPDGA	0.55	0.72	0.68	0.67	0.75	0.74
Closest-Primitive	0.43	0.71	0.64	0.52	0.74	0.72

Table 4.4: Averages of recognition recall across repetitions of 5x10cv.

The critical difference required for a statistical superiority is labelled as "CD".

Besides comparing the accuracies of different serialization schemes, we also compared the accuracies of different coherence models and graph algorithms. Wilcoxon signed rank test determined 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 based model (Mdn=0.677), Z= -2.666, p= 0.008.

In the comparison of different graph algorithms, Friedman test determined that recognition accuracy was statistically significantly different for graph algorithms, $\tilde{\chi}^2(2)$=7, p=0.030. Post hoc analysis with Benforroni 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.

Effect of Modules on Accuracy

Our framework consists of three modules: serialization, segmentation and classification. In this section, we analyze the effect of the framework modules on recognition accuracy. Figure-4.9 shows the relation between the input and output accuracy of

	Prec.(AoN)			Prec.(50% BB)		
	EC	FA	FC	EC	FA	FC
U.Bound	0.74	0.92	0.91	0.83	0.94	0.94
MST	0.68	0.93	0.89	0.79	0.95	0.94
TSP	0.66	0.92	0.88	0.80	0.94	0.94
TSPGA	0.64	0.92	0.88	0.79	0.94	0.93
MSTD	0.59	0.85	0.83	0.74	0.88	0.91
TSPD	0.63	0.81	0.78	0.77	0.83	0.89
TSPDGA	0.60	0.81	0.78	0.75	0.83	0.89
Closest-Primitive	0.55	0.80	0.75	0.72	0.82	0.87

Table 4.5: Averages of recognition precision across repetitions of 5x10cv.

the segmentation and the recognition modules.

Figure-4.9(a) shows that the segmentation and recognition accuracies are nearly equal for all datasets, which means that objects are generally classified correctly, if they are correctly segmented.

Figure-4.9(b) reveals a linear relationship between serialization and segmentation accuracy. It is evident that the dataset and serialization accuracy has a main effect on the recognition accuracy. However, the interaction between the dataset and serialization accuracy is minimal. It means that the effect of serialization accuracy on the recognition accuracy is similar in all datasets. Recognition accuracy increases linearly with serialization accuracy for all datasets in a similar trend.

Figure-4.9(b) also shows that segmentation is the common source for recognition errors, because the error introduced by the serialization and recognition is negligible when compared to the errors of the segmentation.

We can further decompose the segmentation errors onto the components of segmentation module. The segmentation module has two components: a DP based segmentation algorithm and an object model. Since, DP based algorithm is proven

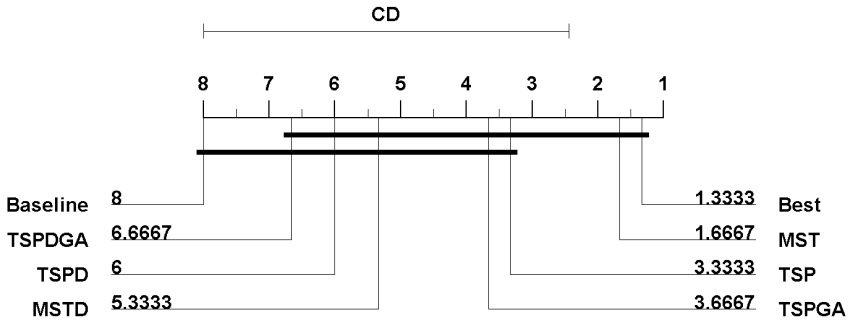


Figure 4.8: Critical difference diagram [Demšar, 2006] showing the average ranks of different serialization schemes.

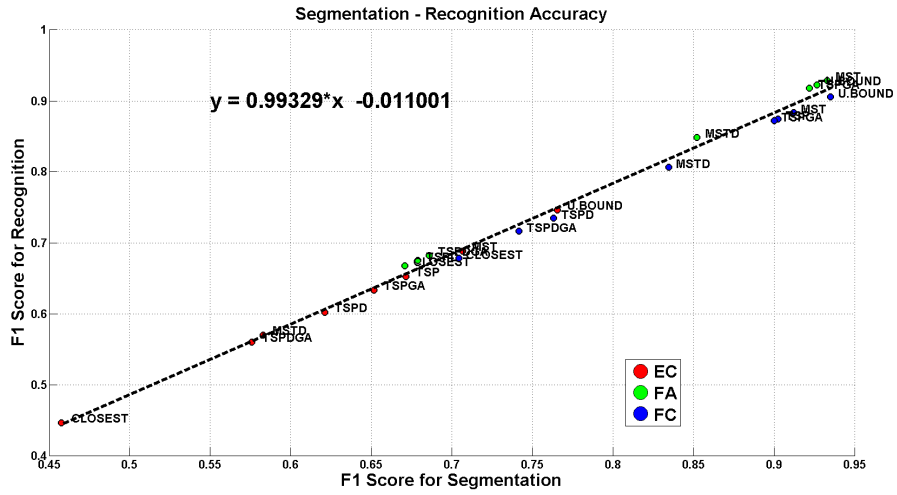
to be optimal, error associated with segmentation module can be attributed to the object model.

4.3.5 Run-time Comparison

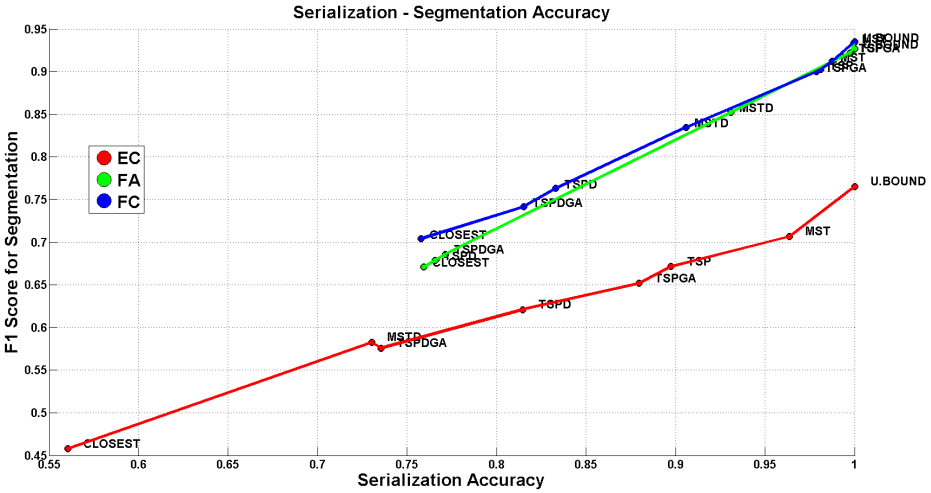
In order to facilitate real-time interaction, sketch recognizers should be efficient and fast. In this section, we analyze the effect of serialization and segmentation on run-time:

- **Serialization Module:** The complexity of serialization is different for each serialization scheme. The complexity of the exact TSP solution is exponential, while the complexity of MST is $O(n^2\ log(n))$. CRS solution to TSP has complexity of $O(n^2)$, but it is not an exact solution to TSP.

