UNIVERSITY OF CALIFORNIA
RIVERSIDE

An Efficient and General-Purpose Technique for Grouping Hand-Drawn Pen Strokes into Objects

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Mechanical Engineering

by

Eric Jeffrey Peterson

December 2010

Dissertation Committee:

Dr. Thomas F. Stahovich, Chairperson
Dr. V. Sudararajan
Dr. Christine Alvarado

Acknowledgments

I would like to thank everyone who has made possible my completion of graduate school. Specifically I would like to thank my research advisor, Tom Stahovich, who has helped me expand and craft my abilities. Thank you for challenging me and never letting me settle for less than excellence. I would also like to thank the rest of my dissertation committee, Christine Alvarado and V. Sundararajan, for helping guide me and my research in meaningful directions. Additionally, I would like to thank my lab-mates, they have provided an endless amount of help and entertainment: WeeSan, Ryan, Rumi, Tyler, Jim, Jack, Josiah, Hank, Ehsan, Matt, John, and Seth. Special thanks go to Diana for helping to edit this dissertation.

I could not have accomplished this without the support of my family, especially my parents: Gordon and Myra. They have shown me how to work hard and enjoy life. To my grandparents, for helping me get through school and for raising such wonderful families, as well as my older sister Amber and her husband Dallas, and my younger sister Josselyn, thank you.

Perhaps most of all, I thank my wife Christine, for loving me and having patience as I spent so many long nights working on my research and this dissertation.

This dissertation is dedicated to my loving wife Christine, who has stuck with me through graduate school, and has supported my decisions every step of the way. I love you Babe!

ABSTRACT OF THE DISSERTATION

An Efficient and General-Purpose Technique for Grouping Hand-Drawn Pen Strokes into Objects

by

Eric Jeffrey Peterson

Doctor of Philosophy, Graduate Program in Mechanical Engineering
University of California, Riverside, December 2010
Dr. Thomas F. Stahovich, Chairperson

Engineers use sketches in the early phases of design because their expressiveness and ease of creation facilitate creativity and efficient communication. Our goal is to build software that leverages these strengths and enables natural sketch-based human-computer interaction. Specifically, our work is focused on creating algorithms that group hand-drawn strokes into individual objects so that they can be recognized. Grouping strokes is a difficult problem. Many previous approaches have required the user to manually group the strokes. Others have used a search process, resulting in a computational cost that rises exponentially with the number of strokes in the sketch. In this dissertation we present a novel method for grouping strokes into objects based on a two-step algorithm that has a polynomial computational cost. In the first step, strokes are classified according to the type of object to which they belong, thus helping to create artificial separation between objects. In the second step, a pairwise classifier groups strokes of the same class into individual objects. Both steps rely on general machine learning techniques which seamlessly integrate spatial and temporal information, and which can be extended to new domains with no hand-coding. Our single-stroke classifier is the first in literature to perform multi-way classification to facilitate efficient grouping, and it performs as well as or better than previous classifiers on text vs. non-text classification. Our grouping algorithm correctly groups between 84% and 91% of the ink in diagrams from four different domains, with between 61% and 82% of objects being perfectly clustered. Our method runs in $O(n^2)$ time, where n is the number of points in the sketch. Real-world performance is improved with a conservative filter to eliminate consideration of distant strokes, and computation occurs incrementally as the sketch is constructed. Even without the filter, the computation for a large sketch containing over 700 strokes took less than 12% of the time required to draw the sketch. Experimental evaluation of our technique has shown it to be accurate and effective in four domains.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Engineers often use sketches in the beginning stages of design because they are an effective way to communicate and express ideas due to their ease of creation and flexibility. Our overarching goal is to allow people to interact with a computer by sketching in the same way that they can convey ideas to each other using sketches. Imagine a tool that allows the user to design a logic circuit just by sketching it out, rather than interacting with a complicated and clunky computer interface. Or an engineering-mechanics tutorial program that provides immediate and relevant feedback as the student draws a free-body diagram and solves the equilibrium equations. Both of these tools require the computer to interpret the hand-drawn sketch. Automatic interpretation requires the program to recognize the individual objects in the sketch. However, before the object can be recognized, the strokes that comprise it must be located. Determining which strokes belong to the same object is comparatively easy in a simple sketched equation, like in Figure 1.1, because there is clear separation between the objects. However, grouping strokes into objects is a much more complicated task for a connected diagram, like the digital circuit sketch shown in Figure 1.2, where there is often no separation between objects.

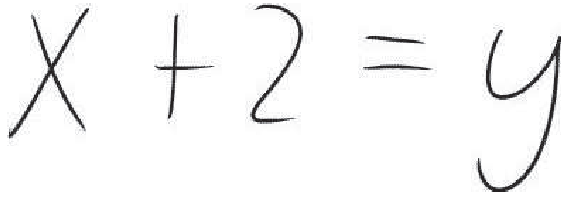


Figure 1.1: A simple sketched equation. The individual characters are separated from each other, making it relatively easy to locate the objects.

It is difficult for a program to know that a group of strokes form a meaningful shape without recognizing it, however the strokes must be grouped before they can be recognized. For example, in Figure 1.1, previous methods would not know whether a cluster containing the two strokes in the “x” represents a meaningful object without recognizing it, but those two strokes must first be clustered. This dependency between grouping and recognition is a chicken-and-egg problem that can have a very large search space for possible groups of strokes. Most previous work has focused on generating a large number of possible groupings (hypotheses), and using recognition to evaluate them [1, 40]. One main drawback to this type of approach is that the number of possible groupings to be evaluated grows exponentially with the number of strokes in the sketch, leading to a system that is too slow for real-time interaction. Other attempts try to reduce the size of the search space by limiting the way people draw, such as requiring one object to be finished before the next begins [10, 38]. While some researchers have tried to lessen the burden of these restrictions [37], their methods still do not leave the user to draw in an unconfined manner. Another approach to addressing this challenge has been to create algorithms that exploit features specific to a given domain [20, 21], but these are difficult to extend to new problems. We aim to create a stroke grouping method that satisfies each of the following requirements:

1. **Accuracy** - It should provide accurate groupings of strokes.
2. **Speed** - It should be fast enough to provide real-time interaction between the user and the computer.
3. **Extensibility** - It should be free of attributes that restrict it to a particular domain, thereby allowing the method to be used in new applications.
4. **Drawing Flexibility** - It should allow the user to draw freely and naturally.

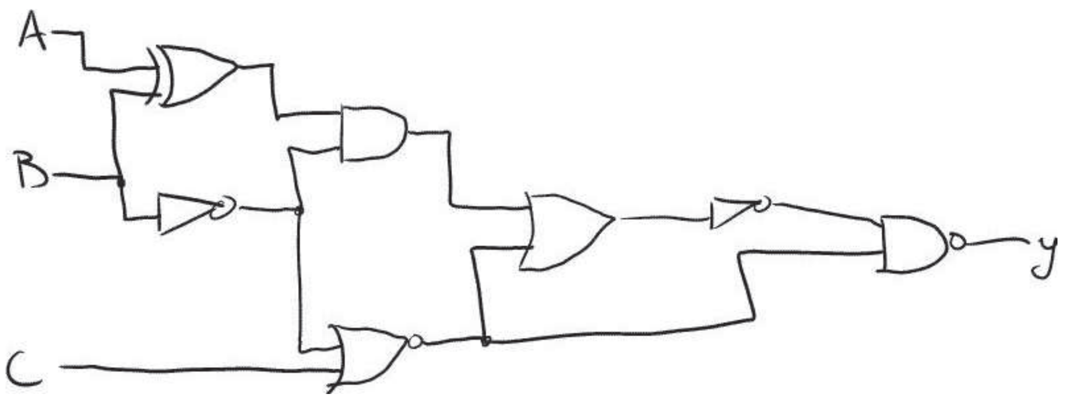


Figure 1.2: A sketch of a digital circuit. There is no clear boundary between adjacent objects in the sketch

1.1 Approach

We have developed a novel two-step approach to locate objects in hand-drawn diagrams. The overall process is shown in Figure 1.3. The first step separates the strokes within a sketch into classes, the second step groups the strokes of a given class into objects. Figure 1.4 shows the single-stroke classification result for the digital circuit in Figure 1.2 – sets of gate, wire, and text strokes. While it may appear that the strokes in Figure 1.4 are already grouped together, a second step is required for the program to determine which strokes belong together.

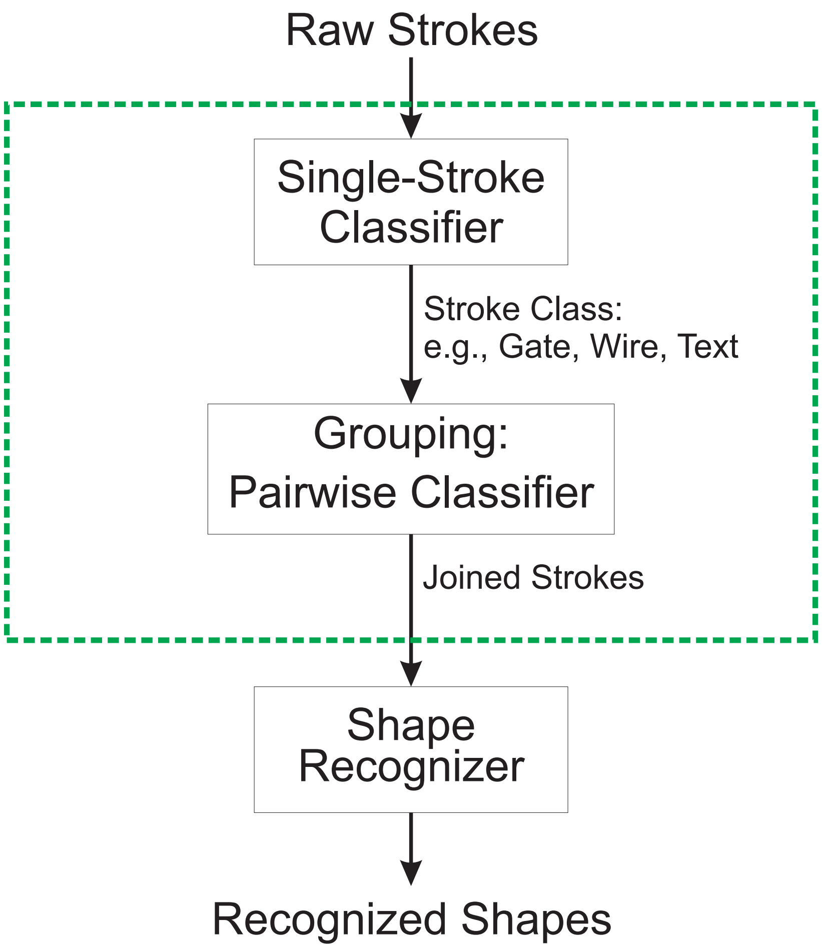
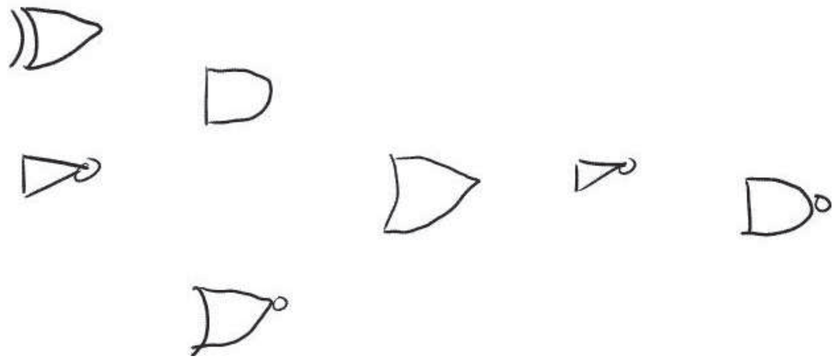


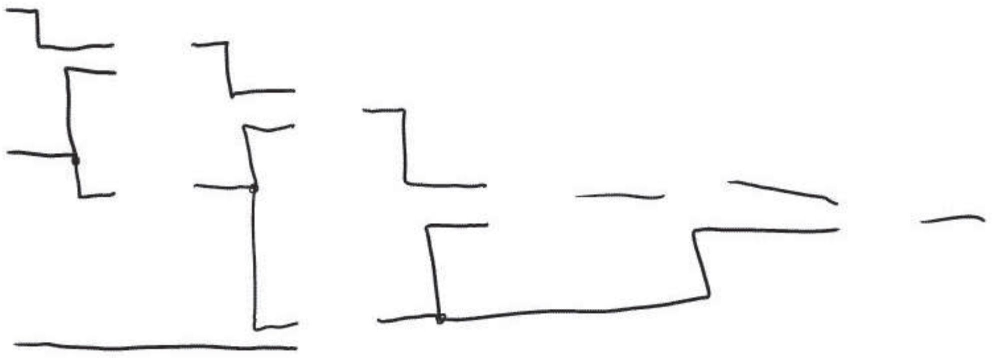
Figure 1.3: A flow chart detailing the sketch recognition process. Our two-step approach is outlined by the dashed box.

We use a classifier learned from labeled training examples to separate the strokes into classes. Learned classifiers can quickly and accurately identify a stroke's class without necessitating hand-coding – a driving characteristic in our decision to use them. To identify a stroke's class, the classifier uses a number of features based on properties of the stroke, such as shape, size, location, drawing dynamics, and geometric and temporal relationships with nearby strokes. The features are general, allowing the method to be extended to new domains. Classifying strokes into different classes creates spatial and temporal separation between individual objects, as shown in Figure 1.4. This separation allows the second step of our algorithm to more accurately and efficiently determine which strokes belong together.

The second step uses sets of pairwise classifiers, one for each class of strokes, to identify pairs of strokes that are likely to be part of the same object, thus forming two-stroke clusters. The pairwise classifiers use features based on spatial and temporal distances between the two strokes. The resulting two-stroke clusters are combined with other clusters that have a stroke in common, a process we refer to as chaining. The resulting clusters represent the individual objects in a sketch. For example, in Figure 1.2, after the gate strokes in the digital circuit have been classified, a pairwise classifier determines which gate strokes belong together. Joined pairs of gate strokes are chained to form complete gates.

We have evaluated our approach on sketches from four different domains: digital circuits, family tree diagrams, and two sets of solutions to statics problems (a field of engineering mechanics). The first statics set is composed of sketches that include free-body diagrams and equilibrium equations. The second set is artificially created from the first by removing the equation related strokes. Full descriptions of these domains


(a) The strokes that are part of gate objects.


(b) The strokes that are part of wire objects.

(c) The strokes that are part of (text) label objects for circuit inputs and outputs.

Figure 1.4: Stroke separation in the digital circuit sketch shown in Figure 1.2.

can be found in Chapter 4.

1.2 Contributions

In this work we examine stroke grouping in freely-drawn diagrammatic sketches. We make four significant contributions to stroke grouping and the larger task of sketch understanding in this dissertation.

1. **Separation for Grouping:** We have shown the importance of single-stroke classification for stroke grouping.

2. **Grouping Methods Survey:** We present the first in-depth comparison of general-purpose grouping techniques. We evaluate two novel approaches that do not rely on recognition.

3. **Single-Stroke Classification:** We present a method for classifying single strokes that is of comparable or better accuracy than other methods we tested for text versus non-text classification. Our approach is also extended to multiple classes, three for digital circuits, three for family tree diagrams, and four for both sets of statics solutions. It can easily be used for more classes, although we have found these class distinctions provide good separation for grouping. We provide an extension to previous sets of features, mainly in the area of contextual information. We find these features to be some of the most valuable for accurate classification.

4. **Pairwise Classification:** We have developed two classification techniques for identifying *pairs* of strokes that belong together. These classification techniques are typically more efficient than previous approaches because they do not attempt to determine whether stroke groups of arbitrary size form a meaningful shape –

an inherently exponential process.

1.3 Outline

This dissertation is organized as follows: Chapter 2 details our approach to single-stroke classification (stroke separation). Chapter 3 describes our algorithms for grouping strokes of a given class. Chapter 4 presents the details of the data used for evaluating our algorithms. Chapter 5 presents results from evaluations of our single-stroke classification and grouping methods. Chapter 6 discusses these results. Chapter 7 discusses work related to the research presented in this dissertation. Finally, Chapter 8 presents our conclusions.

Chapter 2

Single-Stroke Classification

2.1 Introduction

Grouping freely-drawn pen strokes into salient objects is a very difficult part of sketch understanding. One of the most challenging aspects of grouping is that strokes may be close together or even intersecting, yet not belong to the same object. Our insight is to use a classifier to create artificial separation between objects by classifying strokes belonging to different types of objects, thus facilitating efficient stroke grouping.

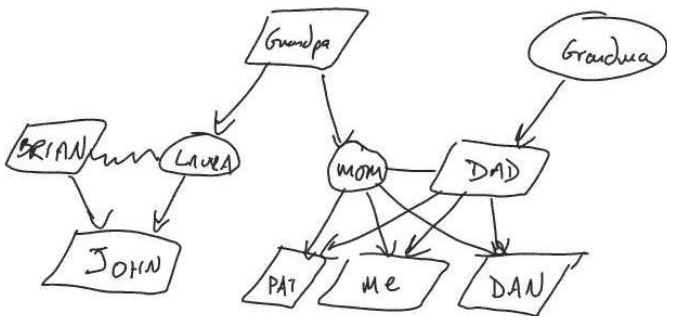
We present here our approach for generalized stroke classification. For the digital circuit shown in Figure 1.2, our approach classifies the strokes into gates, wires, and labels. This classification creates separation between objects, as shown in Figures 1.4(a), 1.4(b), and 1.4(c). Similarly, for a family tree diagram sketch (such as Figure 2.1(a)), the classifier determines whether a stroke is part of a person, link, or text object, as shown in Figures 2.1(b), 2.1(c), and 2.1(d), respectively. We have also applied this classification technique to both of our sets of statics solution sketches. A complete statics solution sketch is shown in Figure 2.2(a). Here, the strokes are classified as part of body, arrow, text, or “other” objects, as shown in Figures 2.2(b), 2.2(c), 2.2(d), and 2.2(e), respectively. As these figures illustrate, classifying strokes in this way effectively transforms a sketch with many interconnected objects into a number of smaller, spatially-isolated sub-sketches, each of which is easier to group.

2.2 Methods

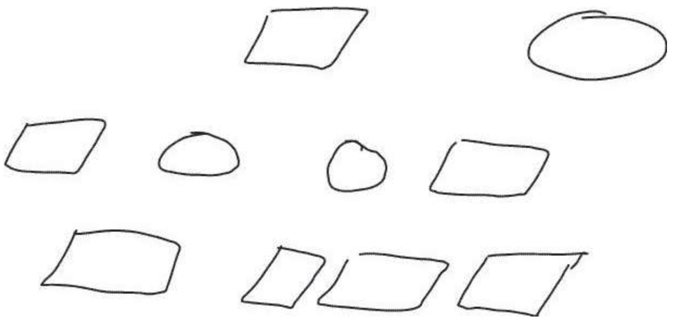
Our primary goal in designing our single-stroke classifier is to create a technique that is effective at creating separation between sketched objects. Achieving this requires careful choice of the classification algorithm, and careful design of the set of classes and the set of features used for classification. Different domains may require different features and different stroke classes.

2.2.1 Classes

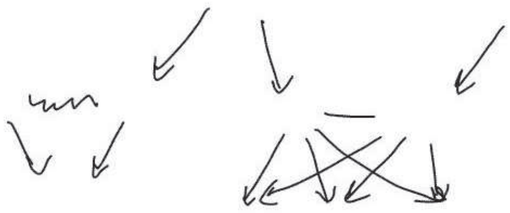
When designing the set of classes, it is best to choose a set such that separation between objects is maximized without splitting individual objects into multiple classes. If classification can be done with perfect accuracy, more classes are generally better. In practice, as the distinctions between classes becomes finer, accuracy decreases. For example, in digital circuits, it may be more difficult to accurately distinguish between fine-grained shapes such as wires, labels, *and* gates, *or* gates, *nand* gates, etc. than to distinguish between gates in general, wires, and labels. In developing our approach, we had to balance this tradeoff between granularity and accuracy.



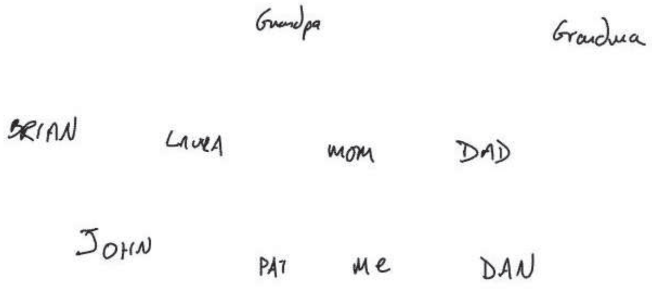
(a) A family tree diagram sketch.



(b) The strokes that are part of people objects.

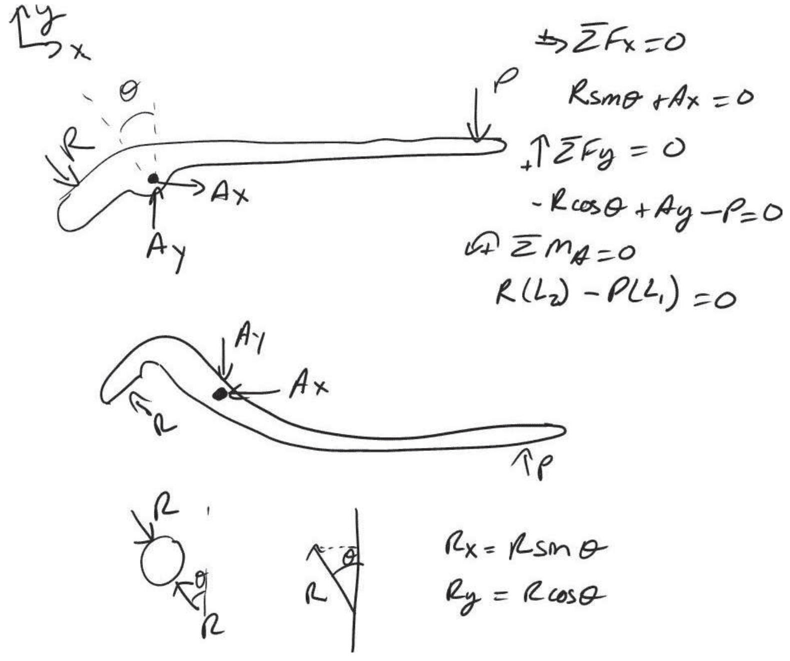


(c) The strokes that are part of link objects.

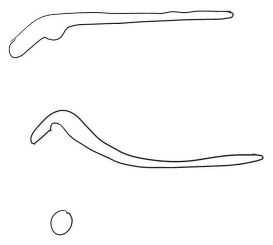


(d) The strokes that are part of text objects.

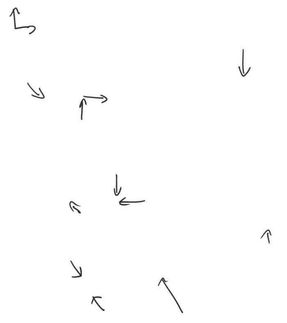
Figure 2.1: Stroke separation in a family tree diagram sketch.



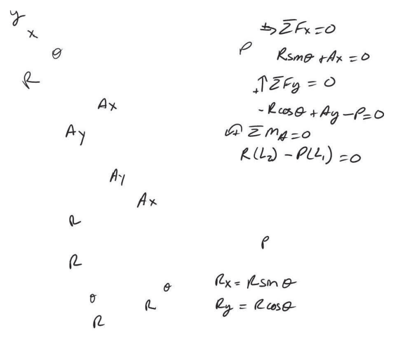
(a) A complete statics solution sketch, with free-body diagrams and equilibrium equations.



(b) The strokes that are part of body objects.



(c) The strokes that are part of arrow objects.



(d) The strokes that are part of text objects.

(e) The strokes in the sketch that don't fit into other categories, and are part of "other" objects.

Figure 2.2: Stroke separation in a statics solution sketch.

We divided each of the domains into a number of classes, as shown in Table 2.1. For digital circuits, we found the best classes to be: gates, wires, and labels. For family trees, we found the best classes to be: people, text, and links. For statics solutions without equations, we found the best classes to be: bodies, arrows, labels, and “other.” For complete statics solutions, we found the best classes to be: bodies, arrows, text, and “other.” Each of the domains is explained in more detail in Chapter 4.

Domain	**Classes**	**Shapes**
Digital Circuits	Gates	AND, OR, NAND, NOR, NOT, NOTBUBBLE, XOR, XNOR, LabelBox, Other
	Wires	Wire
	Labels	Label
Family Trees	Persons	Male, Female
	Text	Text
	Links	Marriage, Divorce, ChildLink, Other
Statics Solutions	Bodies	Body, Box, Triangle
	Arrows	ForceArrow, MomentArrow, PointerArrow, DimensionArrow, CoordSystem, DoubleShaftArrow, ForceEqnArrow, MomentEqnArrow, OtherArrow
	Text	Character
	Other	LeaderLine, Arc, Point, Divider, AngleSquare, Angle, OtherGeometry, Ellipsis, Other
Statics (NoEqn)	Bodies	Body, Box, Triangle
	Arrows	ForceArrow, MomentArrow, PointerArrow, DimensionArrow, CoordSystem, DoubleShaftArrow, ForceEqnArrow, MomentEqnArrow, OtherArrow
	Text	ForceLabel, MomentLabel, AngleLabel, DimensionLabel, LocationLabel, AxisLabel, OtherLabel, OtherEquation, OtherText
	Other	LeaderLine, Arc, Point, Divider, AngleSquare, Angle, OtherGeometry, Ellipsis, Other

Table 2.1: A list of domains for which we perform single-stroke classification. The *Class* column indicates the general type of shape that a stroke belongs to; this is the label used for training the single-stroke classifier. The *Shape* column enumerates the different shapes that make up the class.

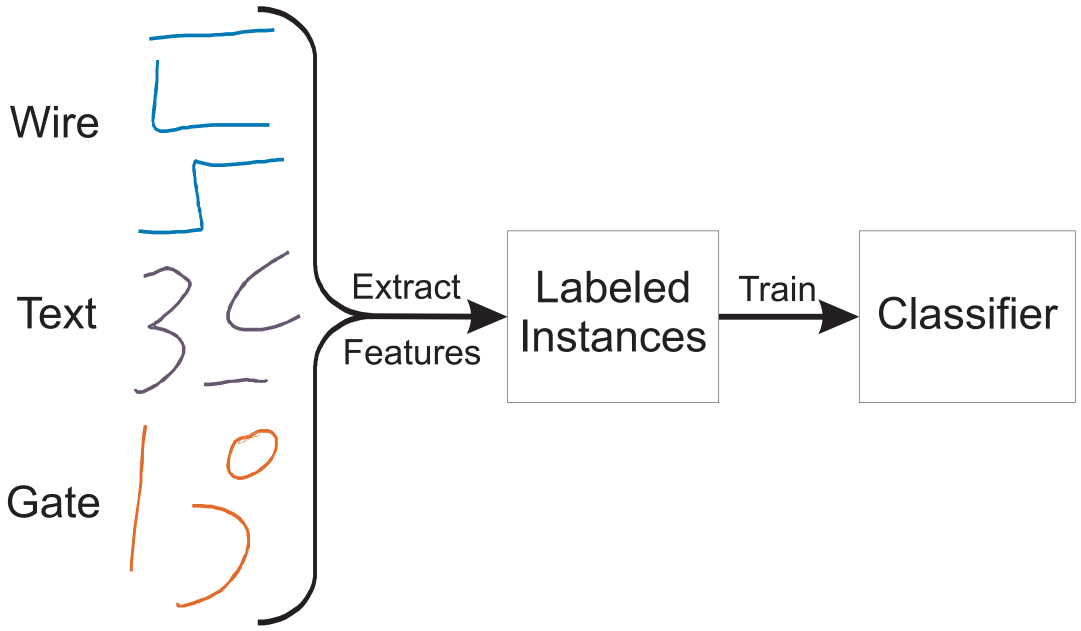
2.2.2 Classifier

Our classification task is well suited to an inductive classifier, trained with labeled examples as shown in Figure 2.3(a). Once the classifier has been trained, it is used to classify unknown instances, as shown in Figure 2.3(b). A large number of classification techniques have been developed. We evaluated several of them, and found that AdaBoosted decision trees (adaptive boosting with a c4.5 decision tree base classifier) had the best performance. We also evaluated multi-layer perceptrons (MLP), c4.5 decision trees (without AdaBoost), AdaBoosted decision stumps, and AdaBoosted MLPs. While the AdaBoosted decision trees performed best, ordinary decision trees, such as c4.5, can be trained much more quickly and are easier to implement. We valued accuracy more than training speed, so we use AdaBoosted decision trees, but different classifiers may be better suited for some situations.

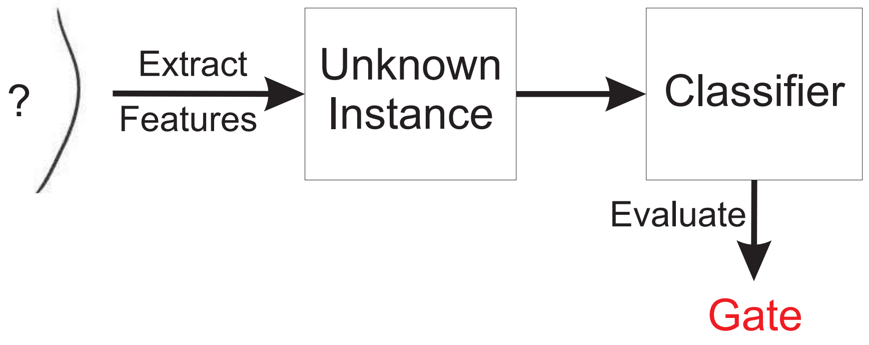
We used WEKA's[1] implementation of these algorithms to quickly train and evaluate different classifiers [11]. Our classifier's specifics are: AdaBoostM1 with 10 iterations, a seed of 1, no resampling, and a weight threshold of 100. The base classifier specifics are: J48 decision tree (an implementation of c4.5), pruned, with a confidence value of 0.25, and the minimum number of instances in a leaf is 2.[2]

[1] `http://www.cs.waikato.ac.nz/ml/weka/`

[2] Our code is written in C#, which uses the .NET framework. We are able to integrate WEKA (written in Java) functionality using the IKVM.NET package, which enables Java and .NET interoperability. The IKVM.NET project converts a Java .jar file into a .NET recognizable dynamic linked library (.dll). More information about IKVM.NET can be found at `http:www.ikvm.net`.



(a) Training a Classifier from labeled examples.



(b) Classification using a trained classifier.

Figure 2.3: Features are extracted from labeled examples to train the single-stroke classifier. This learned classifier can then be used to classify unknown instances. We use an AdaBoosted decision tree classifier.