- **Segmentation Module:** The complexity of the segmentation depends on the structure of the serialization tree, because the number of subtrees rooted at a specific vertex varies with the structure of the tree. If the tree is a chain, (similar to TSP solution), then the number of subtrees is $O(n^2)$. For MST based solutions, the worst case is a star topology, in which the number of subtrees grows exponentially with the number of vertices.



(a)



(b)

Figure 4.9: The interaction plots for accuracies in segmentation and recognition modules. In all datasets, segmentation and recognition accuracies are nearly equal. The main source of recognition errors is the segmentation module.

The recognition accuracy of the serialization schemes and their total run-time exhibit a trade-off, which is presented in Figure-4.10. The optimal choice for a serialization method differs across applications. The elbow of the trade-off curves suggests that TSP is the optimal serialization scheme when run-time and recognition accuracy

are equally important.

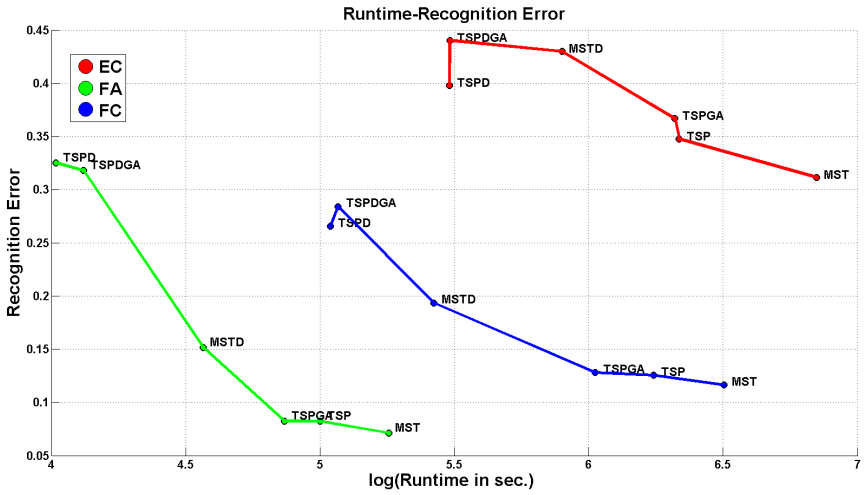


Figure 4.10: The trade-off between runtime and recognition accuracy.

4.3.6 Comparison with Recent Methods in the Literature

In this section, we compare the proposed methods with the state-of-the art in sketch recognition. Several methods are evaluated using EC, FA and FC datasets in recent years. But, unfortunately, all of these methods either use stroke ordering or temporal information in recognition.

Offline sketch recognition methods are rare in the literature, and the datasets used in their evaluation are not publicly available. For this reason, we evaluate the proposed offline methods with online methods. However, in this comparison, we note that it is not fair to compare the offline and online methods, because, several studies indicate a substantial increase in the recognition accuracy when sketch recognizers use temporal information [Arandjelović and Sezgin, 2011, Delaye and Lee, 2015]. For example, Delaye et al [Delaye and Lee, 2015] declares a 25% increase in the recognition accuracy, when the stroke ordering is used on FC dataset. However, the Upper-Bound can be safely compared with other methods, because the Upper-Bound method uses the correct serialization information in recognition. The comparison of Upper-Bound

method can be considered as the evaluation of segmentation module, because our Upper-Bound method also uses the correct serialization.

We report the average accuracies either obtained by 5x2cv or 5x10cv. In the selection of the cross validation scheme, we base our decision on the evaluation approach of the other methods.

EC Dataset

Oltmans [Oltmans, 2007] and Ouyang [Ouyang, 2012] use the EC dataset for the evaluation. EC dataset is the most challenging dataset among three datasets in terms of the number of object types. Oltmans' method [Oltmans, 2007] detects the locations of candidate objects, and, then combines the detections into a coherent solution. Oltmans evaluates his method using 50% overlap ratio criteria of the Pascal challenge. His method achieves a recall=0.739, precision=0.257 and F1 score=0.3814. The other method developed by Ouyang first creates a Conditional Random Field (CRF) over primitives and object candidates, and then performs the approximate Loopy Belief Propagation. Finally, it combines the object hypotheses into a coherent solution using Agglomerative Clustering. Ouyang performs 10-fold cross validation, and he evaluates the method using 50% overlap ratio criteria.

In the comparison with the proposed methods, we use the average of 5x10cv experiment, with a 50% overlap ratio criteria. Comparison with Oltmans' method shows that the coherence based methods have higher accuracy than Oltmans' method. Table-4.7 shows the accuracy comparison with Ouyang's method. In a comparison with Ouyang's method, a Wilcoxon signed rank test determines no significant difference between the F1 scores of Ouyang's method and the F1 scores of Upper-Bound method, $Z = 28, p = 0.959$.

FA Dataset

Bresler et al [Bresler et al., 2014] use FA dataset for evaluation. In Table-4.9, we list the recall of Bresler's method along with the recall values of our proposed methods.

	MST	TSP	TSPGA	MSTD	TSPD	TSPDGA	Closest
Resistor	0.99	0.98	0.98	1.00	0.97	0.91	0.95
Capacitor	1.00	0.98	0.99	0.30	0.88	0.84	0.81
Ground	0.98	0.98	0.98	0.89	0.96	0.96	0.74
Diode	0.93	0.88	0.87	0.75	0.94	0.82	0.79
BJT	0.96	0.74	0.74	0.93	0.52	0.41	0.36
JFET	0.80	0.47	0.41	0.53	0.24	0.19	0.06
Battery	1.00	1.00	0.99	0.23	0.87	0.70	0.71
Voltage Source	1.00	0.83	0.76	0.60	0.73	0.58	0.13
Current Source	0.94	1.00	0.96	0.85	0.88	0.78	0.40
AC Source	1.00	1.00	1.00	0.96	0.96	0.88	0.21
Average	0.96	0.89	0.87	0.70	0.80	0.71	0.52

Table 4.6: The serialization accuracy comparison on EC dataset with 5x10cv.

	Ouyang	U.Bound	MST	TSP	TSPGA	MSTD	TSPD	TSPDGA	Closest
Resistor	0.96	0.82	0.81	0.82	0.82	0.82	0.81	0.81	0.82
Capacitor	0.80	0.78	0.58	0.59	0.60	0.26	0.66	0.62	0.58
Ground	0.86	0.90	0.83	0.83	0.82	0.86	0.83	0.82	0.74
Diode	0.91	0.79	0.70	0.71	0.72	0.69	0.75	0.68	0.72
BJT	0.79	0.81	0.81	0.78	0.74	0.74	0.52	0.49	0.41
JFET	0.71	0.91	0.84	0.72	0.72	0.76	0.46	0.37	0.25
Battery	0.77	0.73	0.75	0.81	0.80	0.36	0.76	0.68	0.67
Voltage Source	0.92	0.95	0.88	0.89	0.88	0.86	0.86	0.82	0.32
Current Source	0.86	0.86	0.86	0.83	0.82	0.80	0.76	0.73	0.51
AC Source	0.82	0.86	0.86	0.87	0.87	0.85	0.86	0.84	0.35
Average	0.84	0.84	0.79	0.78	0.78	0.70	0.73	0.68	0.54

Table 4.7: The F1 score comparison on EC dataset with 5x10cv and 50% BB.

We obtain the recall values by averaging the results of 5x2cv across repetitions. Delaye et al [Delaye and Lee, 2015] also evaluate their SLAC based algorithm on FA dataset. They report an average recall of 97.20%, but they do not report a recall per object type.

	U.Bound	MST	TSP	TSPGA	MSTD	TSPD	TSPDGA	Closest
State	1.00	1.00	1.00	1.00	1.00	0.99	0.99	0.99
Final State	1.00	1.00	1.00	1.00	0.89	0.31	0.33	0.33
Arrow	1.00	1.00	1.00	0.99	0.90	0.99	0.99	0.95
Average	1.00	1.00	1.00	1.00	0.93	0.77	0.77	0.76

Table 4.8: The serialization accuracy comparison on FA dataset with 5x10cv.

Both FC and FA datasets contain graphic objects and handwriting objects such as labels. In the evaluations, we discard handwriting objects, and report the recall values of graphic objects, because, separation of text and graphics objects can be considered as a pre-processing step in recognition. Separation of text and graphics is also an extensively studied topic in the literature.

	Bresler	U.Bound	MST	TSP	TSPGA	MSTD	TSPD	TSPDGA	Closest
State	0.95	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98
Final State	0.96	0.85	0.87	0.85	0.85	0.76	0.25	0.27	0.26
Arrow	0.89	0.95	0.94	0.94	0.94	0.86	0.92	0.92	0.89
Average	0.94	0.93	0.93	0.92	0.92	0.87	0.72	0.72	0.71

Table 4.9: Recall comparison on FA dataset with 5x10cv.

FC Dataset

Carton [Carton et al., 2013], Lemaitre [Lemaitre et al., 2013], Bresler [Bresler et al., 2014], and Delaye et al [Delaye and Lee, 2015] use FC dataset for evaluation. Carton et

al [Carton et al., 2013] and Lemaitre et al [Lemaitre et al., 2013] propose rule-based approaches for sketch recognition. Bresler [Bresler et al., 2014]introduces an arrow detection based method for sketch recognition. We present a comparison of recall values for all these methods in Table-4.11. In the evaluation of our proposed schemes, we use the averages of 5x2cv across repetitions. We do not report precision and F1 scores, because other methods do not report theirs either.

	MST	TSP	TSPGA	MSTD	TSPD	TSPDGA	Closest
Terminator	0.99	1.00	1.00	1.00	0.98	0.98	0.94
Arrow	0.98	0.97	0.97	0.93	0.95	0.93	0.88
Data	1.00	0.99	0.99	0.99	0.90	0.85	0.69
Decision	0.97	0.94	0.93	0.57	0.25	0.28	0.35
Process	0.99	0.99	0.98	0.96	0.92	0.86	0.69
Connection	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Average	0.99	0.98	0.98	0.91	0.83	0.82	0.76

Table 4.10: The serialization accuracy comparison on FC dataset with 5x10cv.

Delaye et al evaluate a Single Linkage Agglomerative Clustering (SLAC) based approach on FC dataset. They report an overall segmentation recall of 77.53% with stroke ordering, but they do not report accuracy per object type. When they perform clustering without using stroke ordering, they achieve a total recall of 64.12%. The MST scheme is similar to this setting, and it achieves a recall of 89.89% without stroke ordering.

	Carton	Lemaitre	Bresler	U.Bound	MST	TSP	TSPGA	MSTD	TSPD	TSPDGA	Closest
Terminator	0.72	0.70	0.88	0.92	0.91	0.91	0.91	0.90	0.87	0.83	0.82
Arrow	0.70	0.69	0.74	0.91	0.86	0.84	0.84	0.78	0.71	0.69	0.67
Data	0.81	0.80	0.89	0.94	0.93	0.92	0.92	0.92	0.83	0.78	0.63
Decision	0.81	0.67	0.74	0.87	0.82	0.81	0.80	0.46	0.21	0.23	0.29
Process	0.85	0.81	0.87	0.92	0.90	0.89	0.88	0.84	0.80	0.74	0.59
Connection	0.82	0.81	0.94	0.88	0.87	0.87	0.86	0.85	0.84	0.81	0.81
Average	0.79	0.75	0.84	0.90	0.88	0.87	0.87	0.79	0.71	0.68	0.64

Table 4.11: Recall comparison on FC dataset with 5x10cv.

Chapter 5

CONTRIBUTIONS

In this thesis, we presented a unified framework for stroke fragmentation and sketch recognition. Following sections discusses to what extent our work covers the requirements, and what makes our work unique.

5.1 Coverage of Requirements

The unified framework covers all aspects of the requirements, and meets them to a large extent.

1. **Accuracy:** The unified framework finds the optimal solution. Evaluation on several datasets shows that its accuracy either matches with the state of the art, or it outperforms them with a wide margin.

2. **Runtime Efficiency:** The unified framework uses dynamic programming for efficiency. This approach reduces the exponential time complexity of naïve algorithms to quadratic time. The unified framework is fast enough to allow real-time fragmentation/recognition while the user is drawing.

3. **Adaptability:** The matching cost function, which is used to compute the cost of a fragmentation/segmentation, is learned through supervised machine learning. Hence, the algorithm is capable of adapting to different primitive types with the inclusion of new features. Supervised learning aspect of our method allows us to adapt to the characteristics of different users and datasets. The annotation cost, which is the main disadvantage of supervised methods, is also minimal.

4. **Freedom on Sketching Style:** In contrast to the existing methods, the unified framework does not have any assumption about the user drawing style or the properties of the domain objects (e.g. the size of the objects and the number of primitives).

5. **Offline/Online Mode:** The framework supports the recognition of both offline and interspersed sketches, as well as online sketches. The accuracy difference between online and offline methods is acceptable due to loss of temporal information.

5.2 Unique Contributions

Following contributions make our work unique:

- We formulated both fragmentation and segmentation problem within a dynamic programming using an adaptive cost function. We proved theoretically that fragmentation and segmentation problems can be solved with a dynamic programming approach.