2.2.3 Features

While the choice of specific classifier can affect the efficacy of the method, the choice of classifier *inputs*, called features, can have a more significant impact on performance. Much of the motivation behind feature creation is an attempt to capture certain perceptual aspects of drawing that a human uses to understand a sketch. The features are a general way of capturing the answer to the question, “What makes these strokes different?” We have found that both the shape of a stroke and the context in which it appears are critical to accurate classification. To this end we extract a total of 27 features that characterize the stroke’s size and location, its shape, the drawing kinematics, and its relationships with other strokes in the sketch. Presented here is our complete set of features used for single-stroke classification (listed in Table 2.2). Some of these features can be found in other work, such as that of Patel, *et al.* [30]. Our work, however, includes novel features and outperforms their system. No individual feature is an identifier for a specific stroke class, but rather the set of features work together to identify the stroke’s class.

The first important property of a stroke is its size, which is represented by four features. *Bounding Box Width*, *Height* and *Area* are properties of the minimum, coordinate-aligned bounding box of the stroke. *Arc Length* is the total length of the stroke measured as a sum of the distance between consecutive points:

$$ArcLength = \sum_{i=2}^{N} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \tag{2.1}$$

where N is the number of points in the stroke. These features are normalized by their average values in the sketch, allowing the classifier to learn the importance of relative stroke size during training.

One insight not captured by Patel *et al.*’s features is that, in many domains, particular kinds of objects often appear in preferred locations on the drawing canvas (defined here as the bounding box of the entire sketch). For example, diagram strokes

Category	Feature Name	Description
Size	Bounding Box Width	Width of the minimum bounding box
	Bounding Box Height	Height of the minimum bounding box
	Bounding Box Area	Area of the minimum bounding box
	Arc Length	Total length of the stroke (Eqn 2.1)
Location	Distance to Left / Right	Minimum distance between the stroke and the closer of the left or right edge of the canvas
	Distance to Top / Bottom	Minimum distance between the stroke and the closer of the top or bottom edge of the canvas
Shape	End Point Ratio	Degree to which the stroke forms a closed path with itself (Eqn 2.2)
	Self Enclosing	Binary version of End Point Ratio
	Self Intersections	Number of times the stroke crosses itself
	Sum of the (signed) Curvature	Sum of each curvature value along the stroke (Eqn 2.4)
	Sum of the Abs Value of the Curvature	Sum of the absolute value of each curvature value along the stroke (Eqn 2.5)
	Sum of the Squared Curvature	Sum of the square of each curvature value along the stroke (Eqn 2.6)
	Sum of the Sqrt of Curvature	Sum of the square root of each curvature value along the stroke (Eqn 2.7)
	Ink Density	Compactness of the stroke (Eqn 2.8)
Drawing Kinematics	Average Pen Speed	Average Speed while drawing
	Maximum Pen Speed	Maximum instantaneous pen speed
	Minimum Pen Speed	Minimum instantaneous pen speed
	Difference between Max and Min	Maximum pen speed minus minimum pen speed
	Time to Draw Stroke	Time from pen down to pen up
Geometric Relations	'LL' Intersections	Count of endpoint-to-endpoint intersections with other strokes
	'XX' Intersections	Count of midpoint-to-midpoint intersections with other strokes
	'XL' Intersections	Count of midpoint-to-endpoint intersections with other strokes
	'LX' Intersections	Count of endpoint-to-midpoint intersections with other strokes
	Closed Path	Binary feature - does the stroke help form a closed path via 'LL' intersections
	Inside Path	Binary feature - is the stroke inside a Closed Path
Temporal Relations	Time to Previous	Time between the end of the previous stroke and the beginning of the current stroke
	Time to Next	Time between the end of the current stroke and the beginning of the next stroke

Table 2.2: Features used for single-stroke classification.

may be drawn in the center of the canvas, with text annotations near the periphery. This phenomenon is captured by two positional features. *Distance to Left/Right* is the minimum distance between the stroke and the closer of the left or right edge of the canvas, divided by the width of the canvas. As an example, consider a stroke that is, at its closest point, 1,000 units from the left edge of the canvas and 3,000 units from the right edge, and the canvas is 10,000 units wide. In this case, the value for *Distance to Left/Right* would be $1,000/10,000$ or 0.100. *Distance to Top/Bottom* is defined analogously, and represents the location relative to the top or bottom of the canvas.

Eight features describe the shape of a stroke. The first three describe its topological properties. *EndPtRatio* (Eqn 2.2) measures the degree to which the stroke forms a closed path with itself. It is defined as the Euclidean distance between the endpoints of the stroke divided by the arc length:

$$EndPtRatio = \frac{\sqrt{(x_N - x_1)^2 + (y_N - y_1)^2}}{ArcLength} \tag{2.2}$$

EndPtRatio can be particularly effective at distinguishing between some classes of strokes. For example, many diagrammatic strokes are closed shapes such as circles, ellipses, or squares, whereas most link type strokes (such as arrows or wires) typically have endpoints that are far away from each other. *EndPtRatio* ranges between 0.0 and 1.0. A value of 0.0 indicates that the stroke's endpoints are coincident, while a value of 1.0 indicates that the stroke is a straight line. *Self Enclosing* is a binary form of the *EndPtRatio*. If *EndPtRatio* is less than a threshold, T, the value of *Self Enclosing* is one, otherwise it is zero. The threshold T is the only hard coded threshold used in the single-stroke classification task, and we use a value of 0.15. *Self Intersections* is the number of times the stroke intersects (crosses) itself.

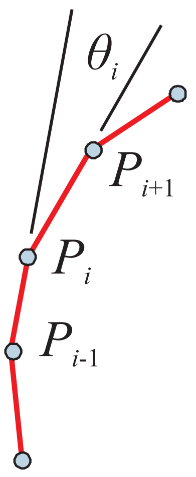


Figure 2.4: Curvature at a point, θ, is the angle between the line segments connecting it to the previous and next points.

The next four shape features characterize the stroke's curvature, as illustrated in Figure 2.4. The curvature, θ_i, at point i is defined as the angle between the segment connecting point $i-1$ to point i, and the segment connecting point i to point $i+1$:

$$\theta_i = arctan\frac{\delta x_i \delta y_{i-1} + \delta x_{i-1}\delta y_i}{\delta x_i \delta x_{i-1} + \delta y_i \delta y_{i-1}} \tag{2.3}$$

where $\delta x_i = x_{i+1} - x_i$ and $\delta y_i = y_{i+1} - y_i$. The four curvature features are obtained by summing various functions of the curvature value at each point along the stroke (see Eqns 2.4-2.7). *Sum of the (signed) Curvature* represents the total turning angle of the stroke, where turns in one direction cancel turns in the other. For example, if the stroke turns 360^o clockwise and then 360^o counterclockwise, *Sum of the (signed) Curvature* is zero, indicating that there is no net change in the direction of the stroke. *Sum of the Absolute Value of the Curvature* provides a measure of how much the curve "wiggles," or deviates from a straight line. In the previous example, for instance, *Sum of the Absolute Value of the Curvature* is 720^o, even though there is no net change in direction. *Sum of the Squared Curvature* emphasizes corners, or points of high curvature. Conversely, *Sum of the Square Root of Curvature* emphasizes points of low curvature. The first three of these features are from [35], while the last is of our own design.

$$SumSignedCurvature = \sum_{i=2}^{N-1} \theta_i \tag{2.4}$$

$$SumAbsCurvature = \sum_{i=2}^{N-1} |\theta_i| \tag{2.5}$$

$$SumSquareCurvature = \sum_{i=2}^{N-1} {\theta_i}^2 \tag{2.6}$$

$$SumSqrtCurvature = \sum_{i=2}^{N-1} \sqrt{|\theta_i|} \tag{2.7}$$

Ink Density measures the compactness of the stroke. In previous work this was particularly useful in helping to distinguish wires from components in analog circuits [10]. It is defined as the ratio of the square of the arc length to the area of the minimum coordinate-aligned bounding box:

$$InkDensity = \frac{ArcLength^2}{BoundingBoxArea} \tag{2.8}$$

Arc length is squared so that it scales in the same way as bounding box area.

Pen speed can provide important information about the intended class of a stroke because people tend to draw familiar shapes differently than novel ones. For example, a person usually writes their name faster than they draw the shape of an airplane or other complicated object. To capture this, the drawing kinematics is represented in terms of four speed-based features: the *Average Pen Speed*, the *Maximum Pen Speed* (instantaneous), *Minimum Pen Speed* (instantaneous), and the *Difference Between Maximum and Minimum* instantaneous pen speeds. Pen speed is often near zero at the two endpoints of a stroke, so when computing the minimum, a few points at each end are ignored. Each speed-based feature value is normalized by the average stroke pen speed in the sketch. The final kinematic feature is the *Time to Draw* the stroke.

The remainder of the features characterizes the geometric and temporal relationships the stroke has with other strokes in the sketch. The first four of these features measures the number of different types of intersections the stroke has with other strokes (as illustrated in Figure 2.5): endpoint-to-endpoint ("LL"), midpoint-to-midpoint ("XX"), midpoint-to-endpoint ("XL"), and endpoint-to-midpoint ("LX"). We have found that this distinction between intersections involving endpoints and midpoints, which is not reported elsewhere in literature, is important for accurate stroke classification.

Because sketches are typically messy, we use a distance tolerance to catch cases where strokes nearly intersect. In effect, the strokes are extended at each end by a small amount, as illustrated in Figure 2.6. A simple linear tolerance is too generous for long

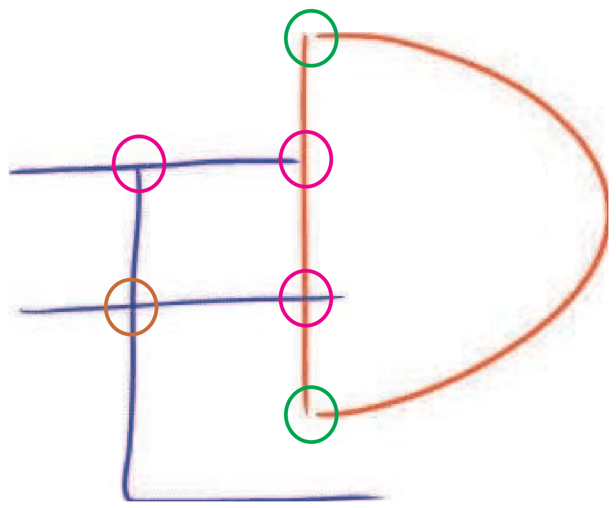


Figure 2.5: Shows the four types of intersections: endpoint-to-endpoint (green), midpoint-to-midpoint (brown), endpoint-to-midpoint (pink), and midpoint-to-endpoint (pink). The distinction between the last two is based on the stroke for which the feature is being computed. For instance, the upper blue horizontal line "tees" into the vertical red line. This is considered an endpoint-to-midpoint intersection for the blue stroke, while it is a midpoint-to-endpoint intersection for the red stroke.

strokes, and too tight for short ones. Instead, our tolerance is derived from the sketch average arc length, L_{avg}, as follows:

$$L_{tol} = min(L_{avg}, \frac{L_i + L_{avg}}{2}) * T \tag{2.9}$$

where L_i is the arc length of the stroke to be extended and T is the same threshold defined earlier, with a value of 0.15. This formula produces a proportionally larger tolerance for short strokes, and a proportionally smaller tolerance for long ones. Before extending a stroke, the small "hooks" at the endpoints are removed, using the algorithm described in [24], so that the direction at the endpoints is meaningful. If the intersection point lies within distance L_{tol} of the actual endpoint of the stroke, it is considered an endpoint intersection. Otherwise, it is considered a midpoint intersection. The case in which two extended strokes do not actually intersect, but their endpoints are within a distance L_{tol} of one another, is still considered an endpoint intersection.

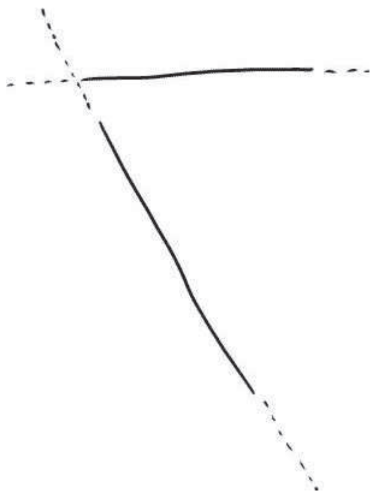


Figure 2.6: Strokes are extended to allow nearly touching strokes to be counted as intersecting. A dynamic length is used, which extends short strokes proportionally longer and long strokes proportionally shorter according to Eqn 2.9.

Our feature set includes two other novel features that characterize higher-level geometric relationships. The binary *Closed Path* feature indicates whether or not the stroke belongs to some set of strokes that connect to each other via *'LL'* intersections to form a closed path. The binary *Inside Path* feature indicates whether or not the stroke is inside the coordinate-aligned minimum bounding box of some closed path. Using a bounding box to test for *Inside Path* can result in false positives, but is inexpensive and has worked adequately for our purposes.

The final two features for single-stroke classification capture temporal relationships. *Time to Previous* is the elapsed time between the end of the previous stroke and the start of the current one. *Time to Next* is defined analogously.

While we have a total of 27 features to characterize the individual strokes, not all of them provide the same amount of information. We have found that the most important features vary from domain to domain; an analysis and list of ranked importance is presented in Section 5.1.4. Each of these features provides evidence for the classifier; no single feature should be considered specific to a given class. For instance, while many gate strokes are part of a *Closed Path*, this feature, by itself, is not an effective gate "recognizer."

Chapter 3

Grouping

3.1 Introduction

Classifying the individual pen strokes, as described in the previous section, reduces the complexity of stroke grouping by decomposing the problem into smaller, easier problems – one for each class. However, even for the strokes in a single class, grouping is a nontrivial task. Our approach uses pairwise classifiers to identify pairs of strokes that belong together. After these pairs have been identified, a "chainer" groups joined pairs that have a stroke in common to form clusters of arbitrary size. For example, the stroke pairs formed by the six strokes in Figure 3.1 are classified and then chained, as illustrated in Figure 3.2. Here, the classifier identifies that stroke pairs AB (strokes A and B), BC, DE, DF, and EF should be joined, while all other pairs should **not** be joined. The chainer then forms two clusters. The first contains strokes A, B, and C because the joined pairs AB and BC have stroke B in common. The second cluster contains strokes D, E, and F because the joined pairs DE, DF, and EF have common strokes.

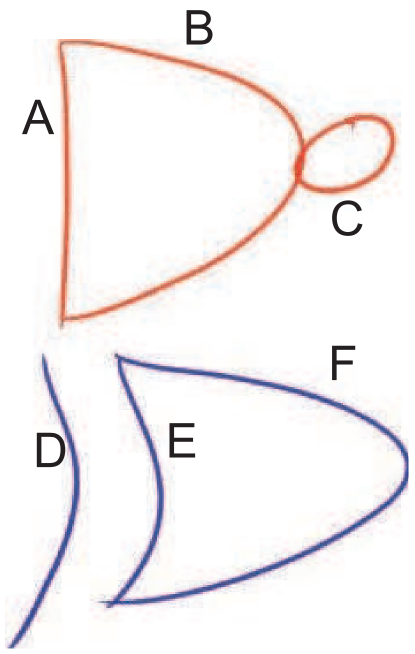


Figure 3.1: Two different gate objects, each composed of three strokes.

We have implemented and tested two different types of pairwise classifiers in an attempt to find a suitable algorithm. The *Thresholded Pairwise Classifier* (TPC) is based on simple linear thresholds, while the *Inductive Pairwise Classifier* (IPC) is an inductive learning technique. Both our TPC and IPC can use two join classes: *Join* and *NoJoin*. Our IPC can also use three join classes, as described in Section 3.3.2. In this chapter we present both classifiers, while Chapter 5 presents our evaluation of them and discusses their performance.

Most previous grouping methods evaluated clusters with an arbitrary number of strokes. Because the groups were of arbitrary size, the number of possible groups grew exponentially with the number of strokes in the sketch. For example, for the strokes

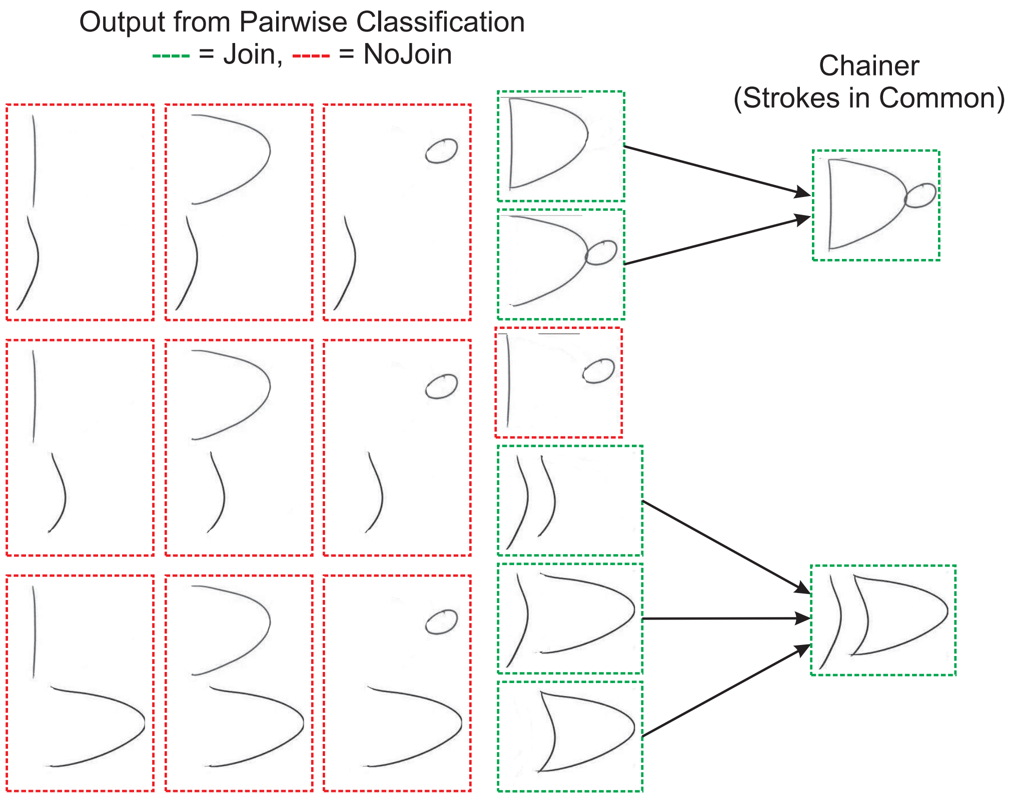

Output from Pairwise Classification
---- = Join, ---- = NoJoin
Chainer
(Strokes in Common)

Figure 3.2: Each pair of strokes in Figure 3.1 is classified to be *Join* or *NoJoin*. The chainer then clusters stroke pairs classified as *Join* which have a stroke in common.

in Figure 3.1, to determine the best set of clusters, a naïve (brute force) grouping approach would need to evaluate all single-stroke clusters, all two-stroke clusters, and so on until all strokes in the sketch are considered to be part of the same cluster. With these six strokes, a total of 63 clusters would need to be evaluated, while there are only two objects. For a sketch with 40 strokes, over one trillion clusters would need to be evaluated. The number of possible clusters for a sketch of size m is $2^m - 1$. Because this brute force approach quickly becomes unmanageable, most previous methods have restricted the maximum cluster size and the maximum possible distance between the strokes. Despite this, these types of algorithms are still exponentially expensive with respect to the number of strokes. Using pairwise classification significantly reduces our computational complexity and running time.

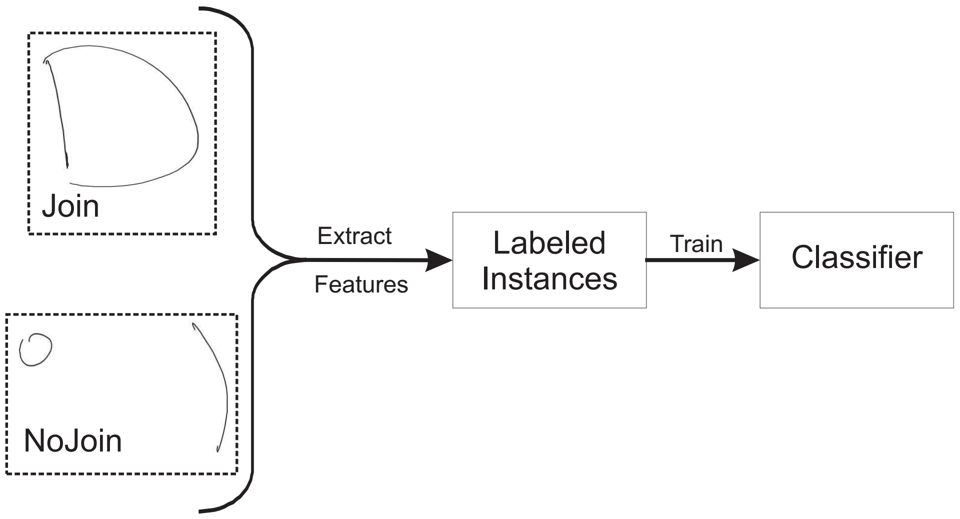
3.2 Thresholded Pairwise Classifier

Our *Thresholded Pairwise Classifier* is based on a simple conjunction of distance and time thresholds. A pair of strokes is joined if the minimum distance between them (d_{min} in Figure 3.4(a)) is less than or equal to a threshold T_{JD}, **and** the elapsed time between them is less than or equal to a time threshold T_{JT}. The values for T_{JD} and T_{JT} are learned from the training data. As discussed in Chapter 5, we evaluated this approach in a user-independent fashion, thus the parameter sets used for a particular user were learned from the others.

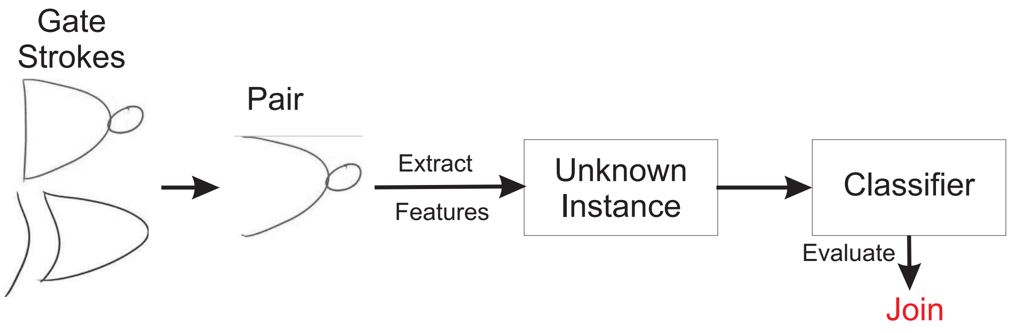
3.3 Inductive Pairwise Classifier

Our *Inductive Pairwise Classifier* considers a richer set of features for characterizing the pair of strokes than the *Thresholded Pairwise Classifier.* These features describe spatial and temporal relations between the strokes. Figure 3.3 illustrates how the classifier is trained and used for classification.

As with the single-stroke classifier, we explored a variety of classification techniques, including decision trees, multi-layer perceptrons, and AdaBoosted decision trees. Based on the experimental results, we chose to classify stroke pairs using an AdaBoosted decision tree. We found that they had higher accuracy than other classification techniques. We again use WEKA's implementation, with AdaBoostM1 and J48 decision trees, with the same parameters as for single-stroke classification, as described in sec-



(a) Training a Classifier from labeled examples.



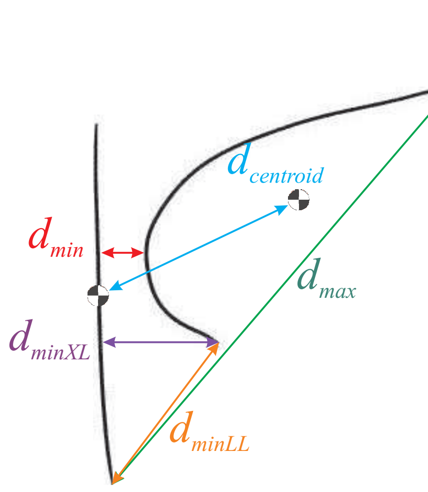
(b) Pairwise classification using a trained classifier.

Figure 3.3: Labeled examples are used to train the pairwise classifier. It can then be used to determine whether an unknown pair of strokes belongs together.

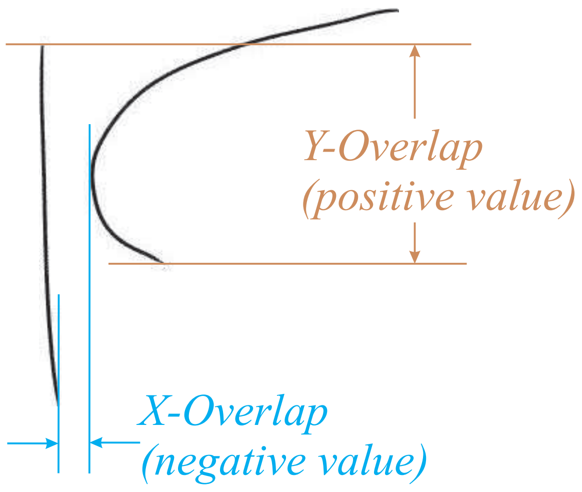
tion 2.2.2.

3.3.1 Features

The pairwise features are different from those used by the single-stroke classifier, and consist of both temporal and spatial measures. The temporal feature, *Time Gap*, is the time between the end of the first stroke and the beginning of the second. The spatial features include a variety of distance measures, overlap measures, perceptual



(a) Pairwise Features: Distance



(b) Pairwise Features: Overlap

Figure 3.4: Pairwise distance and overlap features are computed between the two strokes.

information, and distance ratios. The complete list of features is presented in Table 3.1.

Our distance measures are illustrated in Figure 3.4. The first is the *Minimum Distance* (d_{min}) between the strokes, calculated on a pointwise basis. Similarly, the pointwise *Maximum Distance* (d_{max}) is the farthest distance between a point on one stroke and a point on the other. The *Centroid Distance* ($d_{centroid}$) provides a measure of the distance between the "centers" of the strokes. *Minimum Endpoint-to-Endpoint Distance* (d_{minLL}) is the minimum value of the four inter-stroke endpoint distances. *Minimum Endpoint-to-Anypoint Distance* (d_{minXL}) is the minimum distance between an endpoint of one stroke and any point in the other stroke. *X-overlap* and *Y-overlap* are the length of the intersection between the projections of the two strokes onto the x-axis and y-axis, respectively. Their values are negative if the projections do not intersect. Each distance and overlap feature is normalized by the length of the sketch's bounding box diagonal, mapping all distance values to the range of 0.0 to 1.0 and all overlap values to the range of -1.0 to 1.0, thus making these feature values scale-independent.

The next two features, $Closeness_{Large}$ and $Closeness_{Small}$, compare the proximity of the strokes to each other with their proximity to other strokes. If the pair is comprised of $Stroke_i$ and $Stroke_j$, $Closeness_{Stroke_i}$ compares the pair's minimum distance (d_{min}) to the minimum distance between $Stroke_i$ and any stroke of the same class, computed as:

$$Closeness_{Stroke_i} = \frac{\underset{Stroke_a \in Class_c}{min}(d_{min}(Stroke_i, Stroke_a)) + k}{d_{min}(Stroke_i, Stroke_j) + k} \tag{3.1}$$

where k is a constant offset to avoid division by zero[1], $Closeness_{Stroke_j}$ is defined analogously for the second stroke in the pair. The closeness values range from 0.0 to 1.0. A value close to 1.0 indicates that the stroke is close to the other stroke in the pair compared to other strokes of the same class. This usually indicates that the pair should be joined. $Closeness_{Large}$ is defined as the larger of $Closeness_{Stroke_i}$ and $Closeness_{Stroke_j}$ values. $Closeness_{Small}$ is the smaller of the two values. This is intended to aid the classifier in learning by giving the two closeness values distinct meaning. The value of this distinction is verified in the analysis of feature importance in Section 5.2.8.

$Ratio_{LL}$ and $Ratio_{XL}$ give an indication of whether or not the strokes are closest to each other at their endpoints or elsewhere. $Ratio_{LL}$ is the ratio of the pair's minimum distance d_{min} and the minimum distance between the strokes' endpoints (d_{minLL}):

$$Ratio_{LL} = \frac{d_{min}(Stroke_i, Stroke_j) + k}{d_{minLL}(Stroke_i, Stroke_j) + k} \tag{3.2}$$

[1]We use a value of 10,000 himetrics for k. Himetric units are a measure of length, and one himetric is equivalent to ten micrometers.

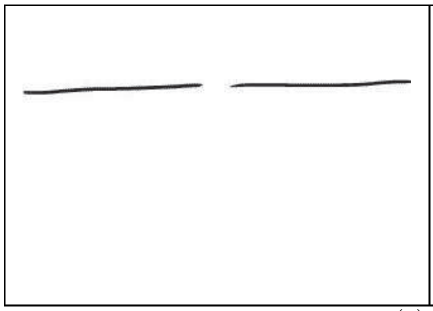
where k is again a constant offset. $Ratio_{XL}$ is computed similarly, except that the denominator uses the distance d_{minXL} rather than d_{minLL}. $Ratio_{LL}$ and $Ratio_{XL}$ have values in the range of 0.0 to 1.0. The combination of these two values helps characterize where the strokes are closest to each other. For example, in Figure 3.5(a), the two strokes are closest at their endpoints, and both $Ratio_{LL}$ and $Ratio_{XL}$ are equal to 1.0. In Figure 3.5(b), one stroke "tees" into the middle of the other, thus $Ratio_{LL}$ is 0.67 and $Ratio_{XL}$ is 1.0. In Figure 3.5(c), the strokes cross at their midpoints, and $Ratio_{LL}$ is 0.59 and $Ratio_{XL}$ is 0.67.

The final feature used for pairwise classification is a Boolean value indicating whether or not the two strokes are part of the *Same Closed Path*; these paths are the same as those used for single-stroke classification.

The computational cost for the pairwise features is largely determined by the pointwise distance calculations. Thus, computation is an $O(n^2)$ process, where n is the number of points in the sketch. The computational cost of the entire grouping process is determined by the pairwise feature computation. The features can be computed incrementally as each new stroke is drawn, so after the last stroke is drawn, very little computation is needed to complete the groups. See Section 5.3 for running time results.