- We extended the total order based formulation of dynamic programming to the partial orders using a tree structure.

- We proposed a complete solution for the recognition of offline and interspersed sketches. We introduced the concept of spatial serialization to align offline and online recognition processes.

- We defined the coherence as the likelihood of two primitives being in the same object. We showed that we can generate a correct serialization based on pairwise coherence between primitives.

- We introduced PairIDM feature to estimate the coherence between primitive pairs. We showed that PairIDM based coherence model outperforms a distance based coherence model.

- We compared the advantages and disadvantages of several spatial serialization schemes in terms of accuracy and runtime. We showed that MST serialization scheme produces compatible accuracies with online methods, however TSP is well-suited when a balance between runtime and accuracy is required.

- Experimentally, we showed that our unified framework is either compatible with state-of-the art, or it surpasses the existing methods in terms of accuracy on several datasets.

5.3 Publications From Thesis

Tumen, R., Acer, M., & Sezgin, T. (2010). Feature Extraction and Classifier Combination for Image-based Sketch Recognition. In SBIM'10 (pp. 6370).

Tumen, R. S., & Sezgin, T. M. (2013). DPFrag: Trainable Stroke Fragmentation Based on Dynamic Programming. IEEE Computer Graphics and Applications, 33(5), 5967.

Tumen, R. S., & Sezgin, T. M. Efficient Sketch Recognition with Spatial Serialization. (In progress).

Chapter 6

FUTURE WORK

6.1 Object Model

The accuracy analysis points out that the *object model* as the weakest link in the recognition chain. We can improve the accuracy of the object model with an improvement in the following subjects:

6.1.1 Object/Non-object Discrimination

The separation of hypotheses into objects and non-objects is a fundamental problem of the segmentation. The existing features for isolated object recognition well-suited for discriminating object classes. However, they perform poorly in discrimination of objects and non-objects. For this reason, we need to design strong features for separating objects from non-objects.

6.1.2 Imbalanced Datasets

Most of the generated hypotheses are non-objects, and the datasets are strongly imbalanced. In this context, we also need to explore the problems of imbalanced training sets and outlier-detection. The validity classifier is designed to reject the outliers, but it requires further improvement.

6.1.3 Context in Recognition

We use the context features in coherence model, but we discard the context features in the object model. This is mainly due to fact that the high variation in context degrades the accuracy of the object model. If future studies may reduce the variation

in the context features, then, they can be embedded into the object model.

6.2 Hierarchical Segmentation

Serialization reduces the recognition complexity to polynomial time, but we can further reduce the complexity by partitioning the serialization tree into a forest. In a simple model, we can define a maximum size for a tree, and split the sketch into a set of trees using a coherence threshold. More complicated models must first discover higher level of abstractions for digital ink, then they can segment the input more efficiently and accurately.

6.3 Unsupervised / Semi-supervised / Active Learning

Supervised machine learning techniques require annotation of the training data. However, we must provide an abstraction from the internal recognition process for the application developer. Unsupervised learning does not require annotation of the training data, however, it results in lower accuracies than supervised learning. In addition, we cannot change the existing object categories. Semi-supervised / active learning methods may reduce the annotation to an acceptable level, but we need further studies about semi-supervised and active learning methods to use them in sketch recognition [Yanık and Sezgin, 2015].

6.4 Learning the Sketching Style

We know that users generally draw from left to right, and from top to bottom. They commonly draw objects sequentially, one after another. They also minimize the distance between sequential strokes spontaneously. However, a complete model of sketching is not a well-studied topic. The studies are generally focused on recognition [Sezgin and Davis, 2008]. However, drawing process itself also deserves attention. We can model the sequential drawing with spatial serialization, but we must also integrate other dynamic properties into the complete model. Using this model, it may be

possible to generate an order of primitives, which is compatible with a drawing style of a user.

Appendices

Appendix 1

GLOSSARY

Sketch: Informal and freehand drawings consisting of a predefined set of symbols.

Object: A set of symbols in a domain. For example, resistor, capacitor and transistor are objects of electrical circuit domain.

Stroke: A set of points sampled between pen-down and pen-up events.

Primitive: Simple geometric shapes such as lines and arcs that constitutes objects.

Stroke Fragmentation: The task of splitting a stroke into primitives.

Sibling: The primitives of the same object.

Interspersed Object: An object of which primitives do not follow each other in the temporal drawing order. Interspersing happens when the user starts to draw a new object before completing the current one.

Sketch Recognition: The segmentation and the classification of a sketch.

Sketch Segmentation: The task of grouping the primitives to form the domain objects.

Object Classification: The task of determining the object type.

Serialization: A total/partial order of primitives.

Correct Serialization: A serialization in which the objects are not interspersed.

Coherence: The likelihood of a primitive pair being siblings.

Appendix 2

ACCURACIES ON EC DATASET

	MST	TSP	TSPGA	MSTD	TSPD	TSPDGA	Closest
Resistor	0.99	0.98	0.98	1.00	0.97	0.91	0.95
Capacitor	1.00	0.98	0.99	0.30	0.88	0.84	0.81
Ground	0.98	0.98	0.98	0.89	0.96	0.96	0.74
Diode	0.93	0.88	0.87	0.75	0.94	0.82	0.79
BJT	0.96	0.74	0.74	0.93	0.52	0.41	0.36
JFET	0.80	0.47	0.41	0.53	0.24	0.19	0.06
Battery	1.00	1.00	0.99	0.23	0.87	0.70	0.71
Voltage Source	1.00	0.83	0.76	0.60	0.73	0.58	0.13
Current Source	0.94	1.00	0.96	0.85	0.88	0.78	0.40
AC Source	1.00	1.00	1.00	0.96	0.96	0.88	0.21
Average	0.96	0.89	0.87	0.70	0.80	0.71	0.52

Table B.1: Serialization Accuracy averaged over object instances. (5x10cv)

	Ouyang	U.Bound	MST	TSP	TSPGA	MSTD	TSPD	TSPDGA	Closest
Resistor	0.96	0.82	0.81	0.82	0.82	0.82	0.81	0.81	0.82
Capacitor	0.80	0.78	0.58	0.59	0.60	0.26	0.66	0.62	0.58
Ground	0.86	0.90	0.83	0.83	0.82	0.86	0.83	0.82	0.74
Diode	0.91	0.79	0.70	0.71	0.72	0.69	0.75	0.68	0.72
BJT	0.79	0.81	0.81	0.78	0.74	0.74	0.52	0.49	0.41
JFET	0.71	0.91	0.84	0.72	0.72	0.76	0.46	0.37	0.25
Battery	0.77	0.73	0.75	0.81	0.80	0.36	0.76	0.68	0.67
Voltage Source	0.92	0.95	0.88	0.89	0.88	0.86	0.86	0.82	0.32
Current Source	0.86	0.86	0.86	0.83	0.82	0.80	0.76	0.73	0.51
AC Source	0.82	0.86	0.86	0.87	0.87	0.85	0.86	0.84	0.35
Average	0.84	0.84	0.79	0.78	0.78	0.70	0.73	0.68	0.54