3.3.2 Stroke Pair Labels

We use two classes for the *Thresholded Pairwise Classifier*: *Join* and *NoJoin*. However, we have found that these two classes are not adequate for the *Inductive Pairwise Classifier* (IPC). These two classes can create confusion for the IPC because pairs

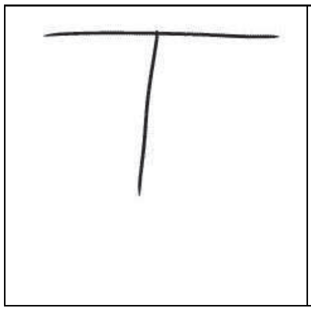


$$d_{min} = 1,000$$
$$d_{minLL} = 1,000$$
$$d_{minXL} = 1,000$$

$$Ratio_{LL} = \frac{1{,}000+10{,}000}{1{,}000+10{,}000} = 1.0$$

$$Ratio_{XL} = \frac{1{,}000+10{,}000}{1{,}000+10{,}000} = 1.0$$

(a)

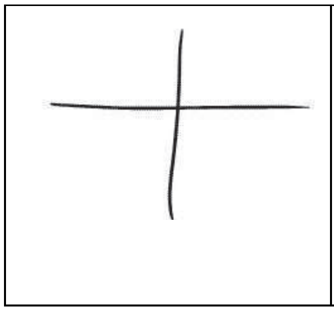


$$d_{min} = 0$$
$$d_{minLL} = 5,000$$
$$d_{minXL} = 0$$

$$Ratio_{LL} = \frac{0+10{,}000}{5{,}000+10{,}000} = 0.67$$

$$Ratio_{XL} = \frac{0+10{,}000}{0+10{,}000} = 1.0$$

(b)



$$d_{min} = 0$$
$$d_{minLL} = 7,071$$
$$d_{minXL} = 5,000$$

$$Ratio_{LL} = \frac{0+10{,}000}{7{,}071+10{,}000} = 0.59$$

$$Ratio_{XL} = \frac{0+10{,}000}{5{,}000+10{,}000} = 0.67$$

(c)

Figure 3.5: Examples of pairwise ratio features being computed. $Ratio_{LL}$ is computed according to Equation 3.2, with $k = 10,000$, and both $Stroke_i$ and $Stroke_j$ have arc lengths of 10,000. $Ratio_{XL}$ is computed analogously.

of strokes that are far apart can belong to the same cluster, while pairs of strokes that are close can belong to different clusters, making it difficult for the classifier to learn the difference between these cases. For example, in Figure 3.1, strokes A and C are farther apart than strokes A and D, despite the fact that A and C are part of the same cluster, while A and D are not. As a remedy, we consider three classes of stroke pairs: *NearJoin*, *FarJoin*, and *NoJoin*. As the name suggests, *NearJoin* pairs contain strokes that are

Category	**Feature Name**	**Description**
Distance	d_{min}	Minimum inter-stroke pointwise distance
	d_{max}	Maximum inter-stroke pointwise distance
	$d_{centroid}$	Distance between strokes' centroids
	d_{minLL}	Minimum inter-stroke endpoint-to-endpoint distance
	d_{minXL}	Minimum inter-stroke endpoint-to-anypoint distance
Overlap	X-Overlap	Intersection of strokes' projection onto x-axis
	Y-Overlap	Intersection of strokes' projection onto y-axis
Temporal	Time Gap	Time between the end of the first stroke and the beginning of the second stroke
Ratios	$Closeness_{Stroke_i}$	Comparison of d_{min} to the distance of neighboring strokes to $Stroke_i$
	$Closeness_{Stroke_j}$	Comparison of d_{min} to the distance of neighboring strokes to $Stroke_j$
	$Ratio_{LL}$	Comparison of d_{minLL} to d_{min}
	$Ratio_{XL}$	Comparison of d_{minXL} to d_{min}
Perceptual	Same Closed Path	Whether the two strokes are part of the same closed path

Table 3.1: List of features for pairwise classification

part of the same object *and* are close to each other. Conversely, *FarJoin* pairs contain strokes that are part of the same object, yet are far apart. In Figure 3.1, the stroke pairs AB, BC, DE, DF, and EF are *NearJoins*. Stroke pair AC is a *FarJoin*. All other pairs in the figure are *NoJoins*. Chaining assembles the pairs classified as *NearJoin* to form larger clusters.

The classifier is usually able to reliably learn the difference between *NearJoins* and *NoJoins*. However, it more regularly confuses *FarJoin* pairs with both *NearJoins* and *NoJoins*. During chaining, we can rely on *NearJoins* to form the entire cluster together, thus *FarJoin* pairs are not directly joined by the chainer, and instead rely on intermediate *NearJoins*.

The key challenge is determining the optimal distinction between *NearJoins* and *FarJoins*. We have explored two approaches to this:

1. ***Minimum Distance*** – Uses the minimum distance between the strokes to determine the pair's label.
2. ***Iterative Re-Labeling*** – Uses cluster accuracy (post-chaining) to re-label pairs in order to improve classifier training.

Minimum Distance Method

The *Minimum Distance* method is used to label pairs of strokes that belong to the same object as either *NearJoin* or *FarJoin*. Pairs of strokes from different objects are always labeled as *NoJoin*. A pair of strokes ($Stroke_i$ and $Stroke_j$) from the same object ($Shape_s$) is labeled as a *NearJoin* if one or more of the following conditions are satisfied:

1. $Stroke_i$ is the nearest stroke to $Stroke_j$ in $Shape_s$, or vice versa.
2. The minimum distance between $Stroke_i$ and $Stroke_j$ is less than or equal to a constant threshold, D_0.
3. The minimum distance between $Stroke_i$ and $Stroke_j$ is about the same distance as another *NearJoin* of $Stroke_i$ or $Stroke_j$, found using Condition 1.

If none of these conditions is satisfied, the pair is labeled as a *FarJoin*.

These conditions can be expressed mathematically as follows. If $d_{min}(Stroke_i, Stroke_j) \leq D_{NJ}$ then the pair is labeled as a *NearJoin*, otherwise it is labeled as a *FarJoin*. The threshold D_{NJ} (Distance for Near Join) is defined as:

$$D_{NJ} = max\{T_{MDJ}, (1+T) * max\{Closeness'_{Stroke_i}, Closeness'_{Stroke_j}\}\} \tag{3.3}$$

where T_{MDJ} (Threshold for Minimum Distance Join) is an empirical threshold of 200 himetrics (2mm), and T is a constant threshold equal to 0.15[2]. $Closeness'$ is the minimum distance from a stroke to any other stroke in the same shape:

$$Closeness'_{Stroke_i} = \min_{Stroke_a \in Shape_s} d_{min}(Stroke_i, Stroke_a) \tag{3.4}$$

$Closeness'_{Stroke_j}$ is defined analogously.

The term $max\{Closeness'_{Stroke_i}, Closeness'_{Stroke_j}\}$ in Equation 3.3 represents Condition 1. Condition 2 is represented by the T_{MDJ} term. Condition 3 is represented by applying the coefficient $(1+T)$ to Condition 1.

We selected a value of 200 himetrics for T_{MDJ} based on initial testing with the *Thresholded Pairwise Classifier* (TPC) that showed 200 to be the best value for T_{JD} for digital circuits. While T_{JD} can vary for each user in each domain[3], 200 himetrics worked well and is used for our *Minimum Distance* labeling method in each of our domains.

Iterative Re-Labeling Method

The *Iterative Re-Labeling* technique iteratively modifies the labels of the training data such that the final stroke clustering accuracy is optimized, rather than the

[2]T is the same as the threshold used for computing stroke intersections and self enclosure, as described in Section 2.2.3

[3]In fact, later (more fine-grained) analysis found different values for T_{JD} than 200. Results from this analysis are presented in Section 5.2.1.

pairwise classification accuracy.

As an example of how the iterative re-labeling works, imagine that the stroke pair DF from Figure 3.1 was initially labeled as *NearJoin*, but was classified as a *FarJoin*. Also, stroke pairs DE and EF were labeled and classified as *NearJoins*, thus the chainer created a cluster with strokes D, E, and F. Although pair DF was not directly joined, it became part of the same cluster via chaining. Because the classifier incorrectly classified stroke pair DF according to the initial label, yet it ended up not mattering, its label was changed to *FarJoin*. This updated label (along with all the other updated labels) allows the classifier to better learn the distinction between *NearJoins*, *FarJoins*, and *NoJoins* in subsequent iterations.

A flowchart depicting the *Iterative Re-Labeling* algorithm is shown in Figure 3.6. It uses the rules in Table 3.2 to iteratively update *NearJoin* and *FarJoin* labels based on the errors in the final clusterings. The stroke pair labels are initially set (seeded) using the labels determined by the *Minimum Distance* method. We examine the effects of using different values for T_{MDJ} during this seeding process in Section 5.2.6. The training data is then split into two equal subsets, Set_1 and Set_2. Next, the algorithm begins iterating by first training a classifier on Set_1, then using it to classify the instances in Set_2. The results for Set_2 are chained to produce clusters, at which point the rules in Table 3.2 are used to determine the new labels for the instances in Set_2. A new classifier is then trained on the updated Set_2, and used to classify the instances in Set_1. The pairs in Set_1 are chained and then re-labeled. This process repeats for a fixed number of iterations; we used five. Then, Set_1 and Set_2 are combined and their

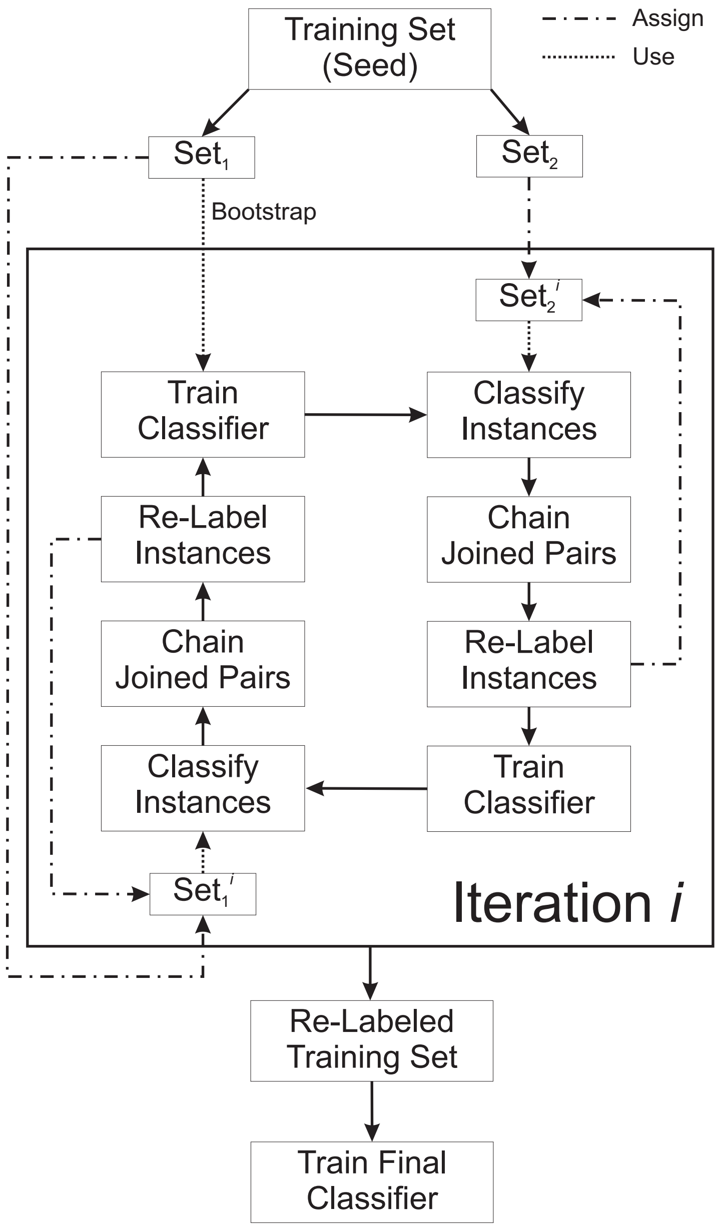


Figure 3.6: Flowchart showing the iterative re-labeling process. The hand-labeled training set is split evenly into Sets 1 and 2. Our algorithm performs five iterations to obtain the final training set. Re-Labeling is performed according to the rules in Table 3.2.

latest labels are used to train a final pairwise classifier.

Definitions		
Abbreviation	**Description**	**Possible Values**
TL_i	True Label of pair i (Labeled as part of same object)	{ Join, NoJoin }
PL_i	Psuedo Label of pair i (Label for classifier training)	{ NearJoin, FarJoin, NoJoin }
AC_i	Actual Classification of pair i (Output from classifier)	{ NearJoin, FarJoin, NoJoin }
CC_i	Chained Classification of pair i (Part of same object based on chaining of classifier output)	{ Join, NoJoin }

Rules	
Condition	**Updated $PL_i =$**
$AC_i == TL_i$ && $TL_i == Join$	NearJoin
$AC_i \neq TC_i$ && $CC_i == TL_i$ && $TL_i == Join$	FarJoin
$AC_i \neq TL_i$ && $CC_i \neq TL_i$ && $TL_i == Join$	NearJoin
$AC_i == TL_i$ && $TL_i == NoJoin$	NoJoin
$AC_i \neq TC_i$ && $CC_i == TL_i$ && $TL_i == NoJoin$	NoJoin
$AC_i \neq TL_i$ && $CC_i \neq TL_i$ && $TL_i == NoJoin$	NoJoin

Table 3.2: Rules for updating grouping labels.

3.4 Summary

We have created two different grouping algorithms: the *Thresholded Pairwise Classifier* and the *Inductive Pairwise Classifier*. Both of these algorithms classify **pairs** of strokes to determine which strokes belong together. They also both use a post-classification chaining technique to cluster stroke pairs that have a stroke in common.

The *Inductive Pairwise Classifier* classifies stroke pairs as *NearJoin*, *FarJoin*, or *NoJoin*. We created two techniques to determine *NearJoin* versus *FarJoin* labels: the *Minimum Distance Method*, and the *Iterative Re-Labeling Method*.

Chapter 4

Test Data

We evaluated our algorithms on four different domains: digital circuit diagrams, complete statics solutions, statics solutions without equations, and family tree diagrams. We collected the data for the first three domains ourselves, while the family tree sketches are taken from a publicly available data set [28].

These domains differ from each other in several respects, and thus provide a good evaluation of the generality of our approach. Sketches of family trees and digital circuits are typically comprised of connected sets of strokes, while statics solutions usually contain multiple drawing regions which are separated from each other. The distribution of strokes in digital circuits favors non-text (shape) strokes, while in statics solutions, a large majority of the strokes are text. Each of the domains has a variety of shapes, some are drawn very similar to their archetype, while others have an amorphous shape. For instance, *and* gates (Table 4.1) are usually drawn such that they resemble the archetypical *and* gate. However, *bodies* (Table 4.2) and *wires* (Table 4.1) do not have predefined shapes. Based on the results from these four domains, we demonstrate that our methods are extensible to new domains.

4.1 Digital Circuits

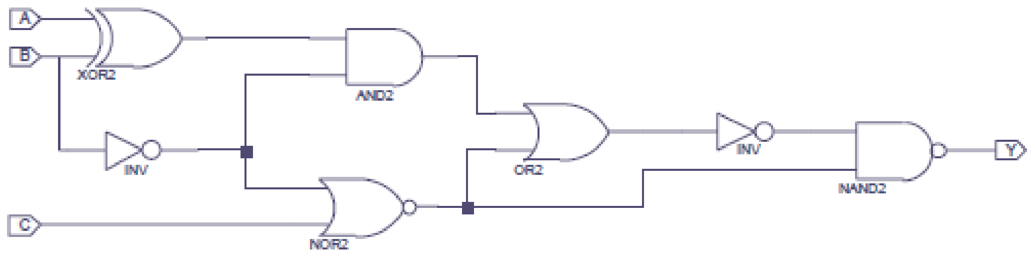
The Digital Circuit data set was collected from 24 undergraduate students at Harvey Mudd College and the University of California, Riverside. A total of 192 sketches of complete circuits were collected from the users. Each study was comprised of two sessions, separated by a week's time. One session was performed using a Tablet PC, the other was performed using a digitizing pen on paper. In each session, the subjects performed three tasks: repeatedly drawing isolated symbols, copying the circuit diagrams shown in Figures 4.1(a) and 4.1(b), and synthesizing circuit diagrams to satisfy the following logic equations:

$$Y = \left(\overline{(AB + C)} \oplus \overline{AC}\right) + B + C \tag{4.1}$$

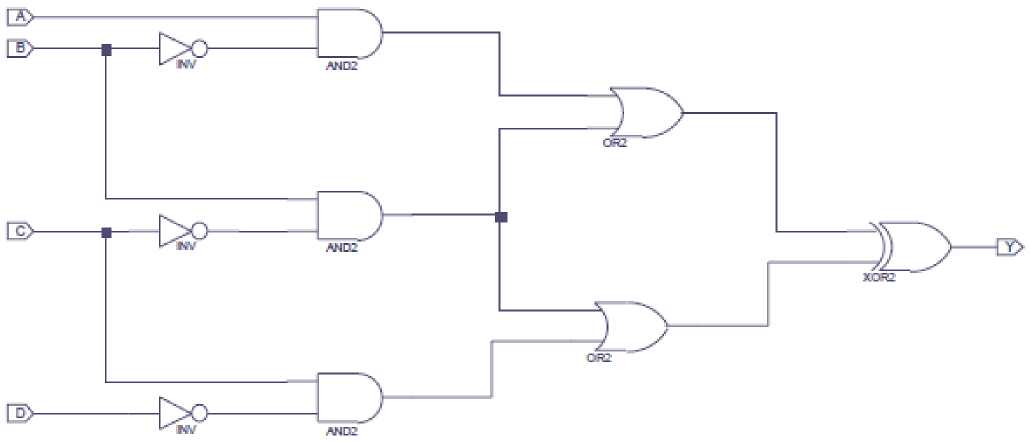
$$Y = \left(\overline{A} + BC\right)\left(A\overline{B} + AC + B\right) \tag{4.2}$$

The orders of the sessions and tasks were evenly distributed among subjects, while the order of sketches within a task was randomized. Our algorithms were trained and evaluated using the complete circuit diagrams from the copy and synthesize tasks.

The subjects were asked to draw naturally, and were given no feedback from the computer (the program simply recorded the ink as it was drawn). Each user-study participant had previously taken, or was in the process of taking, a course covering



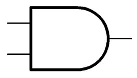
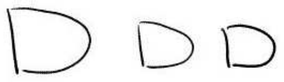
(a) First Digital Logic Circuit.



(b) Second Digital Logic Circuit.

Figure 4.1: User-study participants were instructed to copy these two digital circuit diagrams.

digital circuits and logic. An example sketch is shown in Figure 4.2. Appendix A includes more examples of digital circuit diagram sketches. Table 4.1 shows a list of the expected shapes in the digital circuit domain.

Digital Circuits			
Class	Name	Archetype	Examples
Gate	AND		

Continued on next page

Table 4.1 – continued from previous page

Digital Circuits

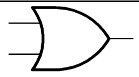
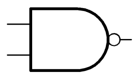
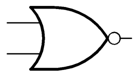
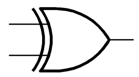
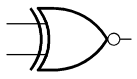
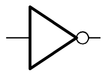
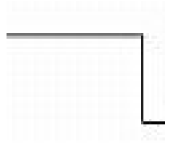
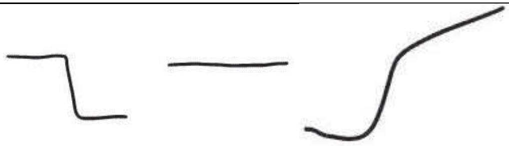
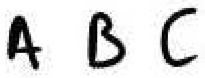
Class	Name	Archetype	Examples
Gate	OR		
Gate	NAND		
Gate	NOR		
Gate	XOR		
Gate	XNOR		Not Observed
Gate	NOT		
Gate	NOTBUBBLE		
Wire	Wire		
Label	Label	Characters	

Table 4.1: Domain shapes for digital circuit diagrams.

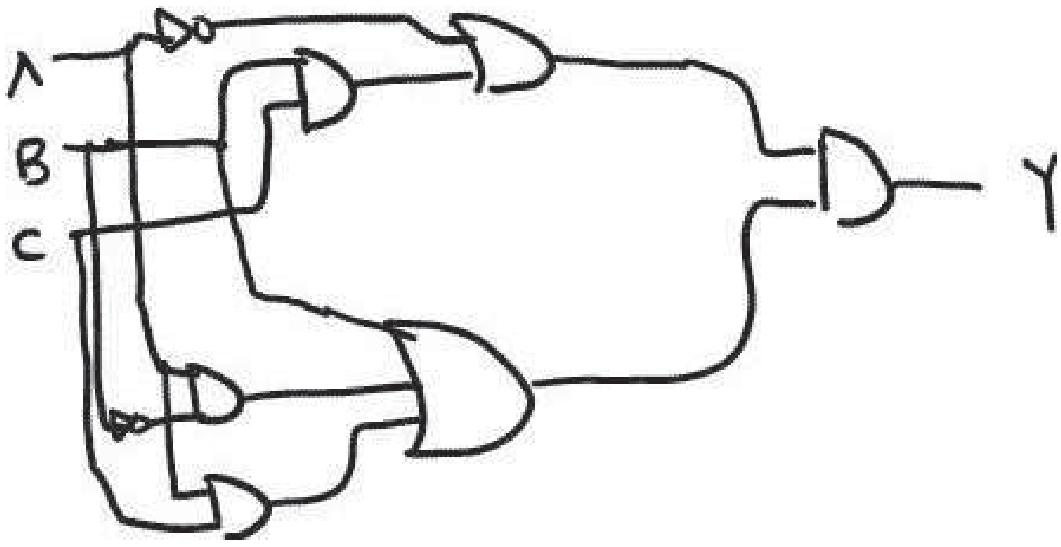


Figure 4.2: Example sketch from the digital circuit user study. This circuit was synthesized from Equation 4.1 and was drawn using a digitizing pen on paper.

4.2 Statics Solutions

We performed a user study to collect solutions to engineering statics problems. We collected sketches from 16 subjects at the University of California, Riverside, all of whom were concurrently enrolled in a Statics course, and were therefore knowledgeable about the topic but not expert. Each participant solved six multi-body (machine) problems by drawing the necessary free-body diagrams and constructing the equilibrium equations necessary to solve for the specified forces. Subjects were instructed *not* to solve the equilibrium equations. Figure 4.3 shows a typical problem from the study; Figure 4.4 shows a solution to this problem that was collected during the study. Note that the subject is not required to algebraically solve the equations to determine the specific value for the moment M.

As in the digital circuit study (Section 4.1), subjects were instructed to draw

The nose-wheel assembly is raised by the application of a torque *M* to link *BC* through the shaft at *B*. If the arm and wheel *AO* have a combined weight of *W* with center of gravity at *G*, find the value of *M* necessary to lift the wheel when *D* is directly under *B*, at which position the angle is θ.

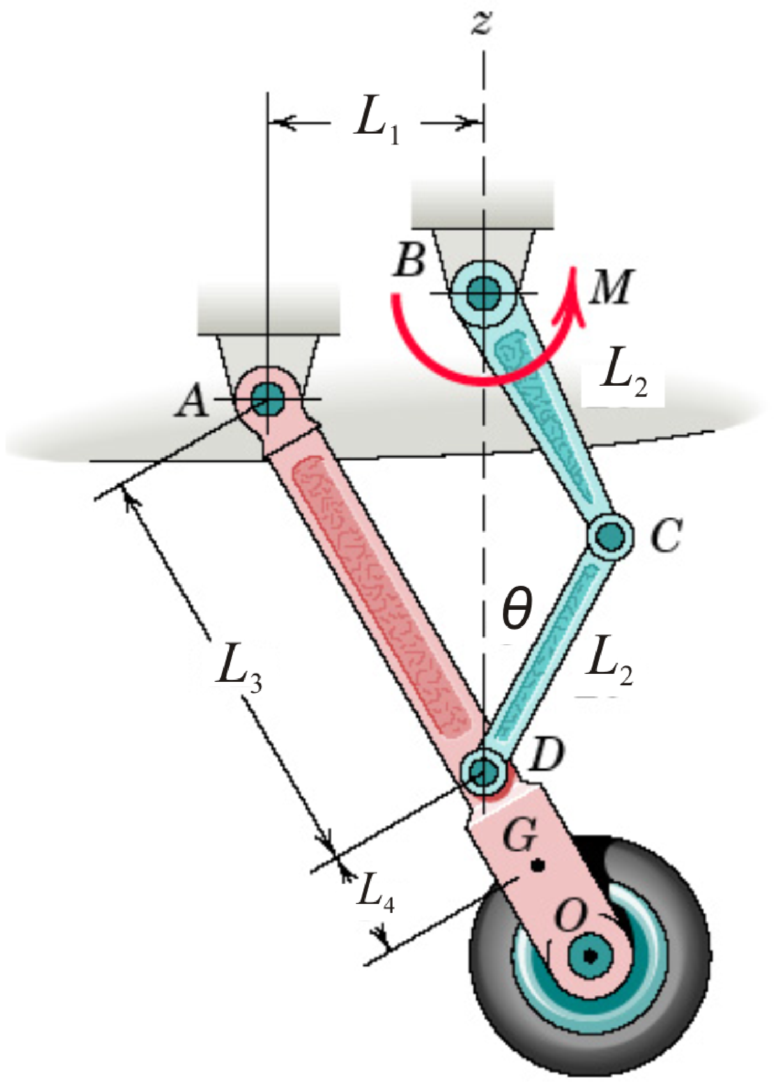
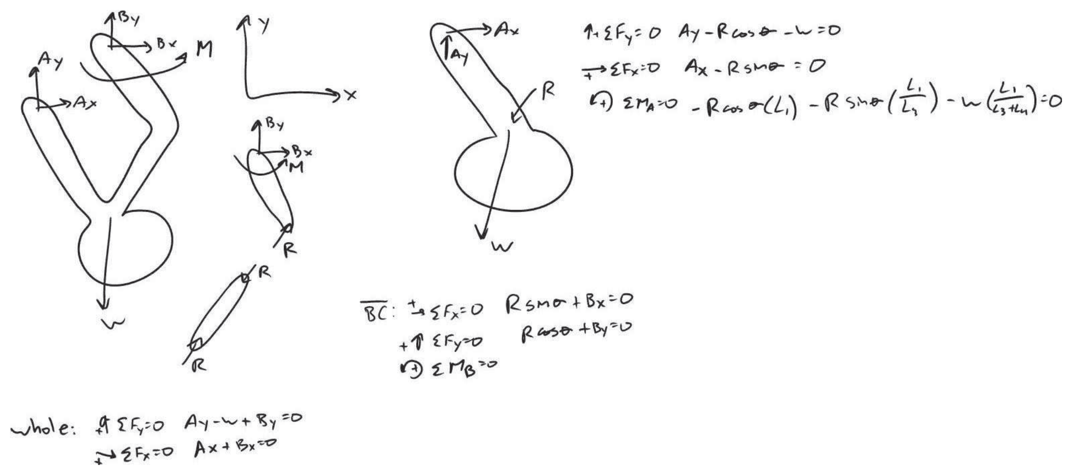


Figure 4.3: Prompt for a statics problem used in the study. The image and prompt are from Engineering Mechanics: Statics [27], copyright John Wiley and Sons, Inc., used with permission. The image and prompt were modified, replacing all numeric values with variables.

naturally and solve the problem as if it was homework or an exam. The program acted as a recording interface with no recognition feedback. A total of six problems were solved by each subject: Figure 4.3 and Figures B.1-B.5. Table 4.2 shows examples of each of the expected shapes for the statics domain.

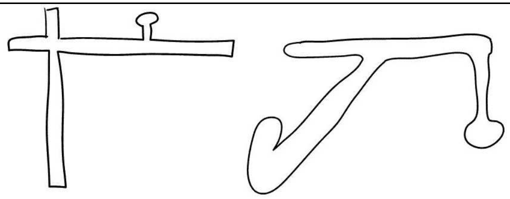
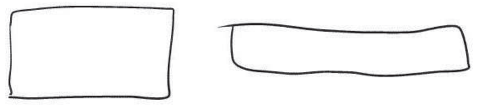
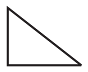
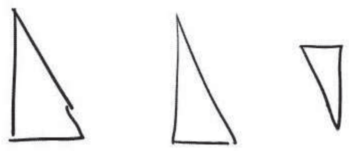
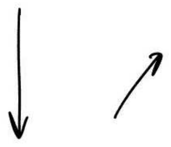
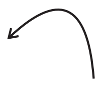


Ay
Ax
By
Bx
M
y
x
W
R
$\uparrow + \Sigma F_y = 0 \quad A_y - R\cos\theta - W = 0$
$\xrightarrow{+} \Sigma F_x = 0 \quad A_x - R\sin\theta = 0$
$\Sigma M_A = 0 \quad -R\cos\theta(L_1) - R\sin\theta\left(\frac{L_1}{L_1}\right) - W\left(\frac{L_1}{L_3+L_4}\right) = 0$
$\overline{BC}$: $\xrightarrow{+} \Sigma F_x = 0 \quad R\sin\theta + B_x = 0$
$+\uparrow \Sigma F_y = 0 \quad R\cos\theta + B_y = 0$
$\Sigma M_B = 0$
whole: $\uparrow \Sigma F_y = 0 \quad A_y - W + B_y = 0$
$\xrightarrow{+} \Sigma F_x = 0 \quad A_x + B_x = 0$

Figure 4.4: Example statics solution collected during the user study to the problem shown in Figure 4.3.

We have created two domains from the data collected during the statics user study. The first includes all strokes and the sketches are referred to as complete statics solutions. The second is created from a subset of the complete statics domain by removing all equation related strokes and it is referred to as statics solutions without equations. We consider this domain because it is representative of an interface with two drawing areas, one for free-body diagrams, and one for equations. A statics solution sketch without equations contains the free-body diagram portion of the complete sketch.

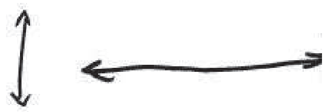
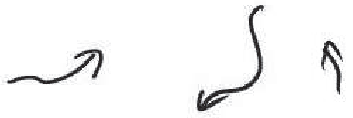
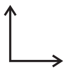
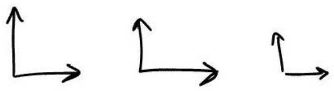
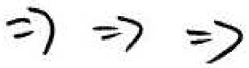
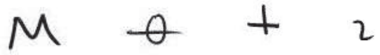
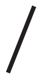
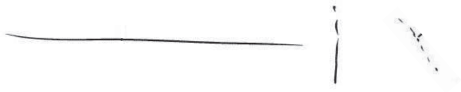
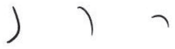
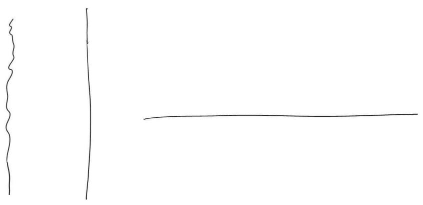
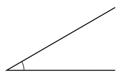
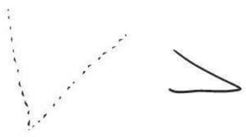
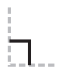
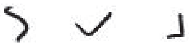
In complete statics solution sketches, we labeled individual characters as their own groups. For instance, the equation $F_1 + P = 0$ would be grouped into the characters: F, 1, $+$, P, $=$, and 0. A side-effect of this labeling scheme is that there is an order of magnitude more text objects than bodies, arrows, and "other" objects, combined. This imbalance skews the overall grouping accuracies towards the accuracies for the text class. For statics solutions without equations, all characters in a label for a force, moment or other object were grouped together. For example, the force label F_1 is a single object in this domain despite being comprised of two characters.