Table B.2: F1 score averaged over object instances.(5x10cv, 50% BB)

	Ouyang	U.Bound	MST	TSP	TSPGA	MSTD	TSPD	TSPDGA	Closest
Resistor	0.96	0.80	0.79	0.79	0.78	0.79	0.77	0.77	0.78
Capacitor	0.79	0.87	0.75	0.79	0.79	0.23	0.77	0.73	0.73
Ground	0.91	0.88	0.80	0.76	0.76	0.80	0.77	0.76	0.67
Diode	0.94	0.89	0.81	0.78	0.77	0.77	0.86	0.75	0.79
BJT	0.79	0.95	0.91	0.81	0.78	0.90	0.51	0.50	0.38
JFET	0.62	0.92	0.81	0.61	0.61	0.71	0.32	0.26	0.16
Battery	0.78	0.68	0.73	0.79	0.79	0.25	0.72	0.59	0.58
Voltage Source	0.87	0.92	0.85	0.85	0.84	0.79	0.78	0.73	0.20
Current Source	0.91	0.83	0.85	0.81	0.78	0.74	0.69	0.64	0.37
AC Source	0.94	0.81	0.81	0.85	0.85	0.83	0.84	0.79	0.21
Average	0.84	0.85	0.81	0.78	0.77	0.68	0.70	0.65	0.49

Table B.3: Recall averaged over object instances.(5x10cv, 50% BB)

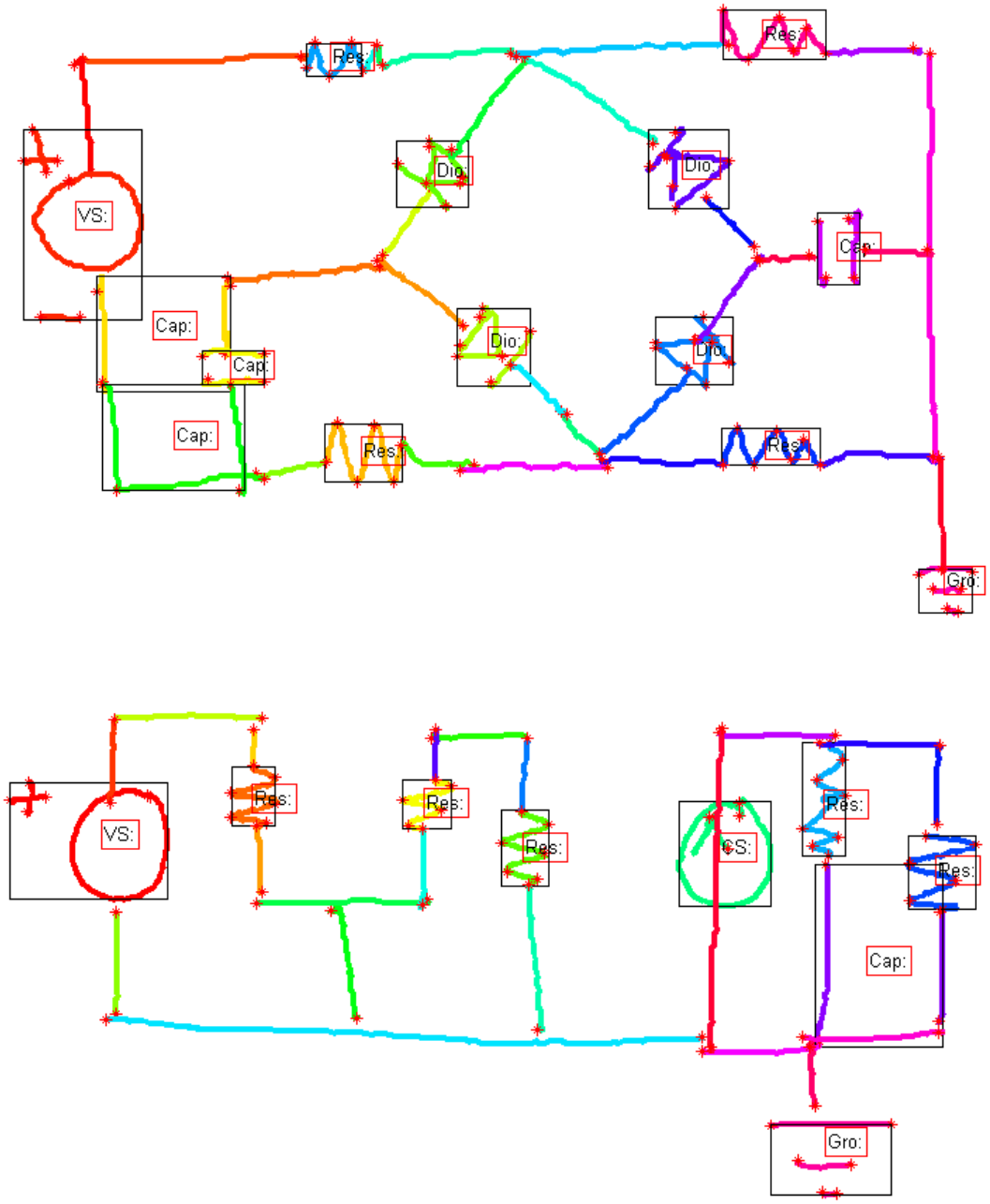


Figure B.1: The false detection of capacitors results in low precision. The context features may help reducing the false positives.

	Ouyang	U.Bound	MST	TSP	TSPGA	MSTD	TSPD	TSPDGA	Closest
Resistor	0.97	0.86	0.84	0.86	0.86	0.85	0.85	0.85	0.86
Capacitor	0.81	0.71	0.48	0.47	0.48	0.30	0.58	0.53	0.48
Ground	0.96	0.92	0.86	0.92	0.90	0.92	0.90	0.89	0.84
Diode	0.88	0.71	0.63	0.66	0.68	0.63	0.67	0.62	0.67
BJT	0.79	0.70	0.72	0.75	0.71	0.62	0.53	0.48	0.44
JFET	0.84	0.91	0.88	0.88	0.88	0.82	0.79	0.67	0.53
Battery	0.77	0.79	0.76	0.82	0.81	0.62	0.81	0.81	0.80
Voltage Source	0.98	0.98	0.91	0.93	0.91	0.94	0.95	0.93	0.85
Current Source	0.82	0.89	0.88	0.86	0.86	0.87	0.85	0.86	0.82
AC Source	0.73	0.91	0.91	0.88	0.90	0.88	0.89	0.91	0.97
Average	0.84	0.84	0.79	0.80	0.80	0.75	0.78	0.75	0.73