Statics			
Class	Name	Archetype	Examples
Body	Bodies	Body Outline	
Body	Box		
Body	Triangle		
Arrow	Force		
Arrow	Moment		
Arrow	Force Equation		
Arrow	Moment Equation		

Continued on next page

Table 4.2 – continued from previous page

Statics

Class	Name	Archetype	Examples
Arrow	Dimension	$\longleftrightarrow$	
Arrow	Pointer	Arrow	
Arrow	Coordinate System		
Arrow	Double Shafted	$\Longrightarrow$	
Text	Text	Characters	
Other	Leader Line		
Other	Arc		
Other	Divider		
Other	Angle		
Other	Angle Square		

Continued on next page

Table 4.2 – continued from previous page

Statics

Class	Name	Archetype	Examples
Other	Point	●	
Other	Ellipsis	■ ■ ■	

Table 4.2: Domain shapes for statics solution sketches.

4.3 Family Tree Diagrams

The family tree data is taken from the ETCHA Sketch[1] corpus [28]. We use a total of 27 sketches from 9 users, all drawn on a Tablet PC (sketches containing fewer than 5 strokes, or those which are subsets of other sketches, were not used). Users drew freely and received no recognition feedback. Figure 4.5 shows an example sketch; more examples can be found in Appendix C. Examples of shapes in the family tree domain are presented in Table 4.3. For this domain, entire words and descriptions of people are considered single groups. For example, the name "Harold", for the lowest person in Figure 4.5, is a single group.

[1]`http://rationale.csail.mit.edu/ETCHASketches/`

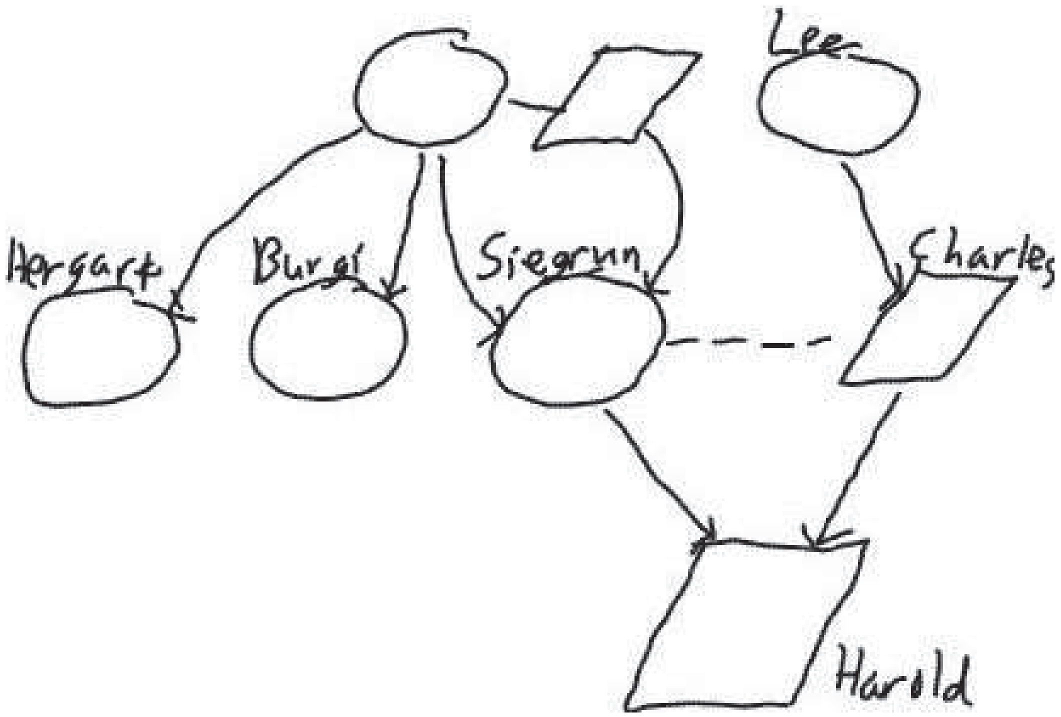
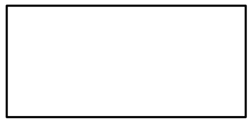
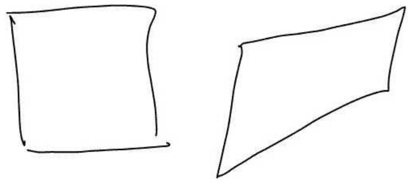
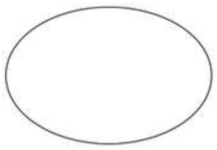
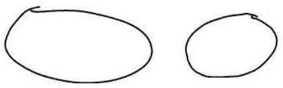


Figure 4.5: Example sketch of a family tree diagram from the ETCHA Sketch corpus.

Family Trees			
Class	Name	Archetype	Examples
People	Male		
People	Female		
Text	Text	Whole words / Titles	Shelton Father wife uncles + aunts
Links	Marriage		

Continued on next page

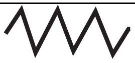
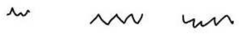
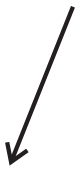
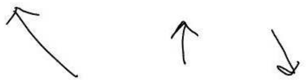
Table 4.3 – continued from previous page			
Family Trees			
Class	Name	Archetype	Examples
Links	Divorce		
Links	Childlink		

Table 4.3: Domain shapes for family tree diagrams.

Chapter 5

Results

In this chapter we present evaluations of our algorithms on data sets presented in Chapter 4. Section 5.1 presents single-stroke classification results. Section 5.2 presents grouping results. Section 5.3 presents computational cost and running time results. We discuss performance, strengths and weaknesses, and possible future work in Chapter 6.

5.1 Single-Stroke Classification Results

The goal of the single-stroke classifier is to classify strokes into categories which result in separation between objects. After the strokes have been classified, strokes of the same class are grouped into complete objects. To evaluate our single-stroke classifier we perform user-holdout validations in each domain. Classifier models that are trained in one domain are not used in others. The results presented in this section are the aggregate of the individual user-holdouts, unless otherwise stated. In this section we will address the following questions:

1. **Benchmarking:** How well does this technique perform compared to previous approaches in text versus non-text classification?

2. **Multi-Way Classification:** How well are strokes classified into different classes?

3. **Effect of User-Specific Training Data:** How does the inclusion of user-specific data affect accuracy?

4. **Single-Stroke Feature Importance:** What are the most important features for single-stroke classification?

5.1.1 Benchmarking

For our application, we classify strokes into three or more classes. However, to benchmark our classifier we restrict it to two classes, text versus non-text, and compare it to two state-of-the-art methods: the entropy approach presented in [3], and Microsoft's© InkAnalyzer, a commercial product. The results of these comparisons in each of the four domains are presented in Figure 5.1. The naïve classifier in the figure classifies every stroke as the most frequent class, therefore its accuracy is the same as the frequency of the most common class.

As can be seen in Figure 5.1, our method performs better than InkAnalyzer in each of the four domains. Our method also performs better than the entropy method in three of four domains, while in family tree sketches, the entropy method has slightly higher accuracy.

For digital circuits, our method's overall accuracy is 97.2%, as compared to 85.7% for the entropy method (which reverted to naïve classification) and 63.4% for

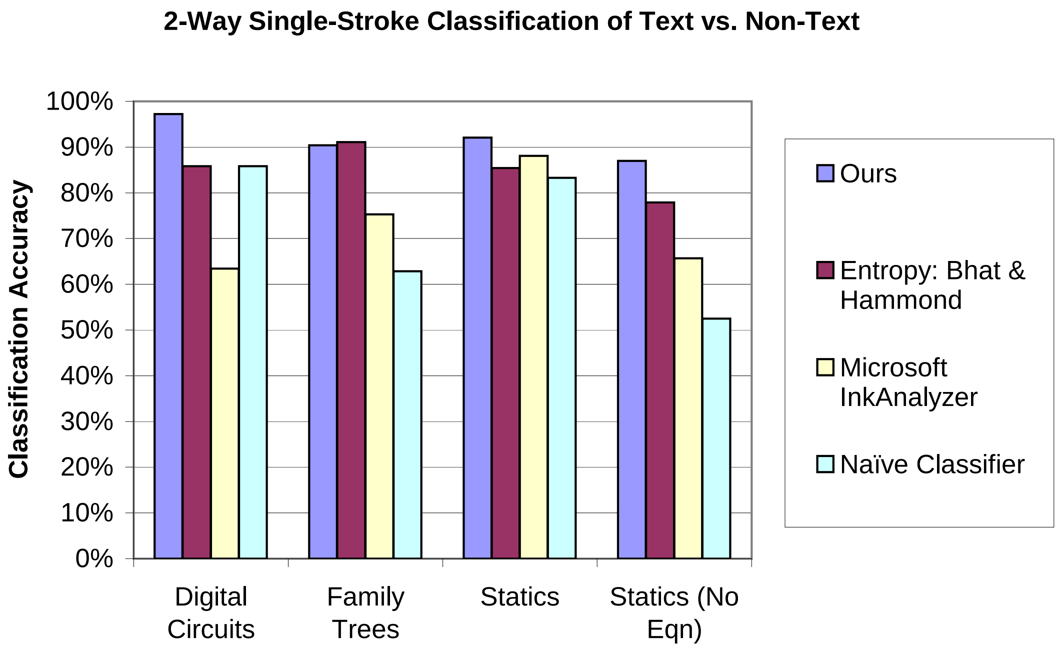


Figure 5.1: Comparison of accuracy for text vs. non-text single-stroke classification against two existing systems. Black lines represent the frequency of the most common class – this is the same as the accuracy of a naïve classifier. InkAnalyzer = Microsoft© InkAnalyzer (a commercial application), Entropy = method by Bhat and Hammond [3].

InkAnalyzer. Table 5.1 shows the confusion matrices for the three methods on digital circuits. Our method performs well for both classes, while InkAnalyzer classified many *NonText* strokes as *Text*, and the entropy method classified every stroke as *NonText*.

In the family tree domain our method's classification accuracy is 90.4%, which is slightly lower than the entropy method's 91.1% accuracy. Both methods performed better than InkAnalyzer, which has an overall accuracy of 75.3%. Family tree sketches, on average, were composed of 62.8% *NonText* strokes, thus all three classifiers performed better than a naïve classifier. Confusion matrices are presented in Table 5.2. Our method performed similarly for both classes, whereas InkAnalyzer again tended to classify many *NonText* strokes as *Text*. The entropy method performed better for *NonText* than *Text*,

Method	Actual Class	Classified As		Accuracy
		Text	NonText	
Ours	Text	1255	142	89.8%
	NonText	133	8272	98.4%
			TOTAL	97.2%
InkAnalyzer	Text	963	434	68.9%
	NonText	3152	5253	62.5%
			TOTAL	63.4%
Entropy	Text	0	1397	0.0%
	NonText	0	8405	100.0%
			TOTAL	85.7%

Table 5.1: Accuracy for single-stroke classification using *Text* vs. *NonText* on digital circuit sketches. InkAnalyzer = Microsoft© InkAnalyzer (a commercial application), Entropy = method by Bhat and Hammond [3].

however both classes were fairly accurate.

Method	Actual Class	Classified As		Accuracy
		Text	NonText	
Ours	Text	555	62	90.0%
	NonText	97	946	90.7%
			TOTAL	90.4%
InkAnalyzer	Text	617	0	100.0%
	NonText	410	633	60.7%
			TOTAL	75.3%
Entropy	Text	505	112	81.8%
	NonText	36	1007	96.5%
			TOTAL	91.1%

Table 5.2: Accuracy for single-stroke classification using *Text* vs. *NonText* on family tree diagram sketches. InkAnalyzer = Microsoft© InkAnalyzer (a commercial application), Entropy = method by Bhat and Hammond [3].

Our method outperforms the other two on statics sketches, with an overall accuracy of 92.1%. InkAnalyzer has an overall accuracy of 88.1%, while the entropy method is 85.4% accurate. Table 5.3 shows the confusion matrix; our method performs well on *Text*, however it does not classify *NonText* accurately. Despite this, our *NonText* accuracy is still much better than that of the other two methods. InkAnalyzer is very good at classifying *Text*, but once again has poor accuracy on *NonText*. The most likely reason for low accuracy on *NonText*, for both our method and the entropy method, is that there is an order of magnitude more *Text* strokes than *NonText* – 83.3% of the strokes were *Text*. This imbalance skews the training process in favor of *Text* examples.

Method	Actual Class	Classified As Text	Classified As NonText	Accuracy
Ours	Text	21300	703	96.8%
	NonText	1378	3030	68.7%
			TOTAL	92.1%
InkAnalyzer	Text	21911	92	99.6%
	NonText	3041	1367	31.0%
			TOTAL	88.1%
Entropy	Text	20558	1445	93.4%
	NonText	2410	1998	45.3%
			TOTAL	85.4%

Table 5.3: Accuracy for single-stroke classification using *Text* vs. *NonText* on complete statics sketches. InkAnalyzer = Microsoft© InkAnalyzer (a commercial application), Entropy = method by Bhat and Hammond [3].

For statics sketches without equations, our method again outperforms the others, with an overall accuracy of 86.8%. The confusion matrix is presented in Table 5.4. Here, the entropy method is second best, with 77.9% overall accuracy, while InkAnalyzer is 65.7% accurate. The most common class was *NonText*, making up 52.5% of the strokes.

5.1.2 Multi-Way Classification

In this section, we evaluate the accuracy of our single-stroke classifier for its intended application. Examples of program output can be found in Appendices A-C. Our overall single-stroke classification accuracy for each domain is presented in Figure

Method	Actual Class	Classified As		Accuracy
		Text	NonText	
Ours	Text	3260	455	87.8%
	NonText	581	3532	85.9%
			TOTAL	86.8%
InkAnalyzer	Text	3663	52	98.6%
	NonText	2630	1483	36.1%
			TOTAL	65.7%
Entropy	Text	3306	409	89.0%
	NonText	1321	2792	67.9%
			TOTAL	77.9%

Table 5.4: Accuracy for single-stroke classification using *Text* vs. *NonText* on statics sketches without equations. InkAnalyzer = Microsoft© InkAnalyzer (a commercial application), Entropy = method by Bhat and Hammond [3].

5.2.

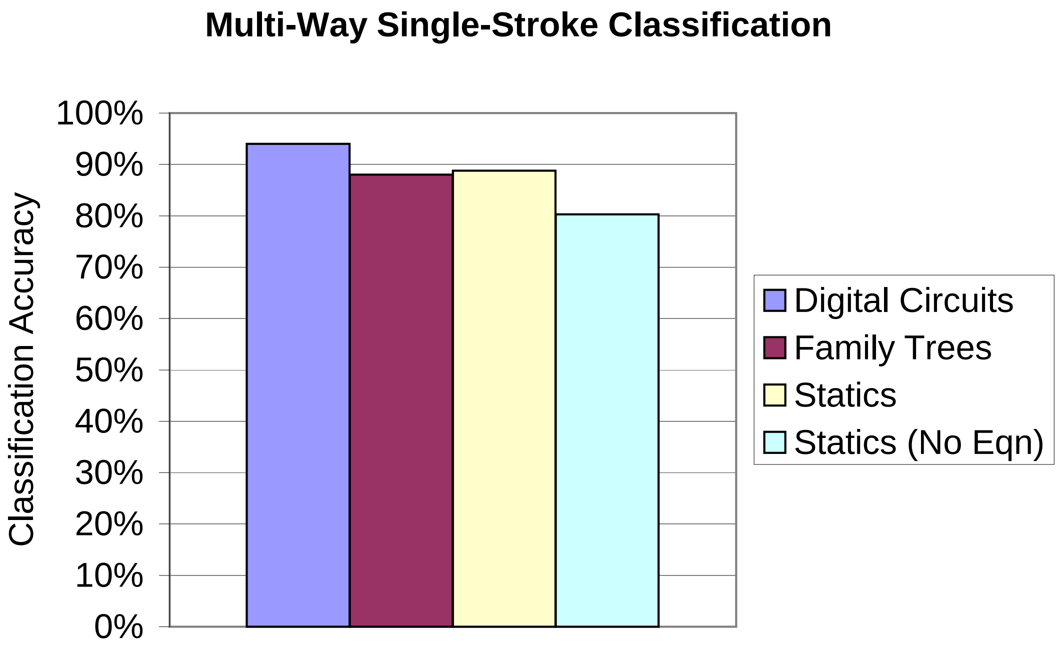


Figure 5.2: Accuracy of the single-stroke classifier, as used in the complete grouping process, in each domain.

In the digital circuit domain, strokes are classified as gate strokes, wire strokes, or label strokes, and the classifier has an overall accuracy of 93.6%. Table 5.5 presents the confusion matrix for this domain. The accuracies on a per-class basis are similar to one another, indicating that the classifier has not learned to classify one class of strokes at the expense of the other classes. There is little confusion between the label and wire classes. There is comparatively more confusion between these classes and the gate class.[1]

Class		Classified As			Accuracy
		Gate	Wire	Label	
Actual	Gate	3966	194	116	92.8%
	Wire	170	3930	29	95.2%
	Label	108	14	1275	91.3%
				TOTAL	93.6%

Table 5.5: Accuracy for multi-way single-stroke classification of digital circuit diagrams.

For family tree diagrams, strokes are separated into three classes: people, text, and links. In this domain the classifier has an overall accuracy of 88.0%. Similar to digital circuits, all classes have similar accuracies.

Class		Classified As			Accuracy
		People	Text	Link	
Actual	People	338	37	32	83.0%
	Text	11	568	38	92.1%
	Link	24	60	552	86.8%
				TOTAL	87.8%

Table 5.6: Accuracy for multi-way single-stroke classification of family tree diagrams.

In statics solutions sketches without equations, the strokes are separated into four classes: *Bodies*, *Arrows*, *Labels*, and *Other*. The overall accuracy is 80.3%, however

[1]Note that we also experimented with a two-step approach to multi-way classification by first classifying strokes into text and non-text strokes, followed by classification of the non-text strokes into gates and wires. This two-step approach performed slightly worse than using a single classifier to perform multi-way classification.

there is a wide range of per-class accuracies. *Bodies* are confused with each of the other classes at roughly equivalent rates. *Arrows* and *Labels* are the most confused classes, indicating that they have many of the same properties. Strokes in the *Other* class are regularly confused with *Arrows* and *Labels*. This is likely due to the wide range of strokes in this category, some of which look very similar to *Arrows* and *Labels*.

Complete statics solution sketches (including equations) are separated similarly, using the classes: *Bodies*, *Arrows*, *Text*, and *Other*. The overall accuracy in this domain is 88.8%. However, this is dominated by the *Text* classification accuracy of 96.7%. The other classes had poor accuracy and were regularly confused with text strokes. The class distribution should be adjusted to improve the accuracy of the other three classes in this domain. The two easiest ways to do this would be to use a subset of the *Text* stroke training examples during training, or to weight the classes such that each example of a non-text stroke is worth more than a text stroke. By weighting the non-text strokes, the weighted number of strokes per class can be equalized.

We have also experimented with a larger number of classes in statics sketches. The classes are: *Bodies*, *Arrows*, *Labels*, *Equations*, *Geometry*, *Lines and Arcs*, *Points*,

Class		Classified As				Accuracy
		Body	Arrow	Labels	Other	
Actual	Body	632	60	75	49	77.5%
	Arrow	40	1559	434	135	71.9%
	Labels	17	272	3330	96	89.6%
	Other	32	172	158	767	67.9%
					TOTAL	80.3%

Table 5.7: Accuracy for multi-way single-stroke classification of statics solutions without equations.

Class		Classified As				Accuracy
		Body	Arrow	Text	Other	
Actual	Body	608	82	101	42	73.0%
	Arrow	51	1505	1164	89	53.6%
	Text	24	474	20703	202	96.7%
	Other	43	115	561	647	47.4%
					TOTAL	88.8%

Table 5.8: Accuracy for multi-way single-stroke classification of statics solutions.

and *Other*. Although the classes are not the same, examples of all shapes are presented in Table 4.2. *Lines and Arcs* are composed of leader lines, arcs, and dividing lines. The *Geometry* class consists of angles and other strokes drawn by the user to help understand the geometry and trigonometry involved in the statics problem. *Labels* are text strokes drawn to label forces, moments, dimensions, etc. *Equations* are the remaining text strokes (consisting of equations and other long strings of text). The confusion matrix in Table 5.9 shows that the classifier is unable to learn the distinction between many of the classes. The two types of text, *Labels* and *Equations*, are often confused. *Geometry* strokes were never correctly classified, while *Other* strokes were classified correctly 10.7% of the time. We have found that a good set of classes is necessary to perform accurate classification, guidelines for this are presented in Section 6.1.1.

Class		Classified As								Accuracy
		Arrow	Body	Eqn	Geo	Label	L/A	Other	Pt	
Actual	Arrow	1230	58	586	3	244	63	8	2	56.1%
	Body	90	616	80	0	48	32	5	2	70.6%
	Equation	238	26	16755	1	1138	132	14	7	91.5%
	Geometry	17	8	27	0	6	1	0	0	0.0%
	Label	157	9	1929	1	1611	34	3	3	43.0%
	LineArc	87	44	306	2	52	358	16	2	41.3%
	Other	29	9	87	0	24	22	22	12	10.7%
	Point	2	4	8	0	10	1	15	115	74.2%
									TOTAL	78.4%

Table 5.9: Single-stroke classification accuracy for 8-way classification of statics sketches.

5.1.3 Effect of User-Specific Training Data

Often in classification, including user-specific training data can improve accuracy. We experimented with this in two domains: digital circuits, and complete statics solutions. We performed a sketch-holdout in each domain (previous sections have used user-holdout). Sketch-holdout is performed by training the classifier on data from every sketch except for the one being evaluated. For instance, in statics each user drew six sketches. For sketch-holdout, five sketches from a user were included in the training data with that of other users, while the sixth sketch was used for testing. We repeated this training and testing process for every sketch. Figure 5.1.3 compares the accuracy of sketch-holdout with that of user-holdout. Sketch-holdout had higher accuracy in both domains. These results indicate that we should include as much user-specific training data as possible in our system.

Confusion matrices are presented in Tables 5.10 and 5.11 for digital circuits and complete statics solutions, respectively. In these domains, the most dramatic increases in accuracy were for *Arrow* and *Other* strokes in the statics sketches. With user-specific training data included, Arrows and text were confused less frequently. This seems to indicate that arrows are often drawn in a style that is unique to a particular user.

Class		Classified As			
		Gate	Wire	Label	Accuracy
Actual	Gate	4019	175	82	94.0%
	Wire	144	3965	20	96.0%
	Label	89	8	1300	93.1%
				TOTAL	94.7%

Table 5.10: Accuracy for multi-way single-stroke classification of digital circuit diagrams, using sketch holdout rather than user holdout.

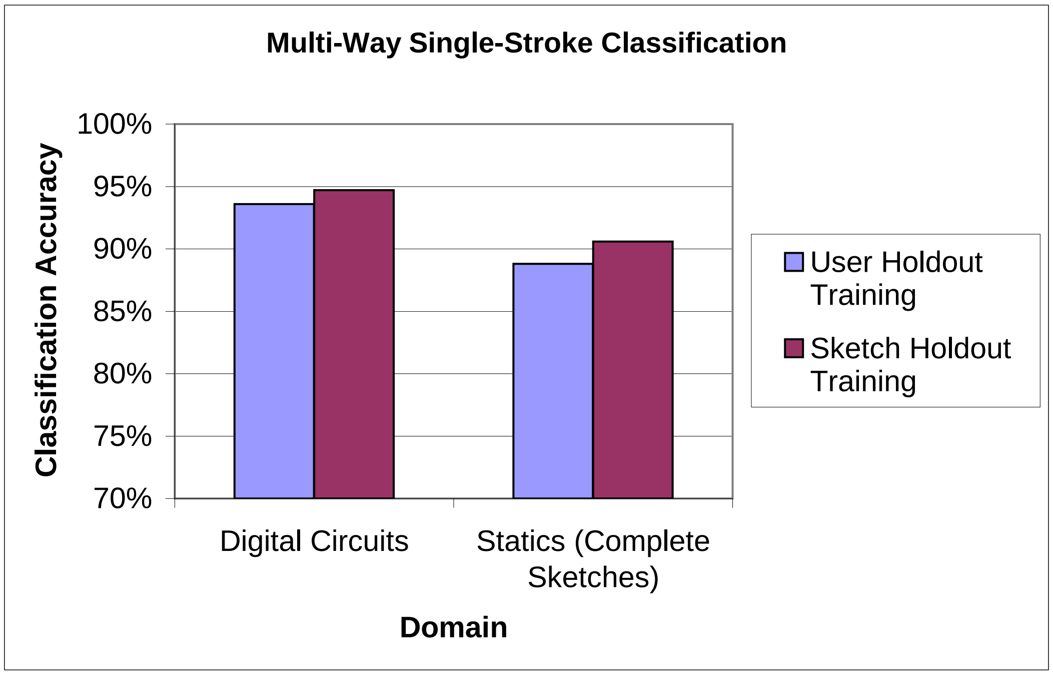


Figure 5.3: The effect of including user-specific training data (*Sketch Holdout*) versus the standard training method (*User Holdout*), for multi-way single-stroke classification.

Class		Classified As				
		Body	Arrow	Text	Other	Accuracy
Actual	Body	612	95	81	45	73.5%
	Arrow	41	1741	947	80	62.0%
	Text	19	396	20830	158	97.3%
	Other	39	103	482	742	54.3%
					TOTAL	90.6%

Table 5.11: Accuracy for multi-way single-stroke classification of complete statics solutions, using sketch holdout rather than user holdout.

5.1.4 Single-Stroke Feature Importance

Not all features are of equal worth to accurate classification. To determine the most important features, we analyzed our data sets using WEKA's information gain ratio metric, and have listed the ranked order of important features for the various domains in Table 5.12. We use WEKA's *Attribute Evaluator* with the *GainRatioAttributeEval* method. This evaluator uses the *Ranker* search method with its default parameters, and 10-fold cross-validation. A feature's rank is determined by its "merit," which is output by WEKA's information gain ratio algorithm.

It is interesting to note that in most domains, high level (perceptual) features were found to have the most information available for accurate classification. We have two perceptual features: *Part of a Closed Path* and *Inside a Closed Path.* We use them in an attempt to capture structural and contextual information, and they are novel to this work. Their importance here leads us to believe that more research into perceptual features could lead to more accurate single-stroke classification.

After the perceptual features, the most important features seem to be related to both size and location. The features representing the size of the stroke were usually in the top ten features, indicating that they are important in each of our domains. We currently use the *Arc Length* and properties of the coordinate-aligned bounding box for single-stroke classification. From this feature importance analysis, it seems that adding more size-based features, such as the area of the stroke's convex hull, may be beneficial. In both the digital circuits and family tree domains the *Distance to Left/Right* was important, but interestingly the *Distance to Top/Bottom* was not very important. This makes sense in the digital circuit domain because the labels are usually either on the left or right side of the drawing area. In computing location features, we use the bounding box of the sketch to define the drawing area and normalize the feature values with the sketch's width to make the feature scale-independent. It may be better to normalize using the area of the drawing window or the average stroke length. Using the bounding

Rank	Digital Circuits	Family Trees	Statics Solutions	Statics (No Eqn)
1	Closed Path	Inside Path	Inside Path	B-Box Area
2	Inside Path	Closed Path	# XX Intersect	B-Box Height
3	Dist L/R Edge	Self Enclosing	B-Box Area	Closed Path
4	# LX Intersect	B-Box Area	Arc Length	Arc Length
5	B-Box Width	EndPt-ArcLength	B-Box Width	B-Box Width
6	Time to Prev.	Dist L/R Edge	Time to Draw	# XX Intersect
7	B-Box Height	B-Box Width	Avg Pen Speed	Time to Draw
8	Arc Length	Arc Length	B-Box Height	Sqrt of Thetas
9	EndPt-ArcLength	B-Box Height	Sqrt of Thetas	Time to Prev.
10	# XX Intersect	Time to Draw	Closed Path	Self Enclosing
11	Time to Draw	Time to Prev.	Time to Prev.	Abs Val. Thetas
12	B-Box Area	Time to Next	Abs Val. Thetas	Time to Next
13	Signed Thetas	# XX Intersect	# LL Intersect	EndPt-ArcLength
14	Time to Next	Path Density	Max Pen Speed	Avg Pen Speed
15	Self Enclosing	Sqrt of Thetas	(Max-Min) Speed	(Max-Min) Speed
16	# LL Intersect	Dist T/B Edge	EndPt-ArcLength	Max Pen Speed
17	Path Density	# LX Intersect	Time to Next	Squared Thetas
18	Min Pen Speed	Signed Thetas	Self Enclosing	# XL Intersect
19	# Self Intersect	# XL Intersect	# XL Intersect	Sum of Thetas
20	Dist T/B Edge	Avg Pen Speed	Dist L/R Edge	Dist L/R Edge
21	Abs Val. Thetas	Abs Val. Thetas	Squared Thetas	Inside Path
22	(Max-Min) Speed	Max Pen Speed	# LX Intersect	Path Density
23	Avg Pen Speed	(Max-Min) Speed	Dist T/B Edge	Min Pen Speed
24	Squared Thetas	Squared Thetas	Signed Thetas	# Self Intersect
25	Sqrt of Thetas	# LL Intersect	# Self Intersect	# LL Intersect
26	Max Pen Speed	Min Pen Speed	Path Density	Dist T/B Edge
27	# XL Intersect	# Self Intersect	Min Pen Speed	# LX Intersect

Table 5.12: The best features for classification of individual strokes. Features are ranked according to their average merit, as determined by WEKA's information-gain-ratio attribute-selection algorithm. Feature Color Code: Red = Perceptual, Orange = Drawing Kinematics, Yellow = Location, Green = Size, Pink = Temporal Relations, Grey = Intersections, Brown = Shape.

box to scale features can give misleading values when the sketch has few strokes, because the sketch's bounding box can be comparatively smaller than with a larger sketch.

Sketches of statics solutions seem to have properties sufficiently different from the other domains that many of the features that are important for statics are not important elsewhere. For instance, the *Average Pen Speed* and the curvature related features are more important than in other domains, indicating that the shape of the stroke is more important in statics solutions, possibly because there are so many text strokes. The importance of shape features suggests that we should investigate other ways to represent it, such as entropy, as calculated in Bhat and Hammond's work [3], and the stroke's best fit (line, circular arc, elliptical arc, polyline, complex, etc.).

Despite our efforts to improve the *Minimum Pen Speed* feature by considering only points away from the end, this feature was usually at or near the bottom of the ranked list. The *Maximum Pen Speed* was also not useful. This seems to suggest that these instantaneous speed features are not reliable indicators for a class. Another feature which was consistently ranked in the bottom half is the *Number of Self Intersections*. This is likely due to the fact that few strokes actually cross themselves.

Table 5.13 is a list of the feature rankings when all domains are combined. We combined the rankings from each of the domains by averaging their "average merit" (see Tables D.1-D.4 for complete results). In this averaged list the perceptual features are again ranked at the top, followed by mostly size-based features.