Table B.4: Precision averaged over object instances.(5x10cv, 50% BB)

Appendix 3

ACCURACIES ON FA DATASET

	U.Bound	MST	TSP	TSPGA	MSTD	TSPD	TSPDGA	Closest
State	1.00	1.00	1.00	1.00	1.00	0.99	0.99	0.99
Final State	1.00	1.00	1.00	1.00	0.89	0.31	0.33	0.33
Arrow	1.00	1.00	1.00	0.99	0.90	0.99	0.99	0.95
Average	1.00	1.00	1.00	1.00	0.93	0.77	0.77	0.76

Table C.1: Serialization Accuracy averaged over object instances. (5x10cv)

	U.Bound	MST	TSP	TSPGA	MSTD	TSPD	TSPDGA	Closest
State	0.92	0.93	0.92	0.92	0.88	0.73	0.73	0.73
Final State	0.91	0.93	0.91	0.91	0.86	0.40	0.43	0.41
Arrow	0.93	0.93	0.92	0.92	0.81	0.89	0.89	0.86
Average	0.92	0.93	0.92	0.92	0.85	0.67	0.68	0.67

Table C.2: F1 score averaged over object instances. (5x10cv, AoN)

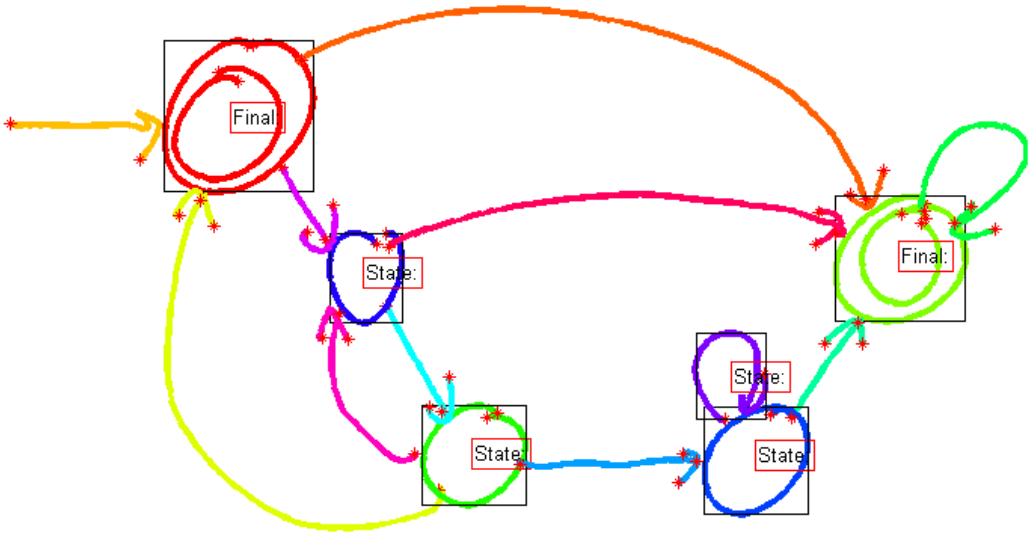


Figure C.1: Highly curved arrows are confused with state symbol.

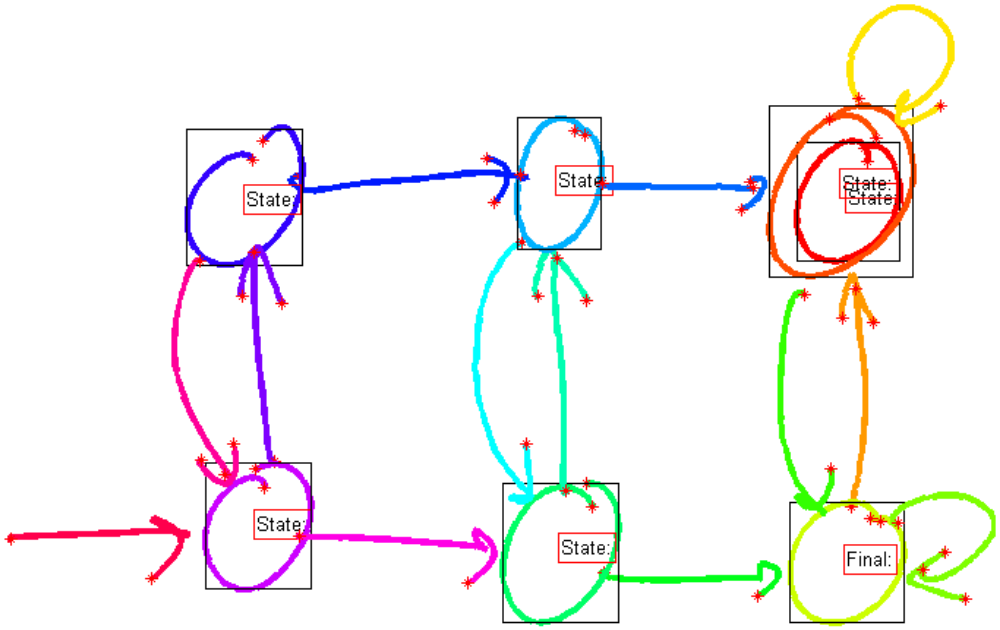


Figure C.2: Final state symbols are recognized as two different state symbols.

	Bresler	U.Bound	MST	TSP	TSPGA	MSTD	TSPD	TSPDGA	Closest
State	0.95	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98
Final State	0.96	0.85	0.87	0.85	0.85	0.76	0.25	0.27	0.26
Arrow	0.89	0.95	0.94	0.94	0.94	0.86	0.92	0.92	0.89
Average	0.94	0.93	0.93	0.92	0.92	0.87	0.72	0.72	0.71

Table C.3: Recall averaged over object instances. (5x10cv, AoN)

	U.Bound	MST	TSP	TSPGA	MSTD	TSPD	TSPDGA	Closest
State	0.87	0.89	0.87	0.87	0.80	0.58	0.59	0.58
Final State	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.97
Arrow	0.91	0.92	0.91	0.91	0.76	0.87	0.86	0.83
Average	0.92	0.93	0.92	0.92	0.85	0.81	0.81	0.80

Table C.4: Precision averaged over object instances. (5x10cv, AoN)

Appendix 4

ACCURACIES ON FC DATASET

	MST	TSP	TSPGA	MSTD	TSPD	TSPDGA	Closest
Terminator	0.99	1.00	1.00	1.00	0.98	0.98	0.94
Arrow	0.98	0.97	0.97	0.93	0.95	0.93	0.88
Data	1.00	0.99	0.99	0.99	0.90	0.85	0.69
Decision	0.97	0.94	0.93	0.57	0.25	0.28	0.35
Process	0.99	0.99	0.98	0.96	0.92	0.86	0.69
Connection	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Average	0.99	0.98	0.98	0.91	0.83	0.82	0.76