Single-Stroke Feature Importance: Average of All Domains	
Average Merit	Attribute Name
0.25	Part of a Closed Path
0.195	Inside a Closed Path
0.14	Self Enclosing
0.127	Bounding Box Area
0.121	Bounding Box Width
0.113	Arc Length
0.111	Bounding Box Height
0.106	Distance To Left or Right Edge
0.102	Number of XX Intersections
0.099	Time to Draw Stroke
0.097	End Point to Arc Length Ratio
0.091	Time to Previous Stroke
0.074	Time to Next Stroke
0.072	Sum of Sqrt of Thetas
0.059	Number of LX Intersections
0.055	Average Pen Speed
0.052	Sum of Abs Value of Thetas
0.05	Path Density
0.049	Sum of Thetas
0.045	(Max - Min) Pen Speed
0.044	Maximum Pen Speed
0.042	Distance To Top or Bottom Edge
0.04	Sum of Squared Thetas
0.039	Number of XL Intersections
0.032	Number of LL Intersections
0.022	Minimum Pen Speed
0.018	Number of Self Intersections

Table 5.13: Ranked list of features for single-stroke classification in all domains (averaged merit). Feature Color Code: Red = Perceptual, Orange = Drawing Kinematics, Yellow = Location, Green = Size, Pink = Temporal Relations, Grey = Intersections, Brown = Shape.

Additional Features

Based on our feature importance analysis, we added six additional features to our single-stroke classifier. Our classifier performed worst on our two statics domains. In this section, we show how the addition of these six features affected classification accuracy.

Contextual and size-based features were the most important for each of our domains. We added *Distance to Previous Stroke* and *Distance to Next Stroke* in an attempt to include more contextual information for the classifier. These features are computed as the distance from the last point in a stroke to the first point in the next stroke. To add to the size features, we compute the *Convex Hull Area* for a stroke. The convex hull is defined as the minimum convex set of points which contains all points in the stroke.

Drawing speed is indicative of drawing style and was also important for statics solutions. To capture additional drawing style information we added three features related to the pen-tip pressure: *Average Pressure*, *Minimum Pressure*, and *Maximum Pressure*.

Incorporating these six features, the accuracy for complete statics solution sketches increased from 88.8% to 89.5%. A confusion matrix is presented in Table 5.14.

Table 5.15 presents the confusion matrix for statics solutions without equations. There seems to be little *new* information provided by these additional features. Similar to complete statics sketches, the accuracy for statics solutions without equations

Class		Classified As				Accuracy
		Body	Arrow	Text	Other	
Actual	Body	616	82	89	46	73.9%
	Arrow	53	1569	1118	69	55.9%
	Text	27	411	20760	205	97.0%
	Other	45	102	522	697	51.0%
					TOTAL	89.5%

Table 5.14: Accuracy for multi-way single-stroke classification of statics solutions. Here, six additional features were added.

Class		Classified As				Accuracy
		Body	Arrow	Labels	Other	
Actual	Body	623	74	71	48	76.3%
	Arrow	42	1562	456	108	72.0%
	Labels	13	257	3353	92	90.3%
	Other	33	146	170	780	69.1%
					TOTAL	80.7%

Table 5.15: Accuracy for multi-way single-stroke classification of statics solutions without equations. Six additional features were added.

increased slightly, from 80.3% to 80.7%. This is discussed more in Section 6.1.3.

5.2 Grouping Results

After the strokes have been classified into different categories, the second step of our algorithm groups the strokes in a given class into individual objects that represent complete shapes. We define two different types of metrics to evaluate the correctness of the shapes. The first characterizes the correctness in terms of the amount of ink (i.e., it is stroke length sensitive) matching. The second is in terms of the number of incorrect strokes. We use two values for the first metric: *Ink Found* and *Ink Extra*. *Ink Found* is the percentage of expected ink (by arc length) that was correctly clustered. *Ink Extra* is the percentage of unexpected ink that was clustered with the object. For the second metric we compute the percentage of shapes that have no errors ("perfect clusters"), those with one or fewer errors, and those with two or fewer errors. Consider, for example, the cluster of strokes shown in Figure 5.4. The expected shape is comprised of three strokes, A, B, and C, which have arc lengths of 100, 200, and 150 units, respectively. If strokes B and C are grouped, but A is left out, and an additional stroke D (which has an arc length of 125 units) is erroneously clustered with B and C, the accuracies for this shape would be: $InkFound = \frac{200+150}{100+200+150} = 78\%$, $InkExtra = \frac{125}{100+200+150} = 28\%$. It would be counted only as a cluster with two or fewer errors. The accuracies reported in the tables below are the average of the accuracies for each shape.

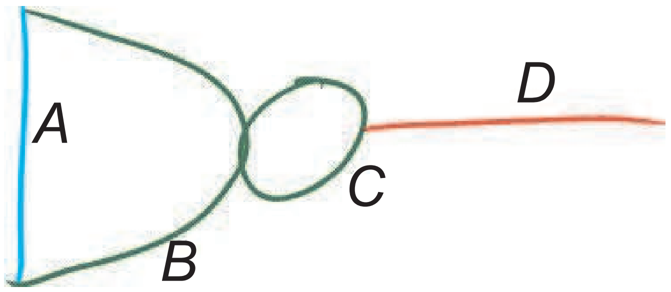


Figure 5.4: An example of how each grouping metric is computed. The two green strokes were correctly grouped together and account for 78% of the ink by arc length, the blue stroke was incorrectly left out of the group, accounting for the last 22% of the ink. The red stroke was incorrectly grouped with the object, and was 28% of the length of the expected shape. The group has a total of two errors (one missing and one extra stroke). The accuracies for this group would be 78% *InkFound* and 28% *InkExtra*. It would be counted as correct only in the category of two or fewer errors.

In order to compute grouping accuracies, we first match expected (hand-labeled) shapes to the machine-generated clusters that our algorithm outputs, using a greedy approach. First, the matching algorithm loops through each expected shape, matching it to the cluster which has the most strokes in common. If there are no previously unmatched clusters which have a stroke in common with the expected shape, it is not matched to any cluster.

In this section we address the following questions related to grouping accuracy:

1. **Grouping Method Comparison:** How well do different pairwise classifiers work?

2. **Effect of User-Specific Training Data:** Can user-specific training data im-

prove accuracy?

3. **Sensitivity to Single-Stroke Errors:** How sensitive is grouping to single-stroke classification errors? When does single-stroke classification help, and hinder?

4. **Effect of Perfect Single-Stroke Classification:** How much improvement can be had by improving the single-stroke classifier?

5. **Sensitivity to the Number of Stroke Classes:** How sensitive is grouping to the number of single-stroke classes?

6. **Iterative Re-Labeling Sensitivity to Seed Value:** How robust is the *Iterative Re-Labeling* method to its seed labels?

7. **Shape to Cluster Matching:** Should clusters be required to have the same class as their expected shapes?

8. **Pairwise Feature Importance:** What are the most important pairwise features?

5.2.1 Grouping Method Comparison

We have implemented and evaluated four different grouping pairwise classifiers. The first method is the *Thresholded Pairwise Classifier* (TPC). This classifier uses thresholds for the minimum distance (T_{JD}) and time gap (T_{JT}) between the strokes to determine whether they should be joined. The values for T_{JD} and T_{JT} are unique to each user in each domain, and were determined via a user-holdout parameter search. We enumerated values for T_{JD} between 50 and 500 himetrics (in increments of 25 himetrics). Similarly, we enumerated values for T_{JT} between 0.5 and 10.0 seconds (in increments of 0.5 seconds), as well as infinity. Setting T_{JT} equal to infinity has the same effect as removing that threshold and using only T_{JD} to classify stroke pairs. The best sets of parameters are determined by their resulting ink found grouping accuracy, and are presented in Table 5.2.1.

Digital Circuits			Family Trees			Statics (No Eqn)			Statics		
User	T_{JD}	T_{JT}	User	T_{JD}	T_{JT}	User	T_{JD}	T_{JT}	User	T_{JD}	T_{JT}
1	150	7.0	1	250	0.5	1	450	0.5	1	50	1.5
2	125	9.5	2	250	0.5	2	450	0.5	2	50	1.5
3	150	7.0	3	250	0.5	3	450	0.5	3	50	1.5
4	125	10.0	4	250	0.5	4	450	0.5	4	50	1.5
5	175	10.0	5	375	0.5	5	450	0.5	5	50	1.5
6	125	9.5	6	250	0.5	6	450	0.5	6	50	1.5
7	125	9.5	7	250	0.5	7	450	0.5	7	50	1.5
8	125	10.0	8	250	0.5	8	450	0.5	8	50	1.5
9	125	9.5	9	250	0.5	11	450	0.5	11	50	1.5
10	150	7.0				12	450	0.5	12	50	1.5
11	125	9.5				13	450	0.5	13	50	1.5
12	125	10.0				14	450	0.5	14	50	1.5
13	175	9.5				15	450	0.5	15	50	1.5
14	150	7.0				16	450	0.5	16	50	1.5
15	150	7.0				17	450	0.5	17	50	1.5
16	150	7.0				18	450	0.5	18	50	1.5
17	150	10.0									
18	150	10.0									
19	150	7.0									
20	150	7.0									
21	150	9.5									
22	125	9.5									
23	125	9.5									
24	125	9.5									

Table 5.16: The best parameters (T_{JD} and T_{JT}) for the *Thresholded Pairwise Classifier* (TPC) in each domain. The parameter sets for each user are found via a user-holdout parameter search.

The three remaining methods use *Inductive Pairwise Classifiers* (IPCs), which are trained using labeled examples. The second pairwise classification method (first IPC) uses only *Join* and *NoJoin* classes (IPC-JnJ). The third method uses the *Minimum Dis-*

tance method for determining *NearJoins* versus *FarJoins* (IPC-MD). The final method uses the *Iterative Re-Labeling* approach to labeling the training instances (IPC-IRL). All four pairwise classifiers rely on chaining to complete the grouping process.

Digital Circuit Sketches

We first look at how these four methods work for digital circuit sketches, with grouping accuracy shown in Figure 5.5. Here we can see that the IPC-MD and IPC-IRL methods performed the best in each category, and were roughly equivalent to each other. Surprisingly, the TPC method performed slightly better than the IPC-JnJ method. This shows that, for this domain, a simple classifier is better when two pairwise classes are used.

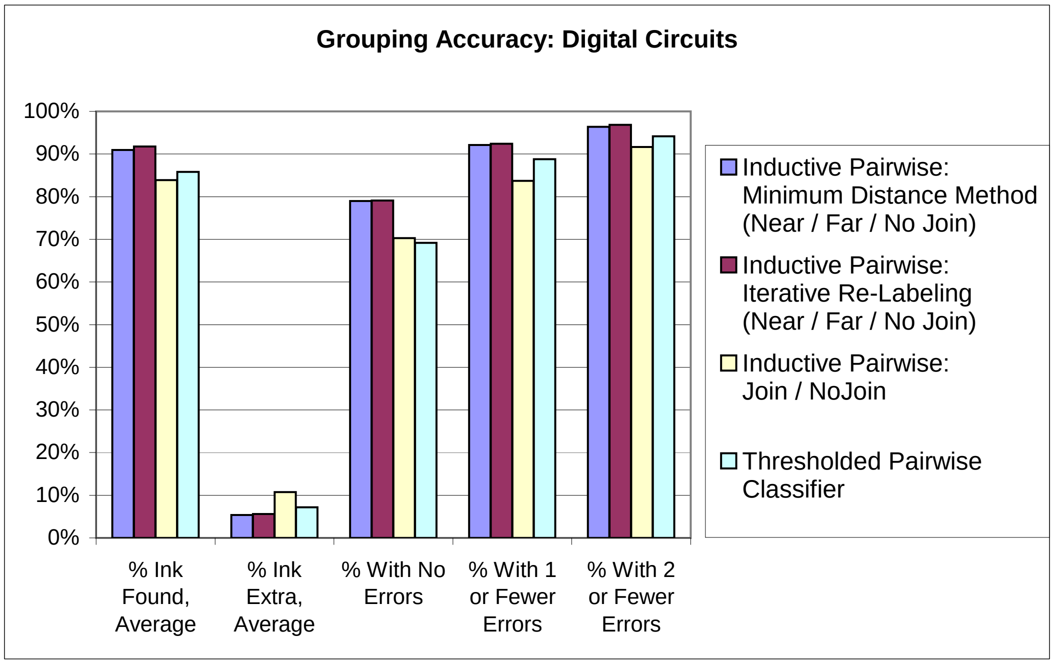


Figure 5.5: A comparison of the four different grouping methods, evaluated on the digital circuit sketch data set.

Table 5.17 presents the grouping accuracies for the IPC-MD method. Here, the accuracies are similar for each class in digital circuit sketches, with an overall accuracy of 90.9% ink found and 79.0% perfect clusters. Using the *Iterative Re-Labeling* method improves overall accuracy to 91.8% ink found and 79.1% perfect clusters (Table 5.18). The other two pairwise classifiers did not perform as well. Table 5.19 shows the accuracy for the IPC-JnJ method. Here, the accuracy for gates goes down slightly, while the wire accuracy goes down significantly. Overall, the grouping accuracy is 83.9% ink found and 70.3% perfect clusters. The TPC method performs similarly, including a large drop in wire accuracy, as shown in Table 5.20. The overall accuracy is 85.8% ink found and 69.2% perfect clusters.

These results show that adding the *NearJoin* versus *FarJoin* distinction is most helpful for grouping wires. It is also helpful for gates, although most gates have enough separation between them that the classifier is not often confused when using only *Join* and *NoJoin* classes. The distinction is not as helpful for labels, likely because label strokes are usually very close to each other if they belong to the same object.

Digital Circuits	Ink: Avg / Shape		Shapes: X Errors or Less		
Class	Found	Extra	0	1	2
Gate	90.7%	4.4%	75.7%	92.7%	97.5%
Wire	90.4%	6.2%	79.7%	89.6%	94.1%
Label	92.8%	5.2%	83.9%	97.3%	99.5%
Overall	90.9%	5.4%	79.0%	92.1%	96.4%

Table 5.17: Grouping accuracy for digital circuit diagrams using the **inductive pairwise classifier** with the **minimum distance** labeling method (IPC-MD). Strokes are classified into gate, wire, and label classes before pairwise classification.

Digital Circuits	Ink: Avg / Shape		Shapes: X Errors or Less		
Class	Found	Extra	0	1	2
Gate	91.8%	2.7%	77.4%	93.5%	97.9%
Wire	91.5%	6.7%	80.1%	90.3%	95.2%
Label	92.5%	8.6%	79.7%	95.3%	98.9%
Overall	91.8%	5.6%	79.1%	92.4%	96.9%

Table 5.18: Grouping accuracy for digital circuit diagrams using the **inductive pairwise classifier** using the **iterative re-labeling** method (IPC-IRL). Strokes are classified into gate, wire, and label classes before pairwise classification.

Digital Circuits	Ink: Avg / Shape		Shapes: X Errors or Less		
Class	Found	Extra	0	1	2
Gate	88.0%	7.6%	71.6%	87.5%	95.7%
Wire	77.2%	14.9%	64.7%	75.4%	85.3%
Label	92.3%	6.6%	81.4%	96.7%	99.3%
Overall	83.9%	10.7%	70.3%	83.7%	91.7%

Table 5.19: Grouping accuracy for digital circuit diagrams using the **inductive pairwise classifier** using only *Join* and *NoJoin* labels (IPC-JnJ). Strokes are classified into gate, wire, and label classes before pairwise classification.

Digital Circuits	Ink: Avg / Shape		Shapes: X Errors or Less		
Class	Found	Extra	0	1	2
Gate	88.8%	8.1%	69.1%	91.8%	96.7%
Wire	80.8%	7.9%	62.6%	82.7%	89.9%
Label	92.4%	3.5%	85.4%	97.8%	99.8%
Overall	85.8%	7.2%	69.2%	88.8%	94.2%

Table 5.20: Grouping accuracy for digital circuit diagrams using the **thresholded pairwise classifier** (TPC). Strokes are classified into gate, wire, and label classes before pairwise classification.

Family Tree Diagram Sketches

Next, we look at how the different grouping methods perform on family tree sketches. An overview of these results is shown in Figure 5.6. Here, *Thresholded Pairwise Classifier* (TPC) method performs the best, followed by the IPC-MD method. Surprisingly, the iterative re-labeling (IPC-IRL) method has noticeably worse perfect cluster accuracy than the minimum distance (IPC-MD) method for the *Inductive Pair-*

wise Classifier. Similar to digital circuits, the IPC-JnJ method performs the worst.

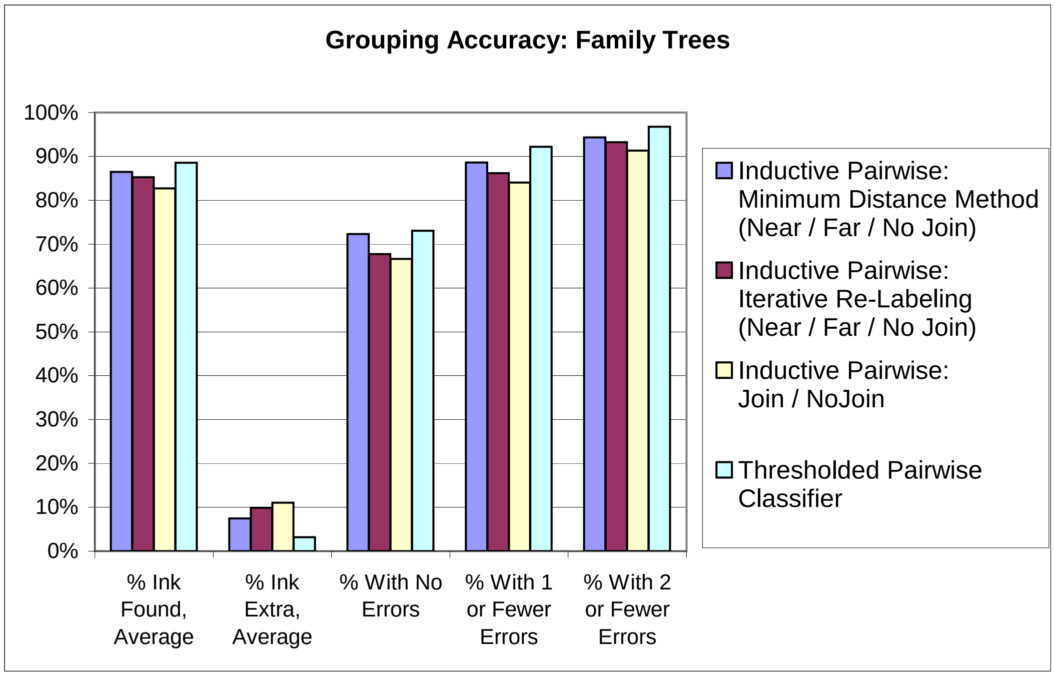


Figure 5.6: A comparison of the four different grouping methods, evaluated on the family tree diagram sketch data set.

The IPC-MD results are presented in Table 5.21; this method has an overall accuracy of 86.5% ink found and 72.3% perfect clusters. The IPC-IRL results, for family trees, are presented in Table 5.22; the iterative re-labeling approach has an overall accuracy of 85.3% ink found and 67.8% perfect clusters. The IPC-JnJ method, with only two pairwise classes (*Join* and *NoJoin*), has an overall accuracy of 82.7% ink found and 66.6% perfect clusters (Table 5.23).

The surprising performance for family tree diagrams is the *Thresholded Pairwise Classifier* (Table 5.24), which not only had the highest overall accuracy, but also the best per-class accuracies. The accuracy of this simple method indicates that the sketches were able to be separated fairly well, either spatially or temporally. For instance, the links between people often overlap, but because there is often very little time gap between strokes of the same shape, a tight time threshold (0.5 seconds) can be used such that overlapping links are not joined.

Family Trees	Ink: Avg / Shape		Shapes: *X* Errors or Less		
Class	Found	Extra	0	1	2
People	91.6%	2.3%	85.3%	93.9%	96.6%
Text	78.2%	17.7%	51.4%	65.4%	74.8%
Link	84.9%	8.5%	68.3%	90.9%	98.0%
Overall	86.5%	7.5%	72.3%	88.6%	94.4%

Table 5.21: Grouping accuracy for family tree diagrams using the **inductive pairwise classifier** using the **minimum distance** labeling method (IPC-MD). Strokes are classified into people, text, and link classes before pairwise classification.

Family Trees	Ink: Avg / Shape		Shapes: *X* Errors or Less		
Class	Found	Extra	0	1	2
People	90.7%	3.4%	81.9%	91.5%	95.9%
Text	72.0%	26.4%	37.4%	52.3%	65.4%
Link	84.8%	10.2%	65.5%	91.4%	98.7%
Overall	85.3%	9.9%	67.8%	86.2%	93.2%

Table 5.22: Grouping accuracy for family tree diagrams using the **inductive pairwise classifier** using the **iterative re-labeling** method (IPC-IRL). Strokes are classified into people, text, and link classes before pairwise classification.

Family Trees	Ink: Avg / Shape		Shapes: *X* Errors or Less		
Class	Found	Extra	0	1	2
People	91.2%	4.4%	82.3%	92.8%	95.9%
Text	62.2%	34.3%	29.0%	43.0%	55.1%
Link	82.0%	9.7%	65.2%	88.7%	97.7%
Overall	82.7%	11.1%	66.6%	84.1%	91.3%

Table 5.23: Grouping accuracy for family tree diagrams using the **inductive pairwise classifier** using only *Join* and *NoJoin* labels (IPC-JnJ). Strokes are classified into people, text, and link classes before pairwise classification.

Family Trees	Ink: Avg / Shape		Shapes: X Errors or Less		
Class	Found	Extra	0	1	2
People	92.6%	1.3%	83.6%	95.9%	99.3%
Text	78.2%	0.5%	51.4%	68.2%	78.5%
Link	88.3%	5.2%	71.0%	96.0%	99.7%
Overall	88.6%	3.2%	73.0%	92.2%	96.7%

Table 5.24: Grouping accuracy for family tree diagrams using the **thresholded pairwise classifier** (TPC). Strokes are classified into people, text, and link classes before pairwise classification.

Statics Solutions Sketches Without Equations

Figure 5.7 shows the grouping results for statics sketches without equations. In this domain, the *Thresholded Pairwise Classifier* (TPC) performed the best, slightly ahead of both the *Inductive Pairwise Classifier* using the *Minimum Distance* labeling method (IPC-MD), and the *Iterative Re-Labeling* method (IPC-IRL). The IPC-JnJ method again performed poorly, having the worst accuracy according to all five metrics.

Detailed results for each of the four methods are presented in Tables 5.25-5.28. The IPC-MD method has an overall accuracy of 83.9% ink found and 61.6% perfect clusters, per-class accuracies are shown in Table 5.25. As shown in Table 5.26, the IPC-IRL method performs about the same as the IPC-MD method. The overall accuracy is 83.6% ink found and 61.6% perfect clusters. Using the IPC-JnJ method, the overall accuracy is 79.7% ink found and 55.3% perfect clusters, as shown in Table 5.27. The best performing method for this domain is the TPC method, which has an overall accuracy of 84.3% ink found and 66.2% perfect clusters, as shown in Table 5.28.

Looking at the per-class accuracies of the IPC-MD versus TPC methods, we see that the IPC-MD method performs significantly better on bodies, while being worse for

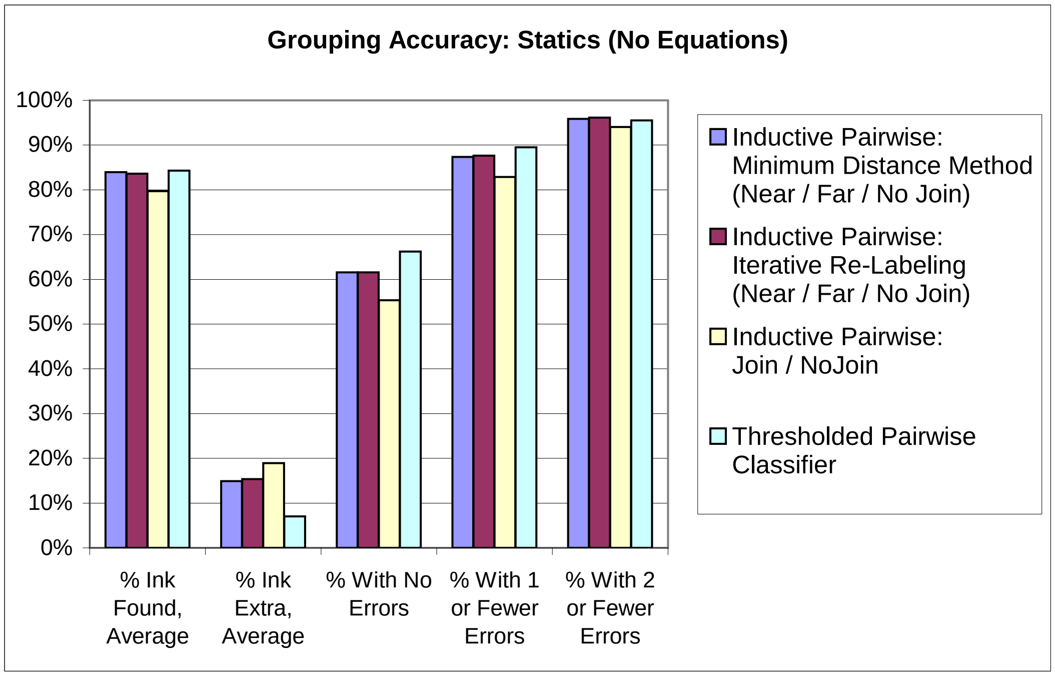


Figure 5.7: A comparison of the four different grouping methods, evaluated on the statics solutions (without equations) data set.

the other classes. The decrease in accuracy for bodies using the TPC method indicates that users paused between drawing the various strokes of bodies, and the pauses were longer than the threshold, T_{JT}. Also of note, there is considerably less ink extra using the TPC – an indication that this method groups less aggressively.

Statics Solutions Class	Ink: Avg / Shape		Shapes: X Errors or Less		
	Found	Extra	0	1	2
Body	91.3%	2.8%	68.3%	85.6%	93.4%
Arrow	80.7%	9.2%	56.6%	89.1%	97.9%
Label	87.3%	8.3%	64.5%	87.4%	96.5%
Other	78.1%	49.5%	60.8%	84.6%	91.3%
Overall	83.9%	14.9%	61.6%	87.4%	95.8%

Table 5.25: Grouping accuracy for statics solutions without equations using the **inductive pairwise classifier** using the **minimum distance** labeling method (IPC-MD). Strokes are classified into arrow, body, text, and "other" classes before pairwise classification.

Statics Solutions	Ink: Avg / Shape		Shapes: X Errors or Less		
Class	Found	Extra	0	1	2
Body	90.5%	3.8%	66.8%	85.3%	93.1%
Arrow	79.8%	9.0%	54.9%	88.3%	98.3%
Label	87.5%	9.1%	65.7%	88.2%	96.3%
Other	77.8%	50.2%	62.1%	86.3%	93.1%
Overall	83.6%	15.3%	61.6%	87.7%	96.2%

Table 5.26: Grouping accuracy for statics solutions without equations using the **inductive pairwise classifier** using the **iterative re-labeling** method (IPC-IRL). Strokes are classified into arrow, body, text, and “other” classes before pairwise classification.

Statics Solutions	Ink: Avg / Shape		Shapes: X Errors or Less		
Class	Found	Extra	0	1	2
Body	85.1%	10.0%	58.6%	76.8%	88.1%
Arrow	77.8%	10.2%	52.4%	84.1%	96.6%
Label	82.2%	16.6%	55.7%	81.3%	93.5%
Other	74.6%	47.6%	58.6%	87.4%	93.3%
Overall	79.7%	18.9%	55.3%	82.9%	94.0%

Table 5.27: Grouping accuracy for statics solutions without equations using the **inductive pairwise classifier** using only *Join* and *NoJoin* labels (IPC-JnJ). Strokes are classified into arrow, body, text, and “other” classes before pairwise classification.

Statics Solutions	Ink: Avg / Shape		Shapes: X Errors or Less		
Class	Found	Extra	0	1	2
Body	71.3%	0.0%	56.4%	74.3%	82.8%
Arrow	84.5%	2.9%	61.0%	94.2%	98.4%
Label	87.3%	3.9%	70.4%	90.3%	97.2%
Other	83.3%	27.2%	71.7%	86.1%	92.1%
Overall	84.3%	7.1%	66.2%	89.5%	95.5%

Table 5.28: Grouping accuracy for statics solutions without equations using the **thresholded pairwise classifier** (TPC). Strokes are classified into arrow, body, text, and “other” classes before pairwise classification.

Complete Statics Solutions Sketches

The grouping results for complete statics sketches are shown in Figure 5.8. Here, only three methods are compared: IPC-MD, IPC-JnJ, and TPC. The IPC-IRL method was not tested on this data set, mainly because it did not perform better than the IPC-MD method for other domains. It also took significantly longer to train and evaluate because of the multiple iterations – a problem for a data set as large as this. Of the three methods considered, the IPC-MD method performed much better than the other two methods, and again the IPC-JnJ method performed the worst.

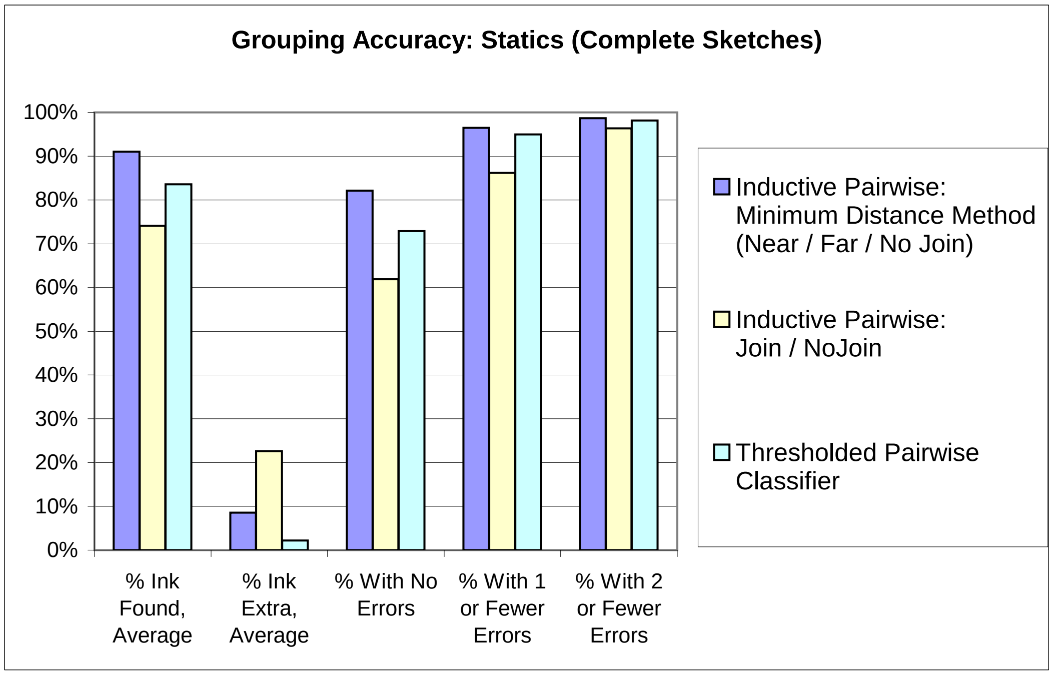


Figure 5.8: A comparison of three different grouping methods, evaluated on the complete statics solutions data set.