Table D.1: Serialization accuracy averaged over object instances. (5x10cv)

	U.Bound	MST	TSP	TSPGA	MSTD	TSPD	TSPDGA	Closest
Terminator	0.92	0.92	0.91	0.91	0.91	0.89	0.87	0.86
Arrow	0.89	0.83	0.81	0.80	0.72	0.65	0.62	0.58
Data	0.95	0.94	0.93	0.92	0.93	0.88	0.84	0.72
Decision	0.87	0.82	0.81	0.81	0.53	0.30	0.33	0.40
Process	0.92	0.91	0.90	0.90	0.86	0.83	0.79	0.66
Connection	0.89	0.89	0.89	0.88	0.88	0.87	0.85	0.85
Average	0.91	0.88	0.87	0.87	0.81	0.73	0.72	0.68

Table D.2: F1 score averaged over object instances. (5x10cv, AoN)

	Carton	Lemaitre	Bresler	U.Bound	MST	TSP	TSPGA	MSTD	TSPD	TSPDGA	Closest
Terminator	0.72	0.70	0.88	0.92	0.91	0.91	0.91	0.90	0.87	0.83	0.82
Arrow	0.70	0.69	0.74	0.91	0.86	0.84	0.84	0.78	0.71	0.69	0.67
Data	0.81	0.80	0.89	0.94	0.93	0.92	0.92	0.92	0.83	0.78	0.63
Decision	0.81	0.67	0.74	0.87	0.82	0.81	0.80	0.46	0.21	0.23	0.29
Process	0.85	0.81	0.87	0.92	0.90	0.89	0.88	0.84	0.80	0.74	0.59
Connection	0.82	0.81	0.94	0.88	0.87	0.87	0.86	0.85	0.84	0.81	0.81
Average	0.79	0.75	0.84	0.90	0.88	0.87	0.87	0.79	0.71	0.68	0.64

Table D.3: Recall averaged over object instances. (5x10cv, AoN)

	U.Bound	MST	TSP	TSPGA	MSTD	TSPD	TSPDGA	Closest
Terminator	0.93	0.92	0.92	0.91	0.92	0.91	0.91	0.91
Arrow	0.86	0.81	0.78	0.77	0.67	0.60	0.56	0.51
Data	0.96	0.94	0.94	0.93	0.95	0.92	0.92	0.84
Decision	0.87	0.82	0.82	0.82	0.64	0.50	0.57	0.63
Process	0.93	0.91	0.91	0.91	0.89	0.87	0.84	0.73
Connection	0.91	0.91	0.91	0.91	0.90	0.90	0.90	0.90
Average	0.91	0.89	0.88	0.88	0.83	0.78	0.78	0.75

Table D.4: Precision averaged over object instances. (5x10cv, AoN)

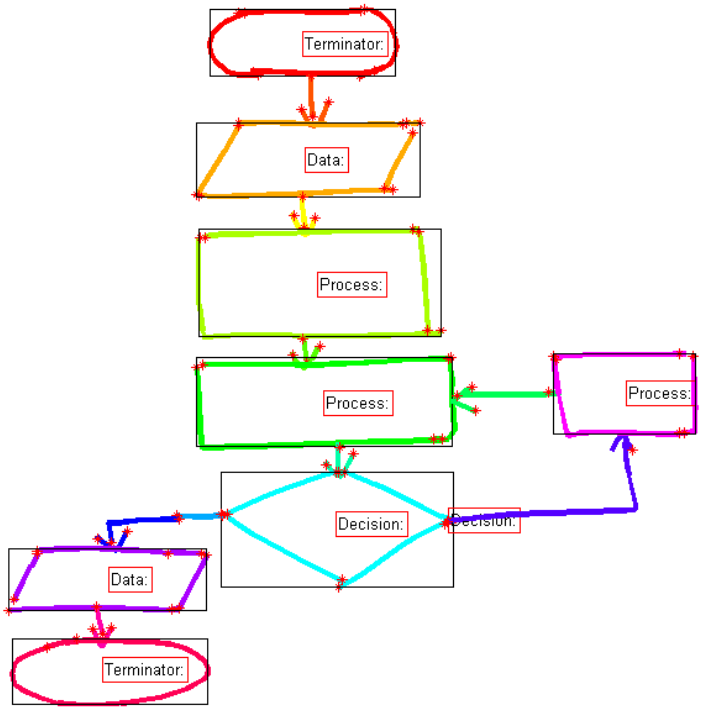


Figure D.1: The high variation in arrow symbols results in mis-recognition in some cases.

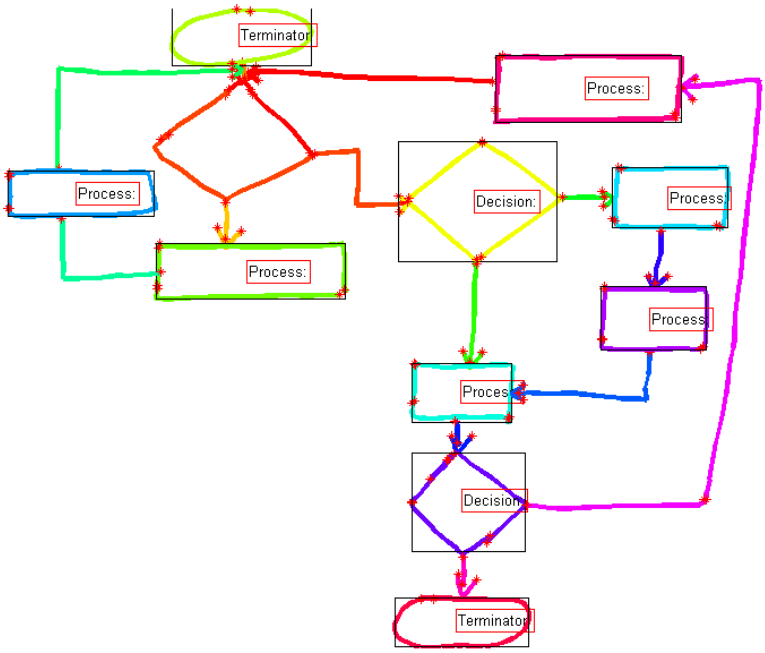


Figure D.2: Under-fragmented primitives results in under-segmentation of decision symbols.

BIBLIOGRAPHY

[Adi et al., 2010] Adi, D. I. S., Shamsuddin, S. M. b., and Hashim, S. Z. M. (2010). Nurbs curve approximation using particle swarm optimization. *Computer Graphics, Imaging and Visualization, International Conference on*, pages 73–79.

[Alvarado and Davis, 2006] Alvarado, C. and Davis, R. (2006). Dynamically constructed bayes nets for multi-domain sketch understanding. In *ACM SIGGRAPH 2006 Courses*, page 32. ACM.