Detailed results for the evaluation of these three methods can be found in Tables 5.29-5.31. The overall accuracy for the *Minimum Distance* method is 91.0% ink found with 82.1% perfect clusters, as shown in Table 5.29. Using only two classes (*Join* and *NoJoin*) with the *Inductive Pairwise Classifier* (IPC-JnJ), the overall accuracy is 74.1% ink found and 61.9% perfect clusters, as shown in Table 5.30. The *Thresholded Pairwise Classifier* (TPC) has an overall accuracy of 83.6% ink found and 72.9% perfect clusters, as shown in Table 5.31.

The overall accuracy for the IPC-MD method is largely determined by the text accuracy, because there are many more text objects than anything else. The accuracy for the other three classes is particularly low in terms of the percentage of perfect clusters. Despite the lower perfect clusters accuracy, the percentage of ink found is reasonably good for bodies, and is not terribly low for arrows and "other." There is a large amount of extra ink for clusters in the "other" class; this is most likely due to the way we match expected shapes to clusters. Clusters are matched according to the number of strokes, which can be quite high for shapes such as dotted leader lines. When long strokes (by arc length) are incorrectly clustered with these short dotted lines, they can show up in the results as large amounts of extra ink for "other" shapes. For example, it is not uncommon for ink extra to be greater than 300%.

Statics Solutions	Ink: Avg / Shape		Shapes: X Errors or Less		
Class	Found	Extra	0	1	2
Body	89.0%	3.8%	62.6%	82.8%	92.3%
Arrow	78.4%	5.4%	51.6%	92.1%	98.5%
Text	93.1%	6.8%	86.7%	97.9%	99.2%
Other	80.3%	47.7%	68.0%	85.8%	92.1%
Overall	91.0%	8.5%	82.1%	96.5%	98.7%

Table 5.29: Grouping accuracy for statics solutions using the **inductive pairwise classifier** using the **minimum distance** labeling method (IPC-MD). Strokes are classified into arrow, body, text, and "other" classes before pairwise classification.

Statics Solutions	Ink: Avg / Shape		Shapes: X Errors or Less		
Class	Found	Extra	0	1	2
Body	86.1%	10.5%	56.4%	72.4%	85.9%
Arrow	59.6%	4.7%	35.5%	78.3%	97.7%
Text	77.1%	26.0%	66.3%	87.7%	96.9%
Other	43.7%	2.3%	37.6%	80.5%	89.1%
Overall	74.1%	22.6%	61.9%	86.2%	96.4%

Table 5.30: Grouping accuracy for statics solutions using the **inductive pairwise classifier** using only *Join* and *NoJoin* labels (IPC-JnJ). Strokes are classified into arrow, body, text, and 'other' classes before pairwise classification.

Statics Solutions	Ink: Avg / Shape		Shapes: X Errors or Less		
Class	Found	Extra	0	1	2
Body	76.6%	0.2%	58.6%	73.9%	82.5%
Arrow	57.5%	1.6%	31.2%	80.0%	97.9%
Text	89.0%	2.3%	79.8%	98.0%	99.2%
Other	40.6%	1.4%	37.6%	78.1%	85.1%
Overall	83.6%	2.2%	72.9%	95.0%	98.1%

Table 5.31: Grouping accuracy for complete statics solutions using the **thresholded pairwise classifier** (TPC). Strokes are classified into people, text, and link classes before pairwise classification.

5.2.2 Effect of User-Specific Training Data

In this section, we evaluate grouping accuracy for the digital circuit and complete statics solution domains when user-specific training data is included by performing sketch-holdouts, rather than user-holdouts. We use the *Inductive Pairwise Classifier* along with the *Minimum Distance* labeling method (IPC-MD) because it consistently performed well in the evaluations from the previous section. Also, the inductive method can be influenced by the small number of training examples that a user adds, especially when using an AdaBoosted decision tree. The results of these evaluations for digital circuits and complete statics sketches are shown in Figures 5.9 and 5.10, respectively. The

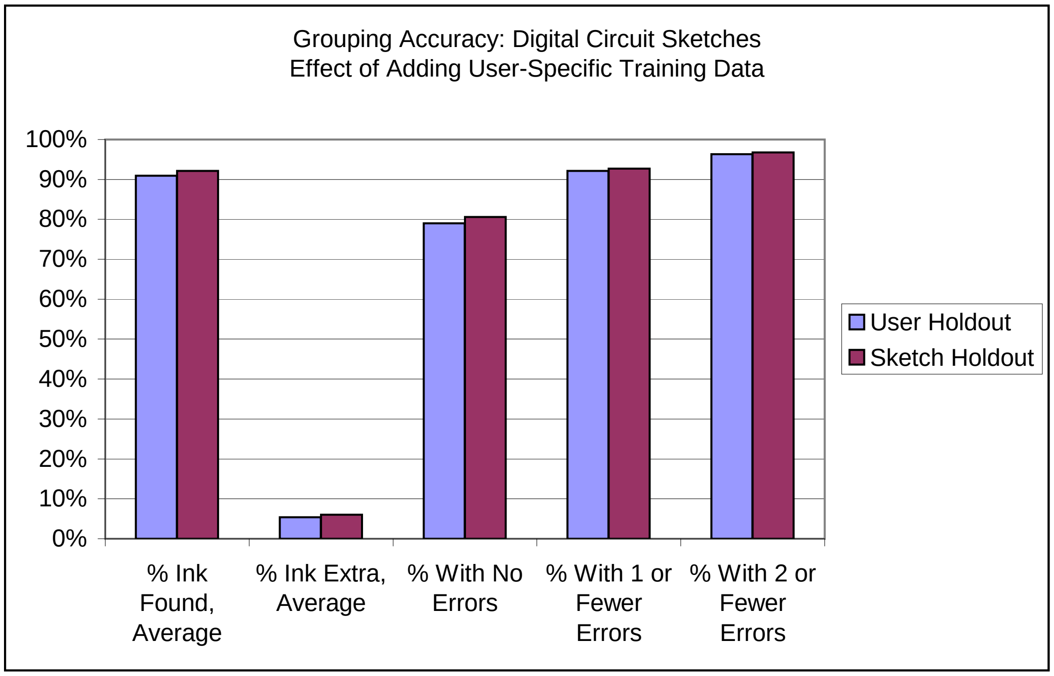


Figure 5.9: A comparison of grouping accuracy when using no user-specific data (*User Holdout*) versus including user-specific data (*Sketch Holdout*), for digital circuit sketches. The IPC-MD grouping method is used for both cases. Strokes are classified into gate, wire, and label classes before pairwise classification. Both the single-stroke classifier and the pairwise classifier used user-specific data.

single-stroke classifiers used for this analysis were also trained using sketch-holdouts.

For digital circuits, the overall accuracy increased from 90.9% to 92.0% ink found and from 79.0% to 80.6% perfect clusters. The detailed results for the inclusion of user-specific training data is presented in Table 5.32; detailed grouping results without user specific data is presented in Table 5.17. For this domain, including user-specific training data does help, however the difference is not large.

For complete statics solution sketches, user-specific data has a similar effect. The overall accuracies increase slightly, from 91.0% to 92.0% ink found, and from 82.1% to 83.8% perfect clusters. Detailed grouping results showing the effects of user-specific

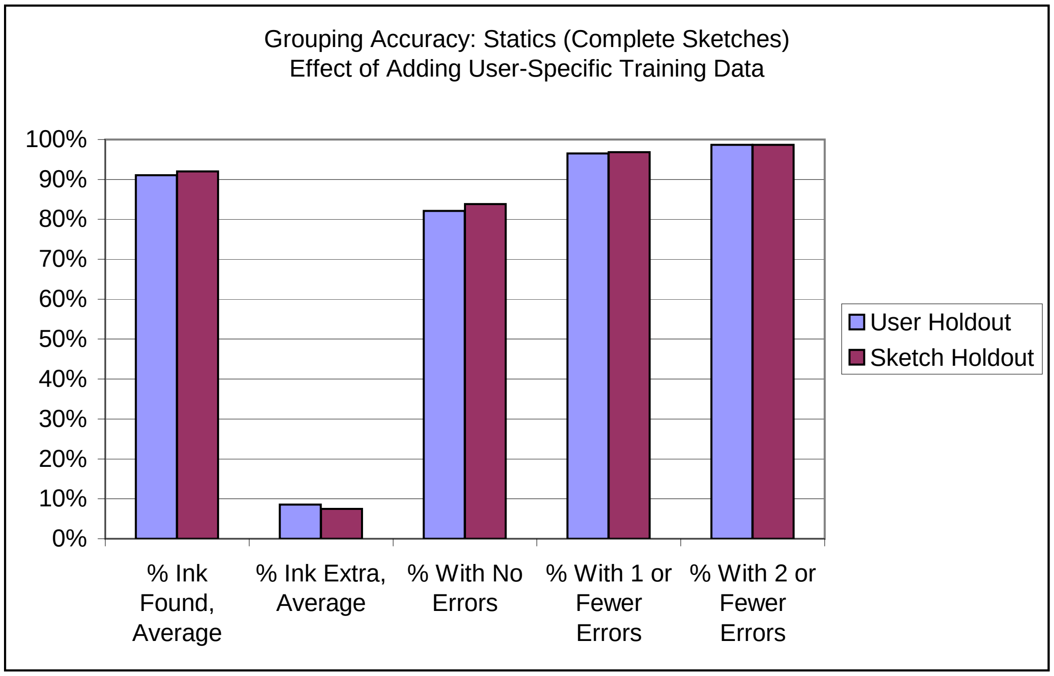


Figure 5.10: A comparison of grouping accuracy when using no user-specific data (*User Holdout*) versus including user-specific data (*Sketch Holdout*), for complete statics solution sketches. The IPC-MD grouping method is used for both cases. Strokes are classified into body, arrow, text, and "other" classes before pairwise classification. Both the single-stroke classifier and the pairwise classifier used user-specific data.

Digital Circuits Class	Ink: Avg / Shape		Shapes: X Errors or Less		
	Found	Extra	0	1	2
Gate	92.4%	3.3%	79.7%	94.4%	98.1%
Wire	91.1%	8.3%	79.2%	89.8%	94.6%
Label	94.1%	5.9%	85.6%	96.4%	99.5%
Overall	92.1%	6.0%	80.6%	92.7%	96.8%

Table 5.32: Grouping accuracy for digital circuit diagrams using the **inductive pairwise classifier** using the **Minimum Distance** labeling method. Strokes are classified into gate, wire, and label classes before pairwise classification. Both single-stroke classifiers and pairwise classifiers used user-specific data.

data are presented in Table 5.33; results without user-specific training data are presented in Table 5.29.

The increases in accuracy in both the digital circuits and statics domains are

Statics Solutions	Ink: Avg / Shape		Shapes: X Errors or Less		
Class	Found	Extra	0	1	2
Body	88.0%	6.9%	61.0%	82.5%	90.5%
Arrow	81.6%	3.8%	57.1%	94.1%	98.7%
Text	93.9%	7.4%	88.2%	98.0%	99.2%
Other	81.0%	16.0%	68.4%	86.8%	91.4%
Overall	92.0%	7.5%	83.8%	96.9%	98.7%

Table 5.33: Grouping accuracy for statics solutions sketches using the **inductive pairwise classifier** using the **Minimum Distance** labeling method. Strokes are classified into body, arrow, text, and "other" classes before pairwise classification. Both single-stroke classifiers and pairwise classifiers used user-specific data.

most likely attributable to an increase in single-stroke classification accuracy. As was shown in Section 5.1.3, the digital circuits single-stroke classifier accuracy increased from 93.6% to 94.7% overall. Similarly for statics solutions, single-stroke accuracy rose from 88.8% to 90.6%.

5.2.3 Sensitivity to Single-Stroke Errors

Errors from the single-stroke classification process are unrecoverable during grouping. In this section we explore the sensitivity of the overall grouping accuracy to simulated single-stroke classification errors. To do this, we artificially created single-stroke classification errors. Some fraction of strokes are randomly selected and assigned a randomly selected incorrect label; the other strokes are labeled correctly. The program then uses a pairwise classifier, trained as before in a user-holdout fashion.

When calculating the grouping accuracy for this technique, we require that a cluster have the same class as the expected shape. For instance, if two strokes were grouped together to form a valid arrow, but the strokes were classified (artificially) as text strokes, the cluster would not be matched to the expected shape. We examine the effects of the different matching schemes more in depth in Section 5.2.7. Because the incorrect labels are assigned randomly, and to random strokes, re-running this analysis gives slightly different results. However, for large data sets such as the digital circuits and complete statics solutions, the effects of this random nature are less pronounced than in the other domains. We used the *Inductive Pairwise Classifier* with *Minimum Distance* method for this evaluation.

Digital Circuits

Figure 5.11 shows the accuracy (ink found and perfect clusters) for digital circuits as a function of single-stroke classification errors. As the single-stroke classification accuracy rises, the percentage of ink found accuracy rises in a nearly direct and linear fashion. This is to be expected, as the strokes begin to have the correct label they can be grouped correctly, and their clusters will have more ink correctly found, even if the cluster is not complete. However, "perfect clusters" is an all-or-nothing accuracy, which does not change so directly. For example, with a three stroke shape, two of the three strokes in the cluster may be grouped, but it will not be counted as correct until all three strokes are included in the cluster. This is reflected by the exponential nature of the curve. If most of the shapes in the sketches have multiple strokes, the curve will look similar to this one. However, if most of the shapes are single strokes, the curve will be almost linear. Our single-stroke classifier typically performs in the high 80% to mid 90% range. In this range for Figure 5.11 the perfect cluster accuracy rises at a very rapid rate with increasing single-stroke accuracy. This indicates that even small improvements in single-stroke classification accuracy can yield large increases in perfect cluster accuracy.

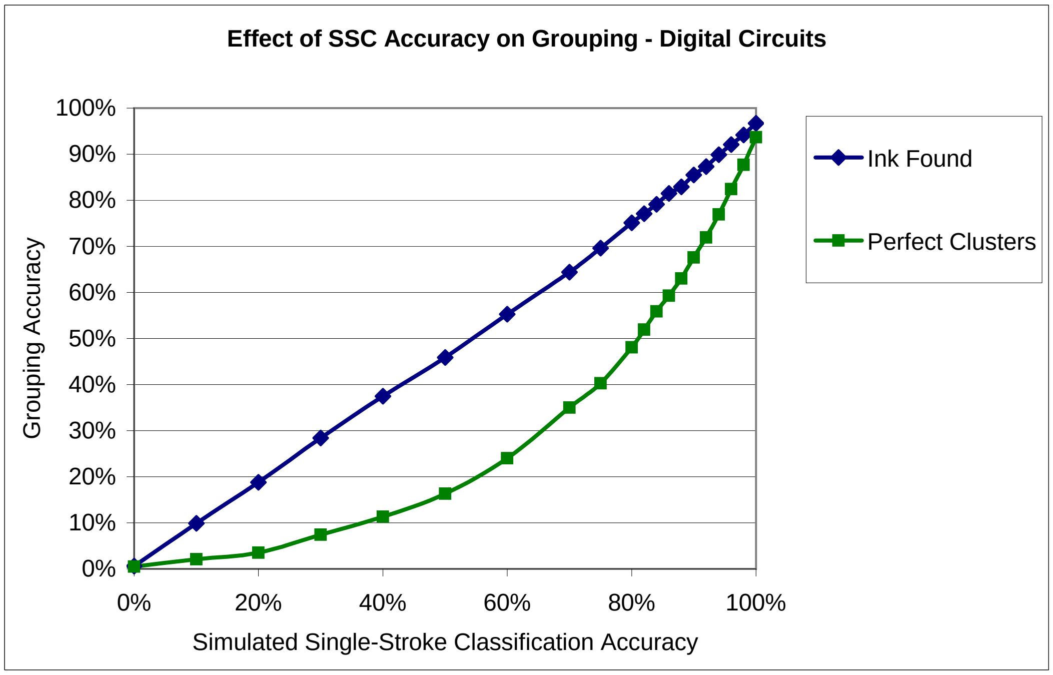


Figure 5.11: The effect of simulated single-stroke classification accuracy on grouping accuracy for digital circuits (using IPC-MD).

Figure 5.12 shows the ink found accuracy from Figure 5.11, while adding a red line showing the accuracy if no single-stroke classification is done at all (i.e., all strokes are part of the same class), and a green line showing accuracy *with* our single-stroke classifier. Where the red line crosses the simulated ink found accuracy is the theoretical point at which single-stroke classification becomes helpful. If the single-stroke classifier is unable to achieve at least 86%, in this case it would be better to **not** classify strokes before doing pairwise classification. Detailed results for grouping without single-stroke classification are presented in Table 5.34. Without single-stroke classification, a single pairwise classifier is used to identify strokes that should be joined. This pairwise classifier is trained using all stroke pairs, regardless of stroke type.

Using single-stroke classification increases the ink found accuracy by 9 percentage points, shown with a green line in Figure 5.12. In this domain, it is quite advantageous to perform single-stroke classification before grouping.

Digital Circuits	Ink: Avg / Shape		Shapes: X Errors or Less		
Class	Found	Extra	0	1	2
Gate	85.9%	25.4%	57.9%	82.3%	92.6%
Wire	75.3%	5.8%	60.4%	83.5%	91.9%
Label	90.2%	5.2%	87.4%	94.3%	98.7%
Overall	81.9%	12.9%	64.4%	85.1%	93.4%

Table 5.34: Grouping accuracy for digital circuit diagrams using the inductive pairwise classifier and minimum distance labeling method, **without** single-stroke classification.

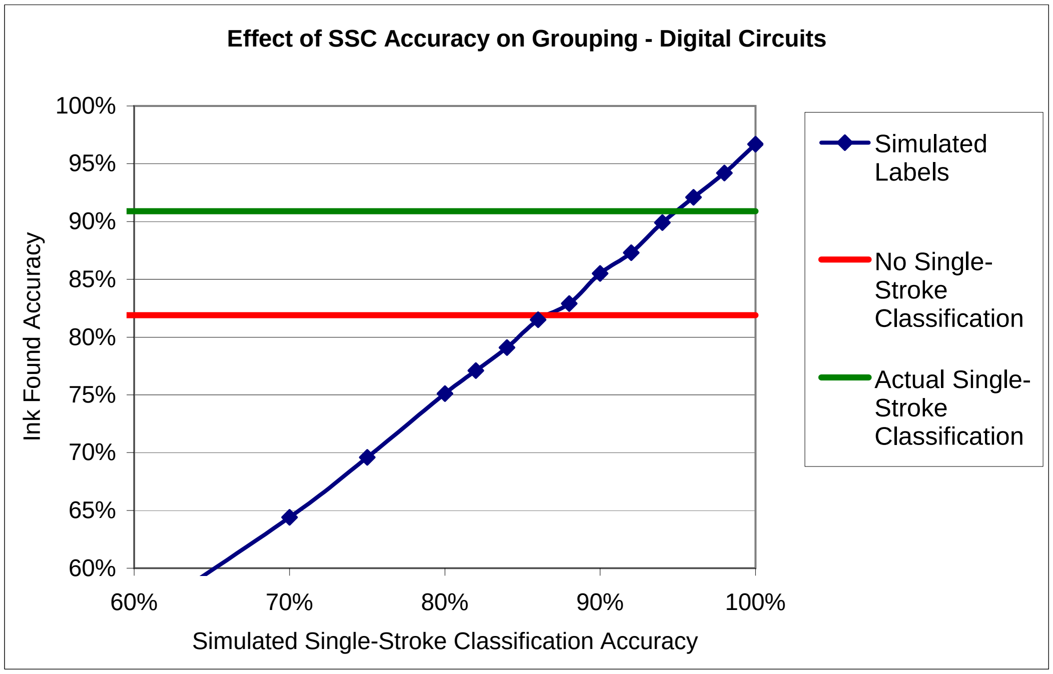


Figure 5.12: The effect of single-stroke classification accuracy (artificially generated) on ink found accuracy for digital circuits (using IPC-MD). The red line represents the ink found accuracy if **no** single-stroke classification is performed. The green line represents the percentage of ink found when using multi-way single-stroke classification.

Figure 5.13 is similar to 5.12, except that grouping accuracy is represented in terms of perfect clusters, rather than ink found. Again, performing single-stroke classification significantly improves grouping accuracy.

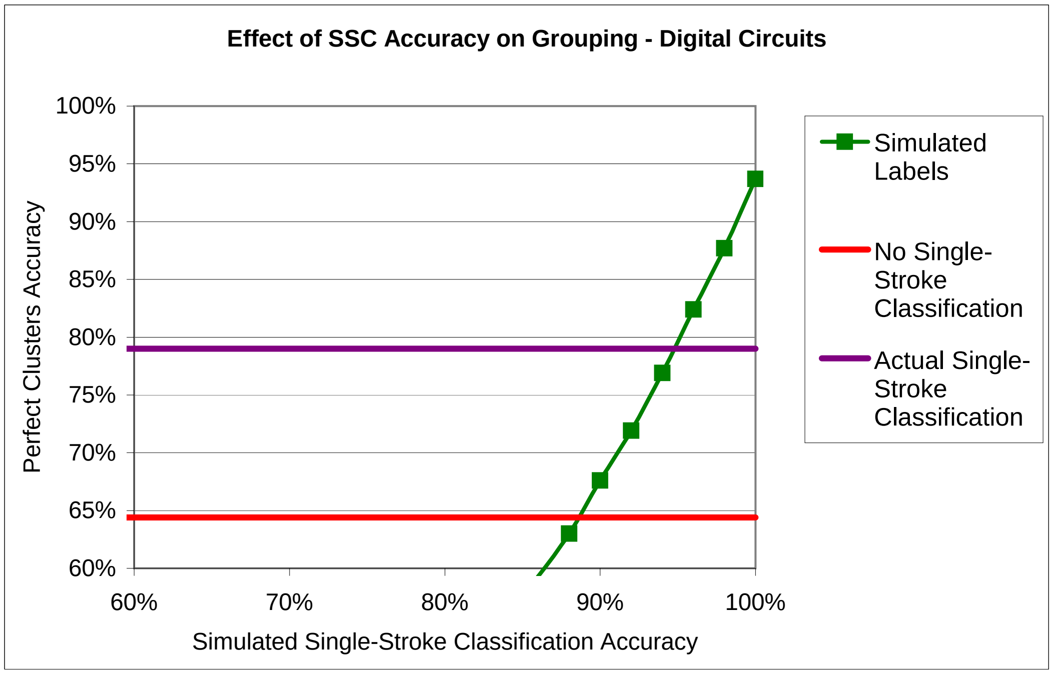


Figure 5.13: The effect of single-stroke classification accuracy (artificially generated) on perfect cluster accuracy for digital circuits (using IPC-MD). The red line represents the ink found accuracy if **no** single-stroke classification is performed. The purple line represents the percentage of ink found when using multi-way single-stroke classification.

Family Tree Diagrams

Next, we consider grouping accuracy as a function of single-stroke accuracy for family tree diagrams. Figure 5.14 shows this relationship for the percentages of ink found and perfect clusters. Here, the shape of these curves resembles that for digital circuits, however these appear noisier. The increase in noise is likely due to a much smaller data set that is more sensitive to random variations in incorrectly labeling strokes.

Figure 5.15 shows part of the ink found curve from Figure 5.14, along with

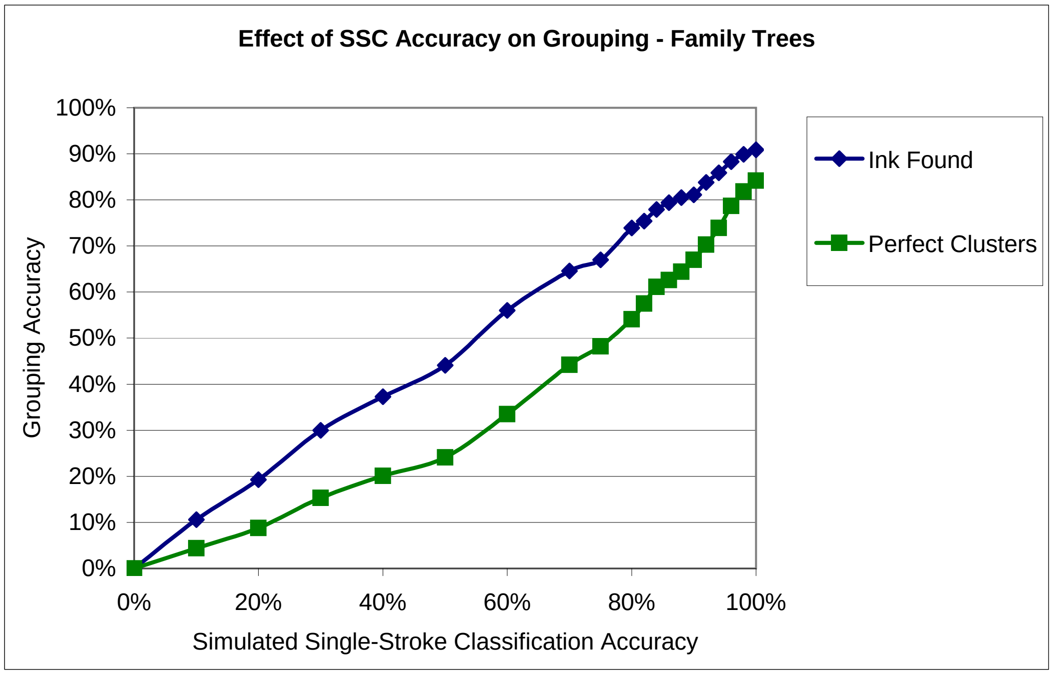


Figure 5.14: The effect of single-stroke classification accuracy (artificially generated) on grouping accuracy for family tree diagrams (using IPC-MD).

two lines. The red line represents ink found accuracy if no single-stroke classification is performed, while the green line represents accuracy using our single-stroke classifier. For family tree diagrams, the increase in accuracy is less pronounced than for digital circuits, likely due to the fact that the single-stroke classifier is not as accurate in this domain. For grouping without single-stroke classification, detailed results are presented in Table 5.35.

The ink found accuracy without single-stroke classification crosses the simulated ink found accuracy curve at around 90% simulated single-stroke accuracy. This indicates that our classifier would need to perform better than 90% to be helpful, however our actual single-stroke accuracy is 87.8%. Despite this, our grouping accuracy with our single-stroke classifier is higher than without it. This means that errors made by the single-stroke classifier are not the same as the simulated errors.

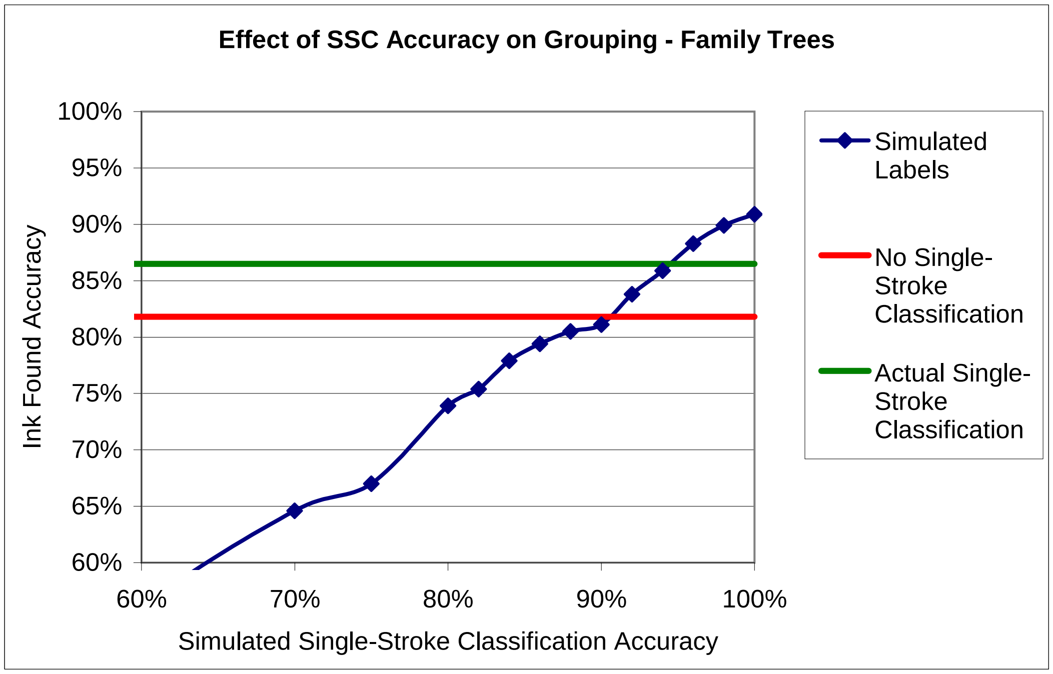


Figure 5.15: The effect of single-stroke classification accuracy (artificially generated) on ink found accuracy for family tree diagrams (using IPC-MD). The red line represents the ink found accuracy if **no** single-stroke classification is performed. The green line represents the percentage of ink found when using multi-way single-stroke classification.

Family Trees	Ink: Avg / Shape		Shapes: X Errors or Less		
Class	Found	Extra	0	1	2
People	83.3%	7.0%	77.8%	91.1%	94.2%
Text	83.6%	68.1%	51.4%	64.5%	71.0%
Link	80.2%	14.6%	66.8%	87.7%	98.5%
Overall	81.8%	19.0%	68.8%	85.8%	93.2%

Table 5.35: Grouping accuracy for family tree diagrams using the inductive pairwise classifier and minimum distance labeling method, **without** using single-stroke classification.

Perfect cluster accuracy is shown in Figure 5.16. Similar to the ink found accuracy, performing single-stroke classification provides a meaningful, yet not particularly

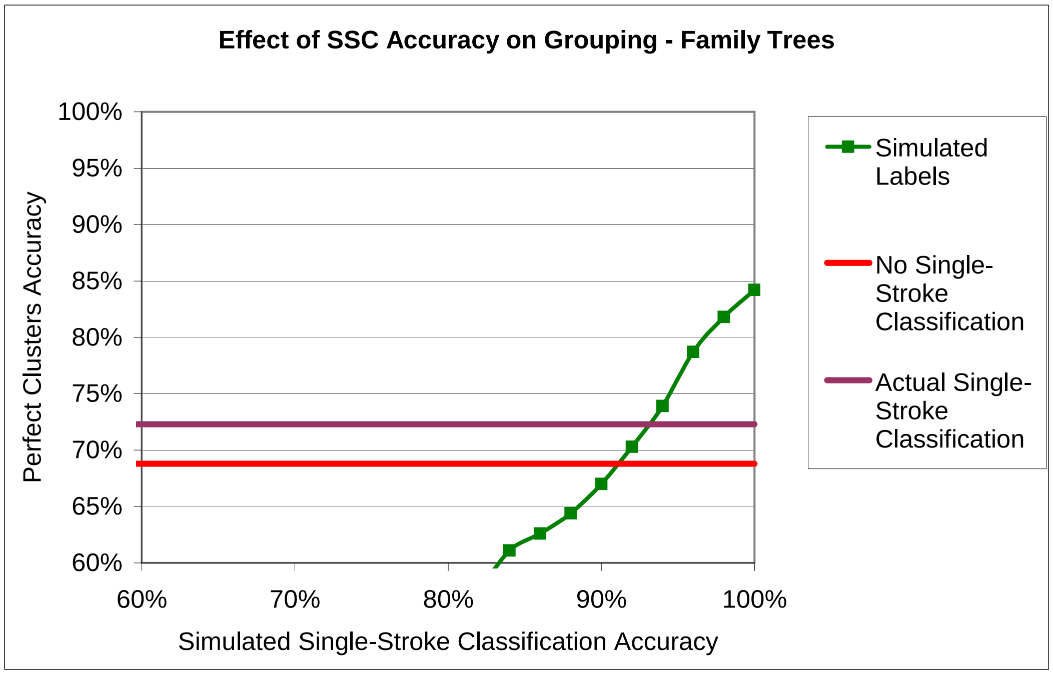


Figure 5.16: The effect of single-stroke classification accuracy (artificially generated) on perfect cluster accuracy for family tree diagrams (using IPC-MD). The red line represents the ink found accuracy if **no** single-stroke classification is performed. The purple line represents the percentage of ink found when using multi-way single-stroke classification.

large benefit for family trees.