[Arandjelović and Sezgin, 2011] Arandjelović, R. and Sezgin, T. M. (2011). Sketch recognition by fusion of temporal and image-based features. *Pattern Recognition*, 44(6):1225–1234.

[Awal et al., 2011] Awal, A.-M., Feng, G., Mouchere, H., and Viard-Gaudin, C. (2011). First experiments on a new online handwritten flowchart database. In *IS&T/SPIE Electronic Imaging*, pages 78740A–78740A. International Society for Optics and Photonics.

[Bresler et al., 2014] Bresler, M., Van Phan, T., Prusa, D., Nakagawa, M., and Hlavác, V. (2014). Recognition system for on-line sketched diagrams. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 563–568. IEEE.

[Carton et al., 2013] Carton, C., Lemaitre, A., and Couasnon, B. (2013). Fusion of statistical and structural information for flowchart recognition. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 1210–1214. IEEE.

[Davis, 2007] Davis, R. (2007). Magic paper: sketch-understanding research. *IEEE Computer*, 40(9):34–41.

[Delaye and Lee, 2015] Delaye, A. and Lee, K. (2015). A flexible framework for online document segmentation by pairwise stroke distance learning. *Pattern Recognition*, 48(4):1193–1206.

[Demšar, 2006] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30.

[Deufemia and Risi, 2005] Deufemia, V. and Risi, M. (2005). A dynamic stroke segmentation technique for sketched symbol recognition. In *Pattern Recognition and Image Analysis*, pages 335–357.

[Dietterich, 1998] Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923.

[Feng et al., 2009] Feng, G., Viard-Gaudin, C., and Sun, Z. (2009). On-line hand-drawn electric circuit diagram recognition using 2d dynamic programming. *Pattern Recognition*, 42(12):3215–3223.

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

[Hammond and Davis, 2006] Hammond, T. and Davis, R. (2006). Ladder: A language to describe drawing, display, and editing in sketch recognition. In *ACM SIGGRAPH 2006 Courses*, page 27. ACM.

[Herold and Stahovich, 2011a] Herold, J. and Stahovich, T. F. (2011a). Classyseg: a machine learning approach to automatic stroke segmentation. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*, pages 109–116.

[Herold and Stahovich, 2011b] Herold, J. and Stahovich, T. F. (2011b). Speedseg: A technique for segmenting pen strokes using pen speed. In *Computers & Graphics*, volume 35, pages 250 – 264.

[Igarashi and Zeleznik, 2007] Igarashi, T. and Zeleznik, B. (2007). Guest editors' introduction: Sketch-based interaction. *Computer Graphics and Applications, IEEE*, 27(1):26–27.

[Kara and Stahovich, 2004] Kara, L. B. and Stahovich, T. F. (2004). Hierarchical parsing and recognition of hand-sketched diagrams. In *Proceedings of the 17th annual ACM symposium on UIST*, pages 13–22. ACM.

[Kolesnikov, 2012] Kolesnikov, A. (2012). Segmentation and multi-model approximation of digital curves. *Pattern Recognition Letters*, pages –.

[Lemaitre et al., 2013] Lemaitre, A., Mouchere, H., Camillerapp, J., and Coüasnon, B. (2013). Interest of syntactic knowledge for on-line flowchart recognition. In *Graphics Recognition. New Trends and Challenges*, pages 89–98. Springer.

[Oltmans, 2007] Oltmans, M. (2007). *Envisioning sketch recognition: a local feature based approach to recognizing informal sketches*. PhD thesis, MIT.

[Oltmans et al., 2004] Oltmans, M., Alvarado, C., and Davis, R. (2004). Etcha sketches: Lessons learned from collecting sketch data. *Making Pen-Based Interaction Intelligent and Natural*, pages 134–140.

[Orbay and Kara, 2011] Orbay, G. and Kara, L. B. (2011). Beautification of design sketches using trainable stroke clustering and curve fitting. In *IEEE Transactions on Visualization and Computer Graphics*, volume 17, pages 694–708.

[Ouyang, 2009] Ouyang, T. (2009). A visual approach to sketched symbol recognition. *Proceedings of the 21st international joint conference on artificial intelligence*.

[Ouyang, 2012] Ouyang, T. (2012). *Understanding freehand diagrams: combining appearance and context for multi-domain sketch recognition.* Ph.d., MIT.

[Ouyang and Davis, 2011] Ouyang, T. Y. and Davis, R. (2011). Chemink: a natural real-time recognition system for chemical drawings. In *Proceedings of the 16th international conference on Intelligent user interfaces*, pages 267–276. ACM.

[Sezgin and Davis, 2008] Sezgin, T. and Davis, R. (2008). Sketch recognition in interspersed drawings using time-based graphical models. *Computers & Graphics*, 32(5):500–510.

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

[Stahovich et al., 2014] Stahovich, T. F., Peterson, E. J., and Lin, H. (2014). An efficient, classification-based approach for grouping pen strokes into objects. *Computers & Graphics*, 42:14–30.

[Sutherland, 1964] Sutherland, I. E. (1964). Sketch pad a man-machine graphical communication system. In *Proceedings of the SHARE design automation workshop*, pages 6–329. ACM.

[Tax, 2015] Tax, D. (2015). Ddtools, the data description toolbox for matlab. version 2.1.2.

[Tversky, 2002] Tversky, B. (2002). What do sketches say about thinking. *2002 AAAI Spring Symposium, Sketch Understanding.*

[Wolin and Hammond, 2008] Wolin, A. and Hammond, T. (2008). Shortstraw: A simple and effective corner finder for polylines. In *Eurographics 5th Annual Workshop on Sketch-Based Interfaces and Modeling*, pages 33–40.

[Wolin et al., 2009] Wolin, A., Paulson, B., and Hammond, T. (2009). Sort, merge, repeat: an algorithm for effectively finding corners in hand-sketched strokes. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, pages 93–99.

[Wu et al., 1999] Wu, L., Oviatt, S. L., and Cohen, P. R. (1999). Multimodal integration-a statistical view. *Multimedia, IEEE Transactions on*, 1(4):334–341.

[Xiong and Jr., 2010] Xiong, Y. and Jr., J. J. L. (2010). A shortstraw-based algorithm for corner finding in sketch-based interfaces. In *Computers & Graphics*, volume 34, pages 513 – 527.

[Yanık and Sezgin, 2015] Yanık, E. and Sezgin, T. M. (2015). Active learning for sketch recognition. *Computers & Graphics*.

[Yin et al., 2005] Yin, L., Yajie, Y., and Wenyin, L. (2005). Online segmentation of freehand stroke by dynamic programming. In *Proceedings of the Eighth International Conference on Document Analysis and Recognition*, pages 197–201.

[Yu, 2003] Yu, B. (2003). Recognition of freehand sketches using mean shift. In *Proceedings of the 8th international conference on Intelligent user interfaces*, IUI '03, pages 204–210.