Statics Solutions Without Equations

For statics solutions without equations, grouping accuracy versus simulated single-stroke classification accuracy is shown in Figure 5.17. Here, there is a similar shape to that for digital circuits (Figure 5.11).

The comparison of ink found accuracy is shown in Figure 5.18. In this case, there is a very small increase when using single-stroke classification. The details for grouping accuracy without single-stroke classification are presented in Table 5.36. In

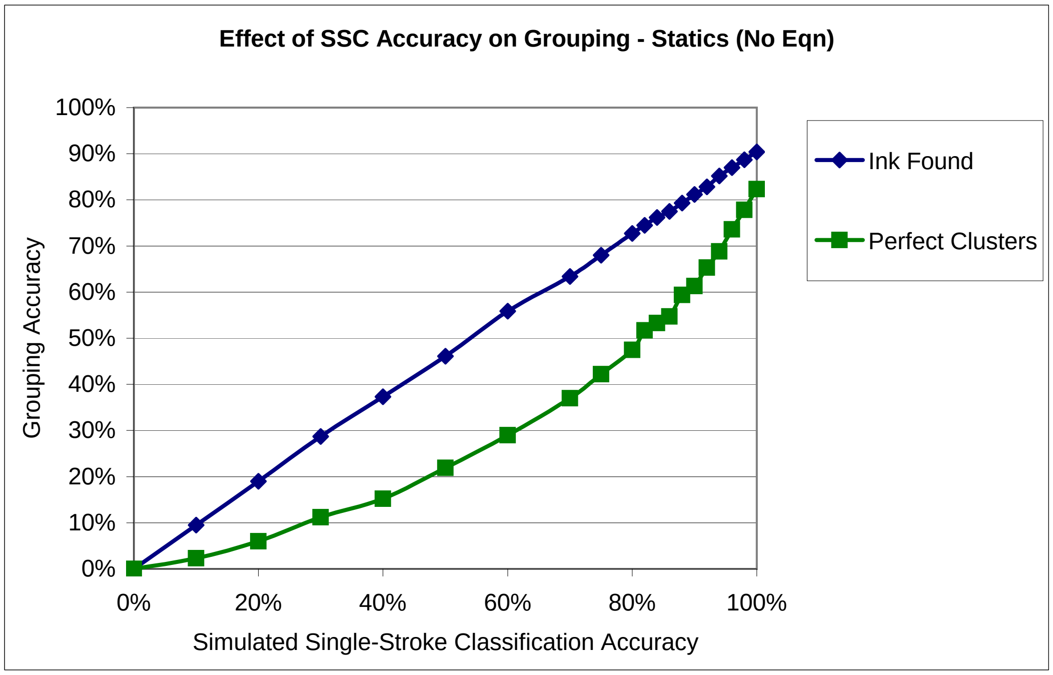


Figure 5.17: The effect of single-stroke classification accuracy (artificially generated) on grouping accuracy for statics solutions without equations (using IPC-MD).

comparison to the accuracies when using single-stroke classification (Table 5.25), ink found is roughly equivalent. There is much more ink extra (overall) when *not* using single-stroke classification due to arrows and "other" clusters having large amounts of extra ink.

Statics Solutions	Ink: Avg / Shape		Shapes: X Errors or Less		
Class	Found	Extra	0	1	2
Body	84.6%	2.3%	64.9%	81.2%	87.8%
Arrow	81.4%	22.4%	63.7%	87.5%	97.8%
Label	88.9%	7.5%	76.7%	88.3%	97.6%
Other	72.4%	169.1%	50.6%	82.2%	89.8%
Overall	83.3%	38.6%	67.0%	86.4%	95.5%

Table 5.36: Grouping accuracy for statics solutions without equations using the inductive pairwise classifier and minimum distance labeling method, **without** using single-stroke classification.

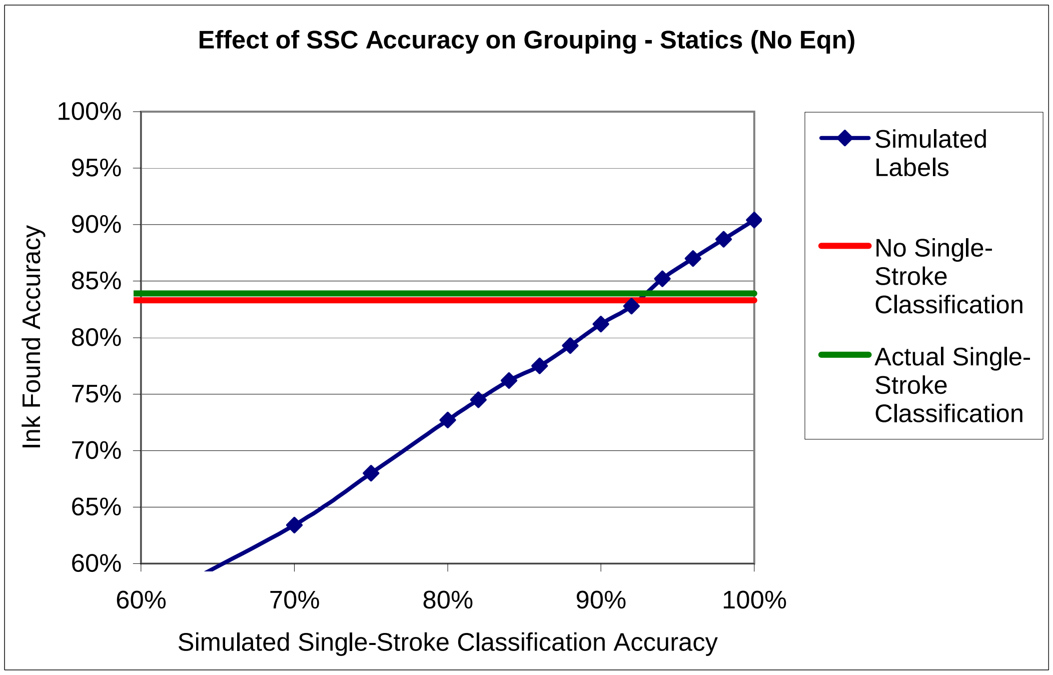


Figure 5.18: The effect of single-stroke classification accuracy (artificially generated) on ink found accuracy for statics solutions without equations (using IPC-MD). The red line represents the ink found accuracy if **no** single-stroke classification is performed. The green line represents the percentage of ink found when using multi-way single-stroke classification.

As shown in Figure 5.19, the percentage of perfect clusters actually improves when not performing single-stroke classification, mostly due to an increase in perfect "label" clusters. At the same time, the perfect cluster accuracy goes down for arrows and "other." Also, the accuracy for clusters with one or fewer errors is higher when using single-stroke classification; this indicates that most erroneous clusters when using single-stroke classification have only a single-stroke wrong. The increase in perfect "label" clusters is most likely due to many labels having multiple strokes, and a single label stroke is often confused with an arrow stroke by the single-stroke classifier. This single wrong stroke prevents the cluster from being perfect, so removing classifications actually

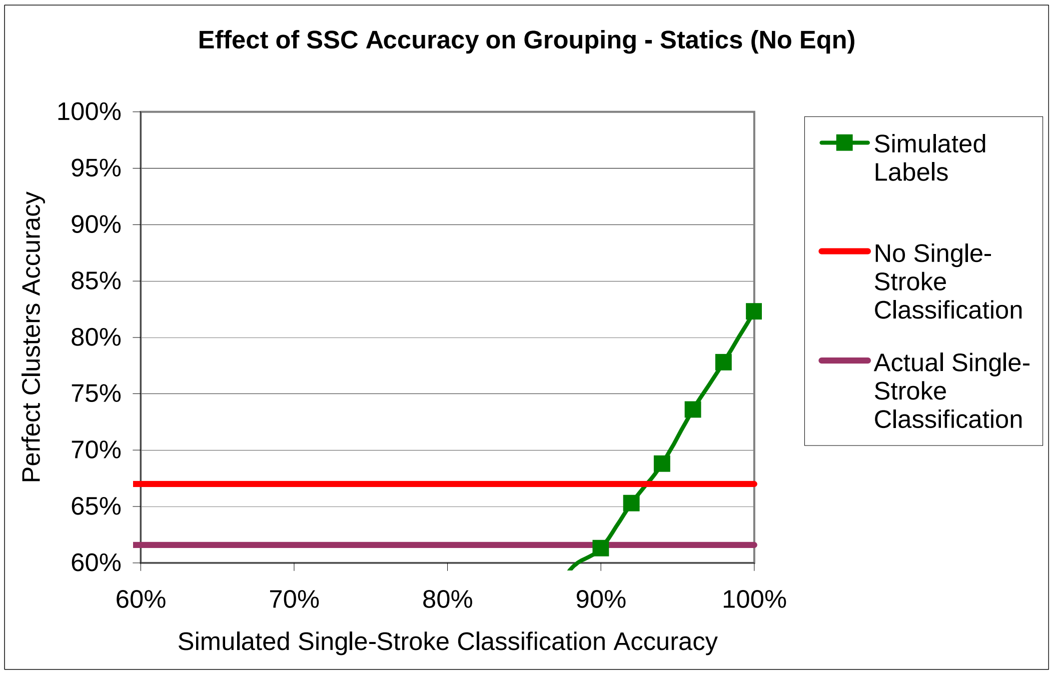


Figure 5.19: The effect of single-stroke classification accuracy (artificially generated) on perfect cluster accuracy for statics solutions without equations (using IPC-MD). The red line represents the ink found accuracy if **no** single-stroke classification is performed. The purple line represents the percentage of ink found when using multi-way single-stroke classification.

makes these labels more likely to be perfect.

Complete Statics Solutions

Figure 5.20 shows the relationship between grouping accuracy and simulated single-stroke classification accuracy for complete statics solution sketches. These sketches have a large number of single-stroke text shapes, making the perfect clusters curve much more linear than in other domains.

Figure 5.21 shows that there is very little difference in ink found accuracy between using single-stroke classification and not. Table 5.37 shows detailed results for

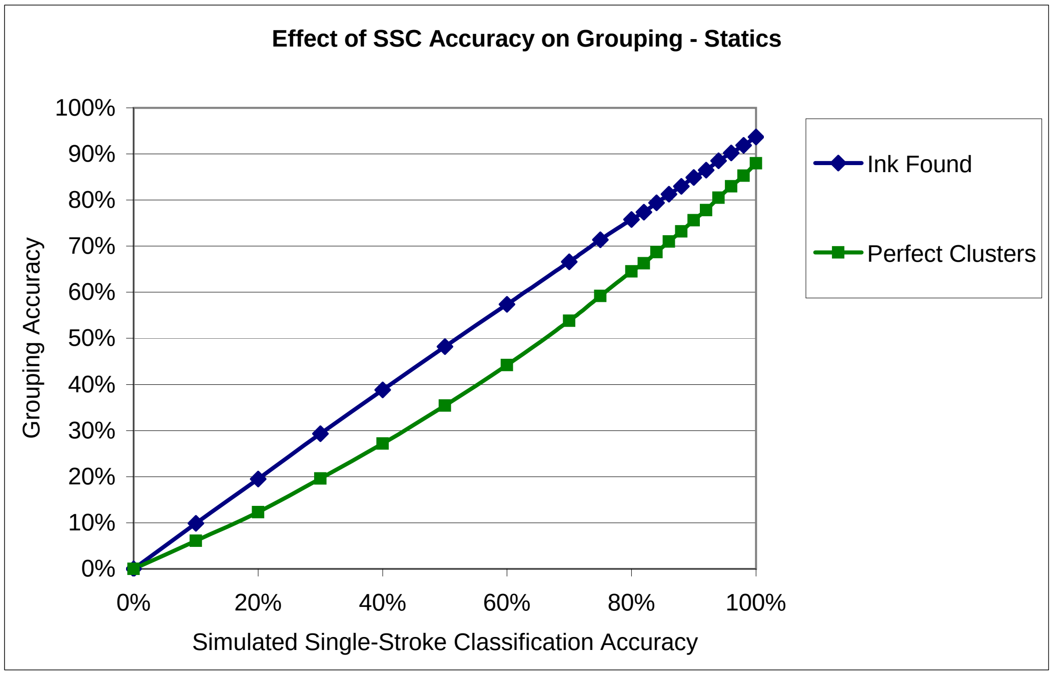


Figure 5.20: The effect of single-stroke classification accuracy (artificially generated) on grouping accuracy for complete statics solutions (using IPC-MD).

grouping without using single-stroke classification. Omitting single-stroke classification improves the accuracy of some classes (arrows), while it makes it worse for other classes (bodies and others). This suggests that while the overall accuracy between these two methods is roughly equivalent, they make different errors. A qualitative evaluation of the differences between these two methods can be found in Section 6.2.2.

Statics Solutions	Ink: Avg / Shape		Shapes: X Errors or Less		
Class	Found	Extra	0	1	2
Body	81.0%	3.1%	57.4%	76.7%	84.4%
Arrow	81.9%	17.9%	63.3%	89.5%	98.4%
Text	93.1%	10.2%	87.0%	97.6%	99.1%
Other	77.2%	210.5%	59.5%	83.2%	90.0%
Overall	91.1%	20.0%	82.9%	95.8%	98.4%

Table 5.37: Grouping accuracy for complete statics solutions using the inductive pairwise classifier and minimum distance labeling method **without** single-stroke classification.

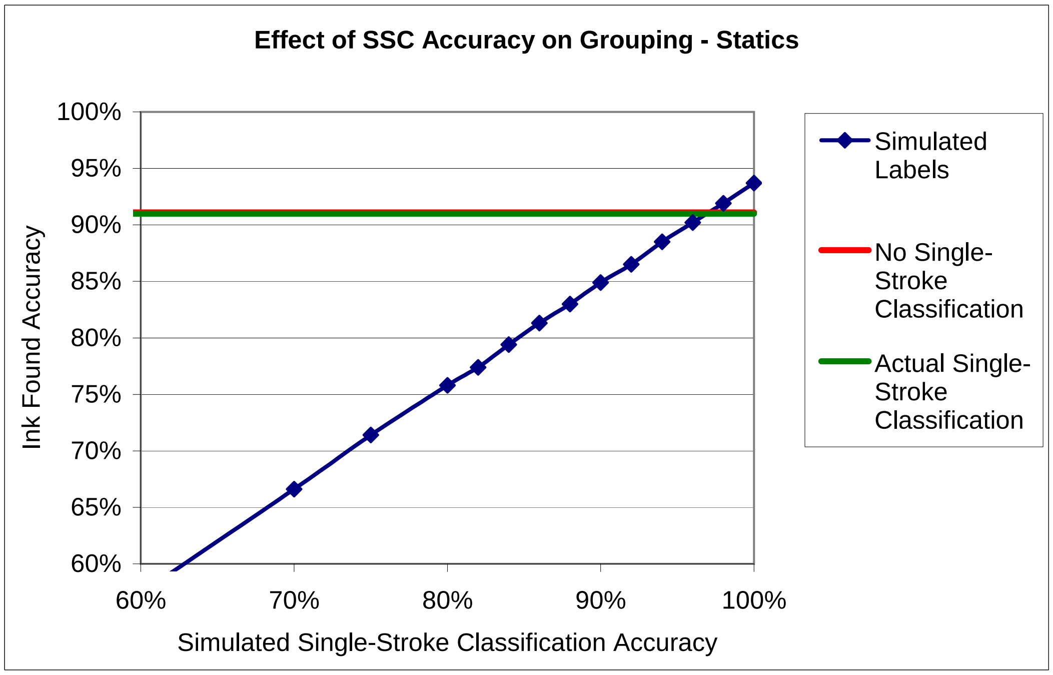


Figure 5.21: The effect of single-stroke classification accuracy (artificially generated) on ink found accuracy for complete statics solutions (using IPC-MD). The red line represents the ink found accuracy if **no** single-stroke classification is performed. The green line represents the percentage of ink found when using multi-way single-stroke classification.

Figure 5.22 shows perfect cluster accuracy for complete statics solutions. While the percentage of perfect clusters increases with our single-stroke classifier, the percentage of clusters with one or fewer errors actually decreases. The perfect cluster accuracy for bodies goes down significantly, while it increases for arrows. Text perfect clusters accuracy is very slightly better, while “other” accuracy goes down without single-stroke classification.

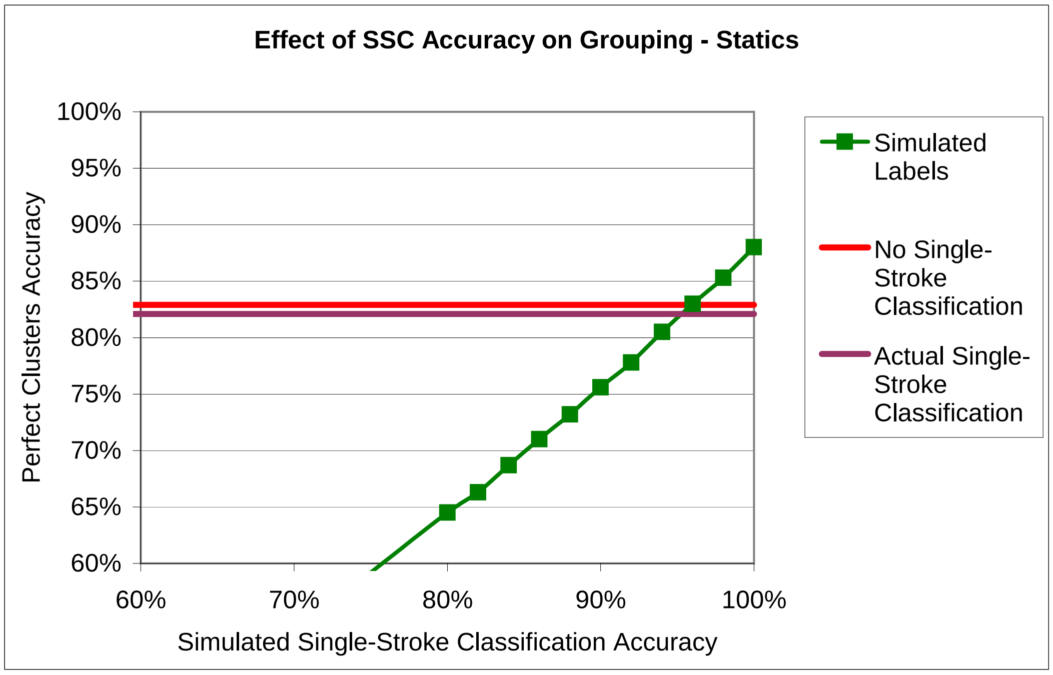


Figure 5.22: The effect of single-stroke classification accuracy (artificially generated) on perfect cluster accuracy for complete statics solution sketches (using IPC-MD). The red line represents the ink found accuracy if **no** single-stroke classification is performed. The purple line represents the percentage of ink found when using multi-way single-stroke classification.

5.2.4 Effect of Perfect Single-Stroke Classification

In the last section we examined the effect of varying single-stroke classification accuracy, and how well the pairwise classifier works without the separation provided by single-stroke classification. We found that in the case of statics sketches, we can achieve better overall grouping accuracy by **not** performing single-stroke classification. The relatively poor accuracy of the single-stroke classifier in the two statics domains introduces many errors that are unrecoverable by the pairwise classifiers. In this section we examine grouping accuracy when the strokes are perfectly labeled. This is the upper-bound on grouping accuracy given our current pairwise classifiers. In this section, all results are found using inductive pairwise classifiers with the minimum distance labeling method (IPC-MD).

In each of the domains, improving single-stroke classification provides large increases in grouping accuracy. The results for each domain are shown Figures 5.23-5.26 and in Tables 5.38-5.41. The results for digital circuit sketches are presented in Table 5.38, and represent an over 60% (5.8 percentage points) reduction in *Ink Found* error for all classes. Table 5.39 presents the results for family tree diagrams. Here, the accuracies are not as high as with digital circuits, however there has been approximately a 50% reduction in error compared to our single-stroke classifier. Table 5.40 shows the results for statics solutions without equations. For this domain, the reduction in error is again over 50%. Finally, for complete statics solutions the improvement is very large for non-text objects, as shown in Table 5.41. However, because there are so many text shapes, and those shapes did not improve very much, the overall accuracy did not improve as significantly as other domains.

Digital Circuits	Ink: Avg / Shape		Shapes: X Errors or Less		
Class	Found	Extra	0	1	2
Gate	96.5%	1.3%	92.0%	99.3%	100.0%
Wire	95.5%	2.6%	91.8%	94.5%	96.3%
Label	99.9%	0.1%	99.6%	100.0%	100.0%
Overall	96.7%	1.7%	93.3%	97.3%	98.3%

Table 5.38: Grouping accuracy for digital circuit diagrams using the **inductive pairwise classifier** and **minimum distance** labeling method, with artificially perfect single-stroke classification.

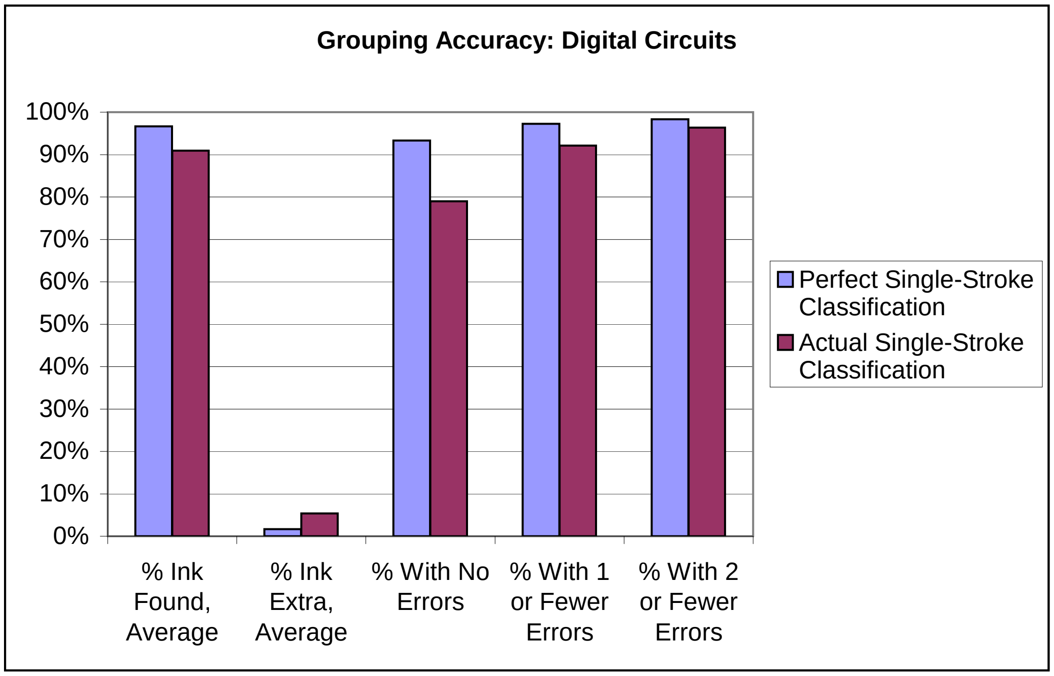


Figure 5.23: A comparison of grouping accuracy when using perfect versus actual single-stroke classification for digital circuits (using IPC-MD).

Family Trees	Ink: Avg / Shape		Shapes: X Errors or Less		
Class	Found	Extra	0	1	2
People	98.7%	1.1%	97.6%	98.3%	99.0%
Text	89.7%	11.6%	79.4%	81.3%	84.1%
Link	91.4%	6.1%	83.6%	95.0%	99.5%
Overall	93.8%	5.0%	88.2%	94.4%	97.2%

Table 5.39: Grouping accuracy for family tree diagrams using the **inductive pairwise classifier** and **minimum distance** labeling method, with artificially perfect single-stroke classification.

5.2.5 Sensitivity to the Number of Stroke Classes

The last two sections have discussed grouping accuracy related to single-stroke classification accuracy, this section seeks to show the best number of single-stroke classes to use. As shown in Figure 5.27, using different classes leads to widely different grouping accuracy. We have used the complete statics solutions domain for this evaluation, be-

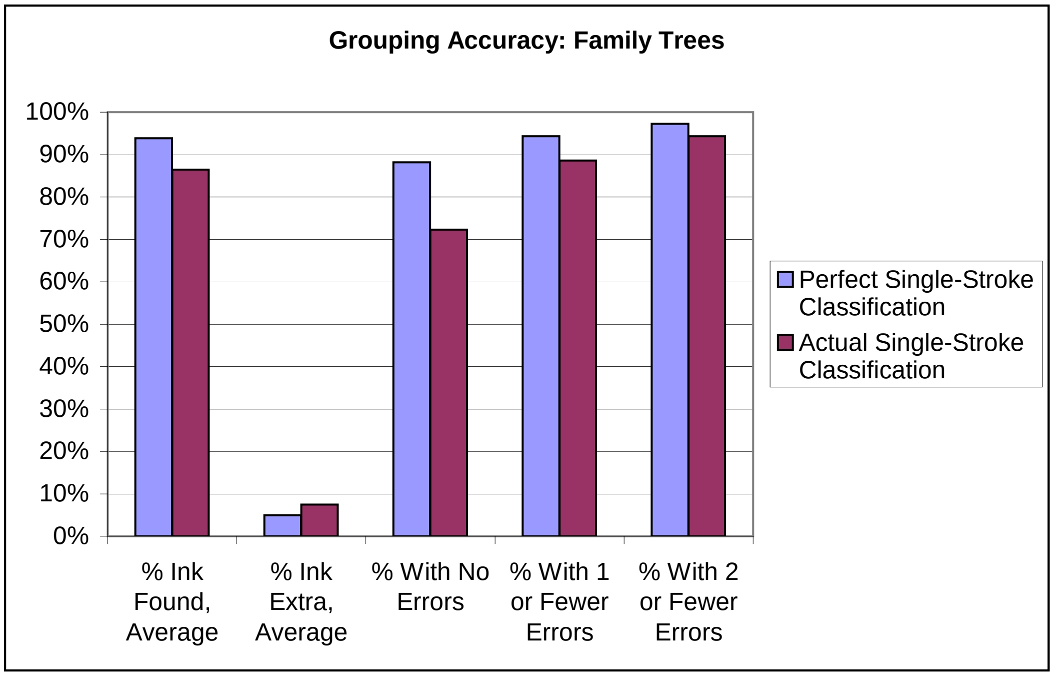


Figure 5.24: A comparison of grouping accuracy when using perfect versus actual single-stroke classification for family trees (using IPC-MD).

Statics Solutions Class	Ink: Avg / Shape		Shapes: X Errors or Less		
	Found	Extra	0	1	2
Body	98.1%	3.3%	95.6%	98.4%	98.7%
Arrow	92.5%	6.5%	83.6%	93.9%	99.6%
Label	95.8%	1.4%	89.9%	92.9%	98.5%
Other	84.6%	43.5%	72.5%	88.1%	93.6%
Overall	93.0%	10.2%	85.5%	93.0%	98.1%

Table 5.40: Grouping accuracy for statics solutions without equations using the **inductive pairwise classifier** and **minimum distance** labeling method, with artificially perfect single-stroke classification.

cause it contains many different shapes, leading to multiple different ways to split them. Here, determining which types of shapes should be part of the same class is nontrivial. There are many different shape types, and there is often little distinction between them. We use the IPC-MD grouping method for the comparisons in this section.

In Figure 5.27, the left-most point (one class) represents grouping without

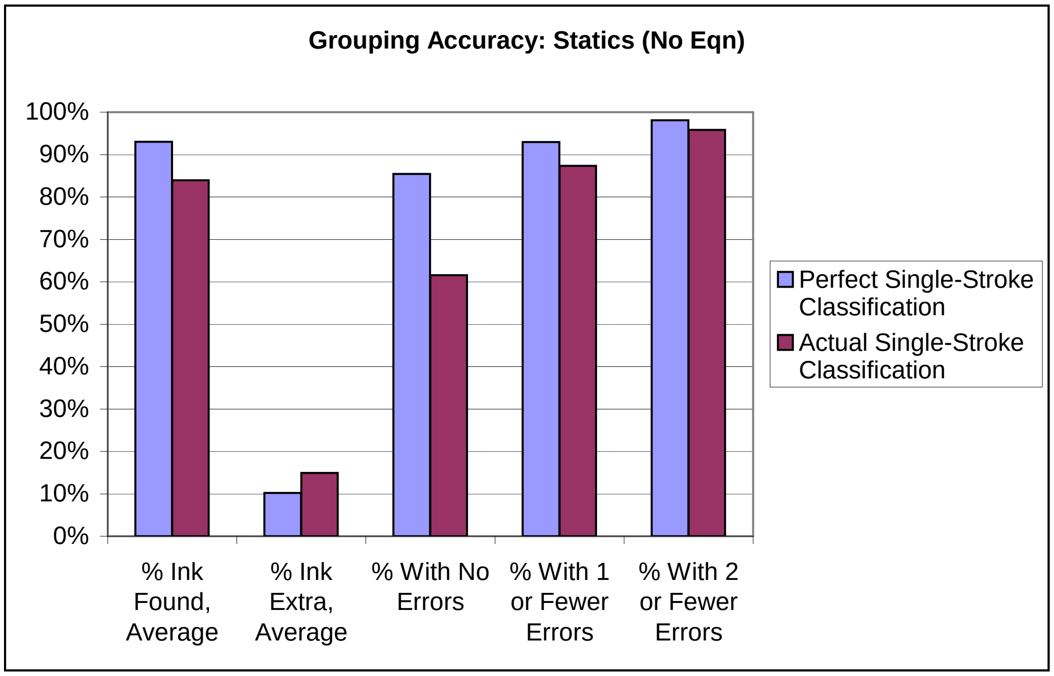


Figure 5.25: A comparison of grouping accuracy when using perfect versus actual single-stroke classification for statics solutions without equations (using IPC-MD).

Statics Solutions	Ink: Avg / Shape		Shapes: X Errors or Less		
Class	Found	Extra	0	1	2
Body	98.4%	3.1%	96.3%	98.2%	98.5%
Arrow	93.7%	5.3%	86.2%	94.1%	99.9%
Text	94.0%	7.7%	88.7%	98.1%	99.2%
Other	86.2%	49.1%	76.3%	88.2%	94.5%
Overall	93.7%	9.3%	88.0%	97.3%	99.1%

Table 5.41: Grouping accuracy for complete statics solutions using the **inductive pairwise classifier** and **minimum distance** labeling method, with artificially perfect single-stroke classification.

single-stroke classification. The next point is for two classes: text and non-text. For the four class point, strokes are split into bodies, arrows, text, and "other." For eight classes, strokes are classified as bodies, arrows, labels, equations, geometry, lines and arcs, points, and "other." The accuracies for the two, four, and eight class single-stroke classifiers can be found in Section 5.1.

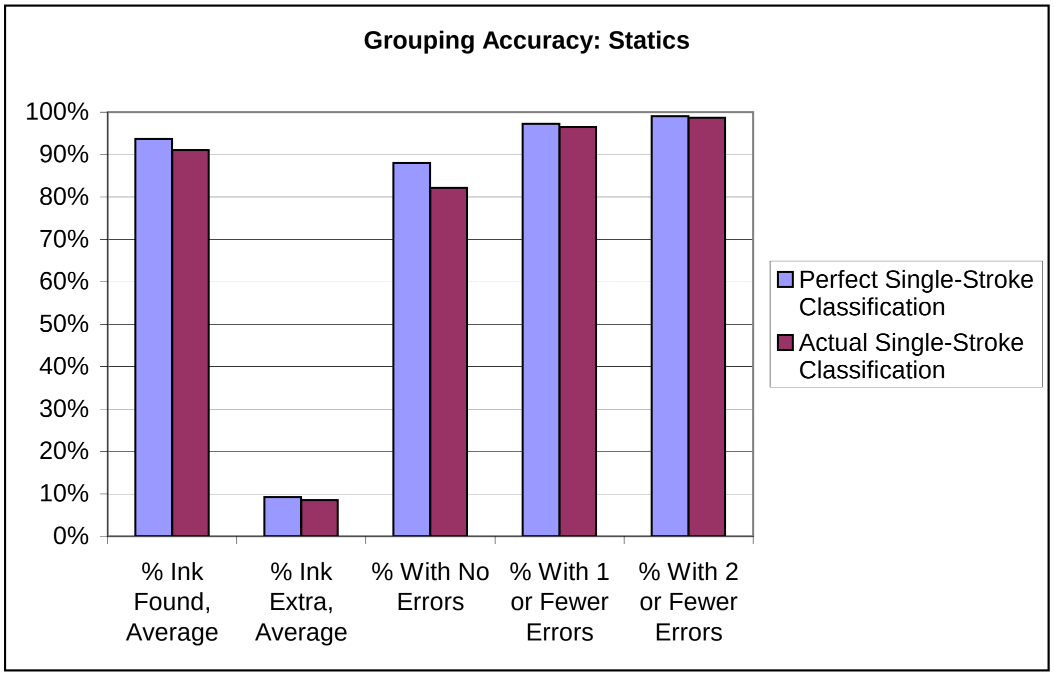


Figure 5.26: A comparison of grouping accuracy when using perfect versus actual single-stroke classification for complete statics solutions (using IPC-MD).

When using perfect single-stroke classifications (dashed line), it is generally beneficial to classify strokes into more classes. However, there is a point above which using more classes does not help grouping, even with a perfect single-stroke classifier. Real single-stroke classifiers are going to have more errors as the number of similar classes increases. These errors will lead to more grouping errors.

Using two classes, text and non-text, the single-stroke classifier introduces labeling errors compared to without single-stroke classification. It does not separate the objects in each class enough to offset them. The net result is a reduction in grouping accuracy. While the single-stroke classification accuracy decreases when using four classes as compared to two, it also separates the shapes in each class enough to making grouping within each class much easier. The net result of this is an increase in overall

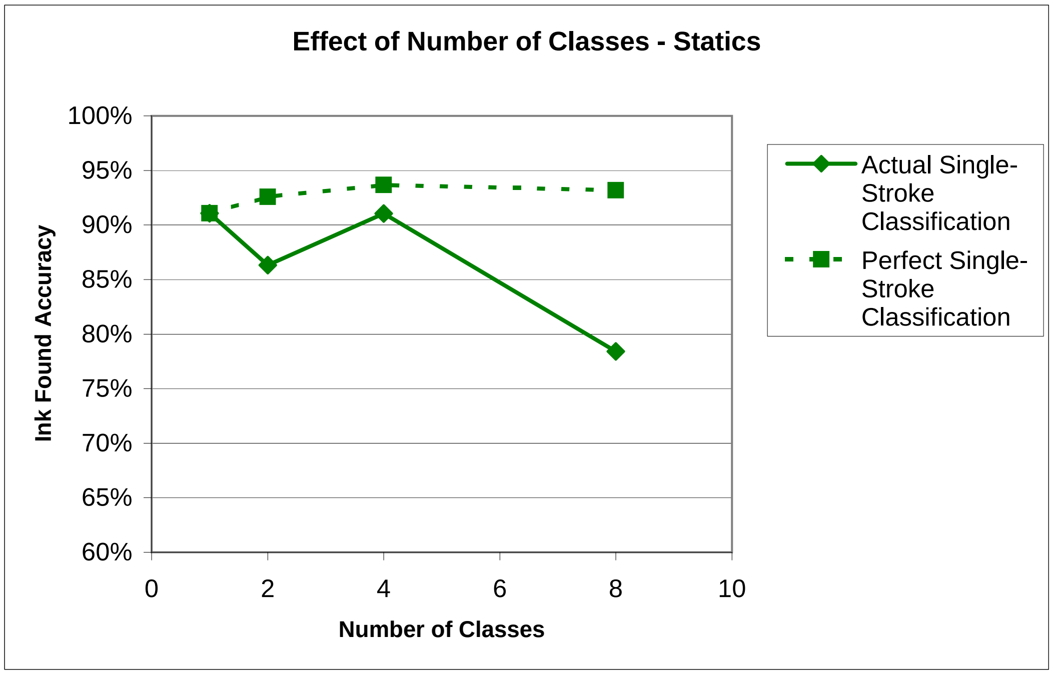


Figure 5.27: Effect of different numbers of classes for single-stroke classification on the percentage of ink found in statics solutions. The dotted line shows the accuracy if we had a perfect single-stroke classifier with a given number of classes. The solid line represents using a single-stroke classifier trained on our data set, which does make errors. We use the IPC-MD grouping method for this comparison.

grouping accuracy. Beyond this point, further separating the sketch does not help the pairwise classifiers, yet it significantly decreases the accuracy of the single-stroke classifier. The single-stroke classifier creates many more unrecoverable errors than before, making overall grouping accuracy much worse.

Determining exactly how many classes to use and what types of shapes to put in each class can be quite difficult. There are a few guidelines that we have found useful; they are discussed in Section 6.1.1.

5.2.6 Iterative Re-Labeling Sensitivity to Seed Value

The *Iterative Re-Labeling* (IPC-IRL) pairwise labeling method described in Section 3.3.2 must be seeded with an initial set of labels for each stroke pair. In this section we examine the effects of using different seed labels. The sets of seed labels are found using the *Minimum Distance* method, with different threshold values (D_0). For the IPC-IRL results reported earlier, we used seed values found using $D_0 = 200$. In this section, we examine the effects of using D_0 values of: 100, 200, 400, 1000, and 5000 himetric units.

Figure 5.28 shows the grouping accuracy at each step of the process. The "Pre" point is the grouping accuracy before any iterating begins. The subsequent steps show the iteration as a number (1-5), while the a and b denote whether the classifier was trained on Set_1 or Set_2, respectively. "Post" is the final accuracy after recombining the two re-labeled data sets and training new pairwise classifiers on the combined set. Each curve in the figure represents a different set of labels used to seed the iterative re-labeling process ("Pre" step). With a threshold, D_0, of 5000, almost no stroke pair is labeled as *FarJoin*; in other words, it closely resembles labeling the pairs as only *Join* or *NoJoin*.

For digital circuits, the classifiers from all five seed labels quickly converge to the same accuracy, with little fluctuation between the accuracy of the two sets. Each curve also ends at approximately the same accuracy. For digital circuits, the seed labels seem to have no effect on the accuracy of the final classifiers.

Figure 5.29 shows the accuracies during the iterative process for family tree

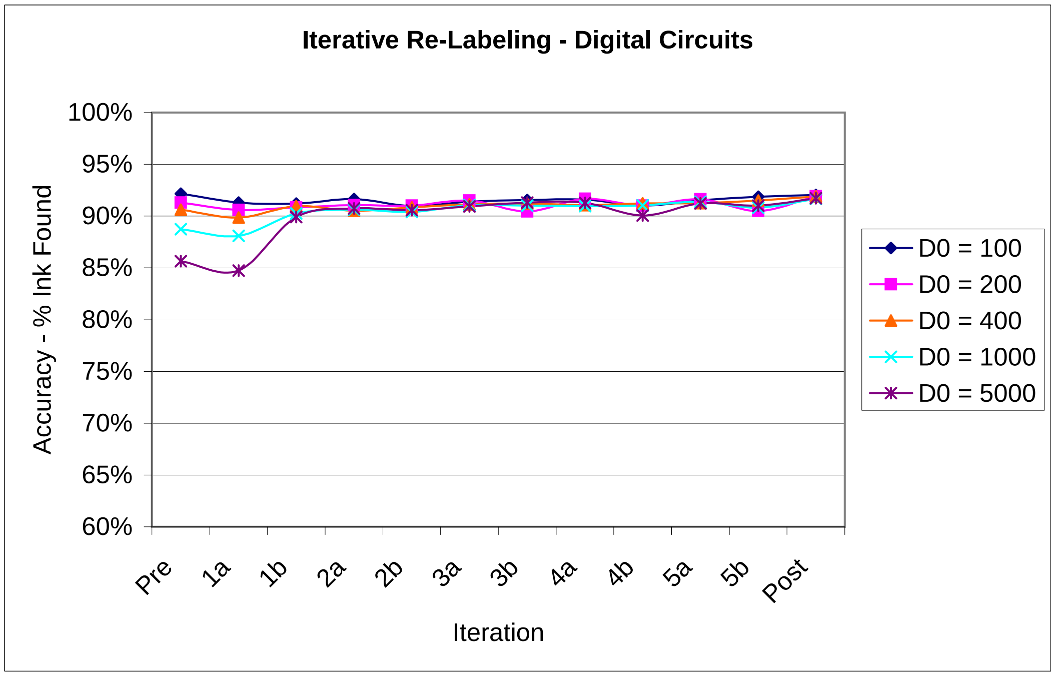


Figure 5.28: The effect of seeding the iterative re-labeling process with different initial labels on digital circuit sketches. The initial seed labels are found with the minimum distance method, where the distance threshold D_0 is the number in the legend.

diagrams. The family tree data set is significantly smaller than the digital circuit set, which makes the accuracy more sensitive to the random split of the data. This is shown as the fluctuations between accuracies for the "a" and "b" iteration steps. Interestingly, the seed that started with the highest accuracy ($D_0 = 400$) ended with the worst accuracy, while the worst pre-iteration labels ended with the best accuracy ($D_0 = 5000$). The re-labeling does not improve the grouping accuracy in this domain; this is reflected in Section 5.2.1, where the IPC-MD method outperforms the IPC-IRL method.

Results for statics solutions without equations are shown in Figure 5.30. Similar to the family tree data, there are no gains in accuracy. The accuracy occasionally fluctuates between the "a" and "b" steps, however the final accuracy is fairly similar for

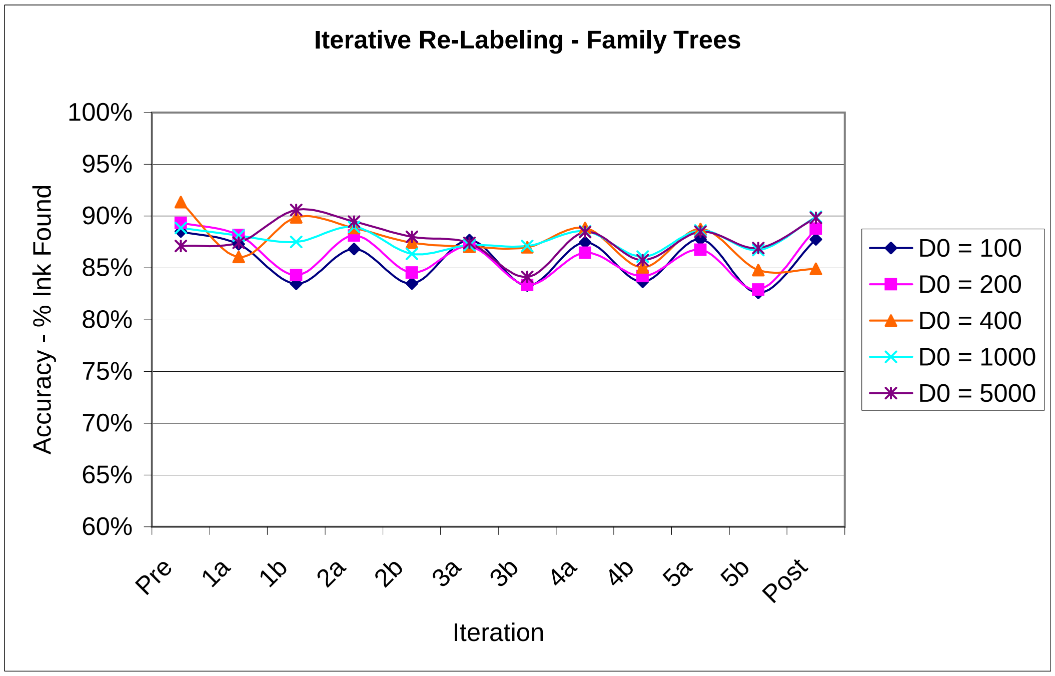


Figure 5.29: Shows the effect of seeding the iterative re-labeling process with different initial labels on family tree sketches. The initial seed labels are found with the minimum distance method, where the distance threshold D_0 is the number in the legend. The minimum distance method uses a value of 200 himetric units.

each set of seed labels.

5.2.7 Shape to Cluster Matching

We have experimented with two different methods for matching the expected shapes to the machine-generated clusters. The first method was described in Section 5.2, where machine-generated clusters are matched to the expected shapes based on the number of strokes in common, and a cluster can be matched to any shape, regardless of the class of the strokes in the cluster. For instance, if two strokes of an *and* gate were classified as wires and then clustered together, this cluster would be matched to the expected *and* gate (which is labeled as a gate rather than a wire). In this section we

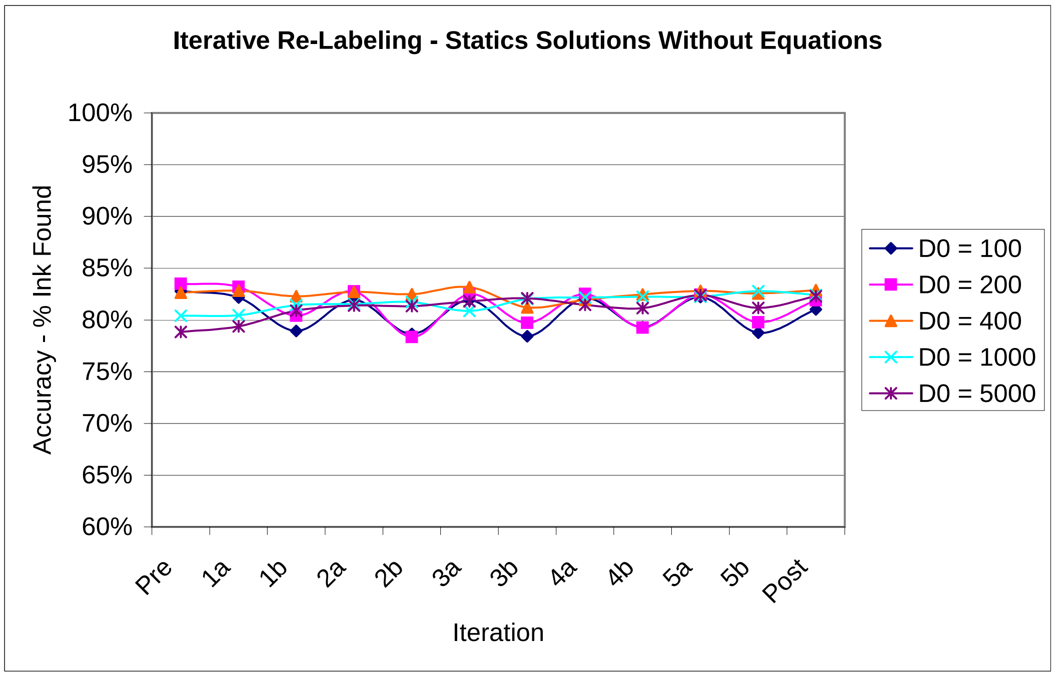


Figure 5.30: Shows the effect of seeding the iterative re-labeling process with different initial labels on sketches of statics solutions without equations. The initial seed labels are found with the minimum distance method, where the distance threshold D_0 is the number in the legend. The minimum distance method uses a value of 200 himetric units.

examine a second method, which matches expected shapes only to clusters containing strokes of the correct class. "Class-dependent" matching allows shape recognizers to be customized for a given class (shape recognizers are used *after* clusters are formed, as shown in Figure 1.3).

The results for each domain are presented in Figures 5.31-5.34. For digital circuit sketches (Figure 5.31), the percentage of ink found actually increases when using class-dependent matching, while perfect cluster accuracy decreases, both by very small amounts. The accuracy for family tree diagrams is very slightly worse when using class-dependent matching (Figure 5.32). The effect is much more noticeable for statics solutions, both with and without equations, as shown in Figures 5.34 and 5.33, respectively.

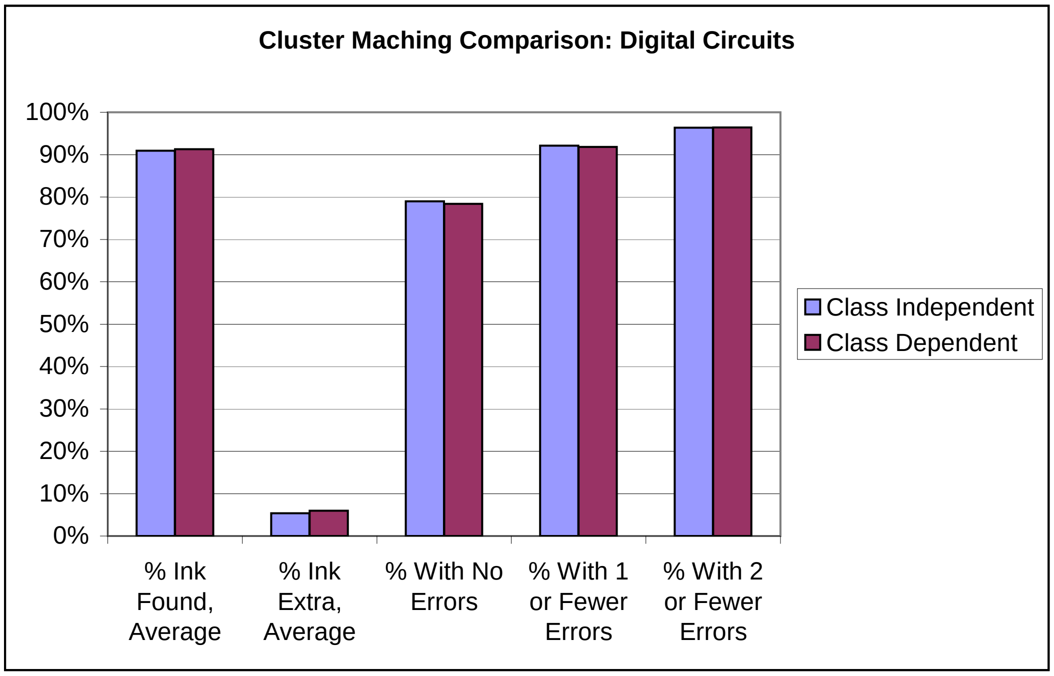


Figure 5.31: The effect of using a class-dependent versus class-independent cluster matching technique for digital circuit sketches. With a class-dependent matcher, clusters of a given class can only be matched to expected shapes of the same class. Class-independent matching allows any cluster to be matched to a given shape.

For digital circuits and family trees there is little difference in accuracy depending on which cluster matching method is used. For shape recognition in these two domains, it would be advisable to use class-dependent recognizers or templates. With both statics domains however, the drop in accuracy between the two methods indicates that there are quite a few clusters with the wrong class label. Therefore, it would most likely be better to use a single shape recognizer that is not dependent on class.

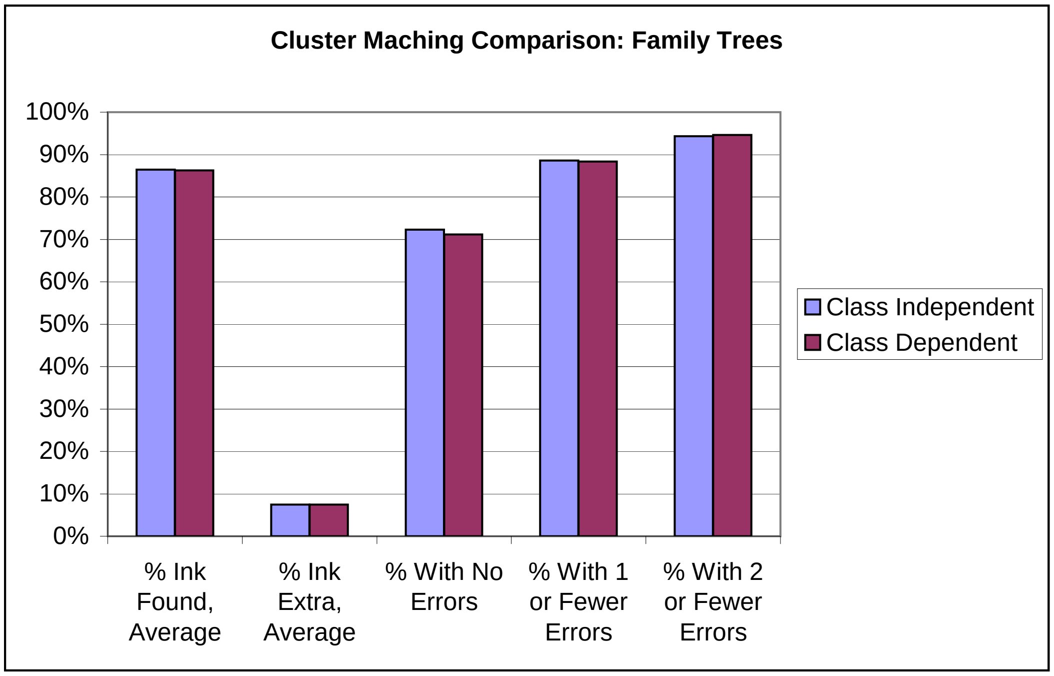


Figure 5.32: The effect of using a class-dependent versus class-independent cluster matching technique for family tree sketches. With a class-dependent matcher, clusters of a given class can only be matched to expected shapes of the same class. Class-independent matching allows any cluster to be matched to a given shape.

5.2.8 Pairwise Feature Importance

Similar to the single-stroke feature importance in Section 5.1.4, we have analyzed the importance of the pairwise features for each class in each domain. Again, we did this with WEKA's attribute selection methods, using information gain ratio and the *Ranker* search method with 10-fold cross-validation. Figure 5.42 shows the ranked list of the feature importance for the inductive pairwise classifier, where the merit (WEKA's internal measure of importance, in this case determined by the information gain ratio) is averaged from each of the 14 pairwise classifiers, one for each class in each domain (three for digital circuits, three for family trees, four for complete statics solutions, and four for statics solutions without equations). Tables showing the feature importance for

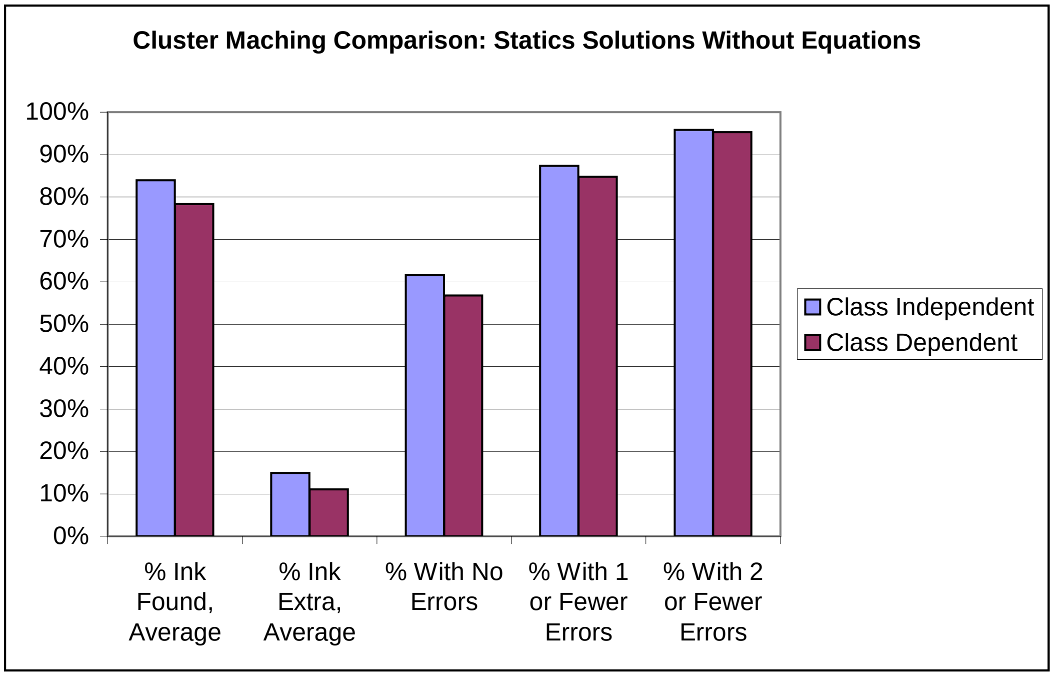


Figure 5.33: The effect of using a class-dependent versus class-independent cluster matching technique for statics solution sketches without equations. With a class-dependent matcher, clusters of a given class can only be matched to expected shapes of the same class. Class-independent matching allows any cluster to be matched to a given shape.

each of the 14 pairwise classifiers, individually, are presented in Tables D.5-D.18.

The two most important features for pairwise classification are: whether the pair of strokes is part of the same closed path, and the minimum distance between the strokes. These two features are not unexpected. Often times strokes belong together if the are close to each other. Also, if two strokes are part of the same closed path, they most likely belong together.

The two closeness features are also found to be important, however the smaller of these two ($Closeness_{Small}$) has a higher average merit. The other two features in the top half of the ranking are the minimum endpoint-to-endpoint distance and minimum

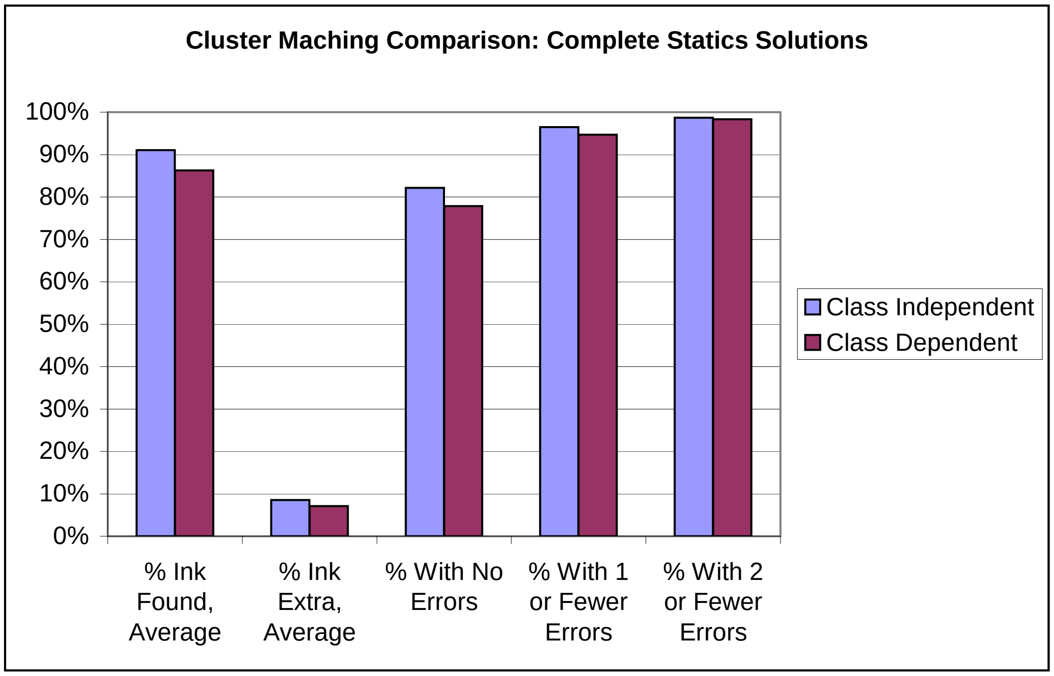


Figure 5.34: The effect of using a class-dependent versus class-independent cluster matching technique for complete statics solution sketches. With a class-dependent matcher, clusters of a given class can only be matched to expected shapes of the same class. Class-independent matching allows any cluster to be matched to a given shape.

Feature Importance, Pairwise - Average of All Domains	
Average Merit	Attribute Name
0.392	Part of Same Closed Path
0.306	Minimum Distance
0.277	$Closeness_{Small}$
0.255	Minimum Endpoint-to-Endpoint Distance
0.246	Minimum Endpoint-to-Anypoint Distance
0.234	$Closeness_{Large}$
0.207	Centroid Distance
0.202	Time Gap
0.156	Maximum Distance
0.131	Y-Overlap
0.116	X-Overlap
0.072	Ratio XL
0.048	Ratio LL

Table 5.42: Ranked list of features for pairwise classification in all domains (averaged merit).

endpoint-to-anypoint distance.

The importance of the time gap between the strokes varies quite widely (see Tables D.5-D.18), and is most important for all text and label shapes, as well as “other” in the statics domains. The time gap is not particularly useful for wires, because they are often not drawn sequentially.

The two overlaps and ratios were fairly consistently at the bottom of the rankings. This is surprising at first glance, particularly the *X-Overlap* for text in statics sketches (mostly equations). However, since one equation was often drawn directly above another one, there are many cases where the strokes overlap in this direction without belonging to the same character.

5.3 Running Time

One of the most important aspects of our stroke grouping method is the speed with which it can group a sketch. Our method uses classifiers to determine whether a *pair* of strokes is part of the same object, rather than relying on a search process with recognition. For example, search methods can be exponentially expensive. Since our algorithm compares pointwise distances between strokes, and this computation dominates the cost of the feature calculation step, the computation time is expected to be of order $O(n^2)$ where n is the number of points in the sketch. Figure 5.35 shows actual computational cost versus the number of points in the sketch, n, confirming that the cost is $O(n^2)$.

Fortunately, it is possible to reduce the computation cost using a simple filter.

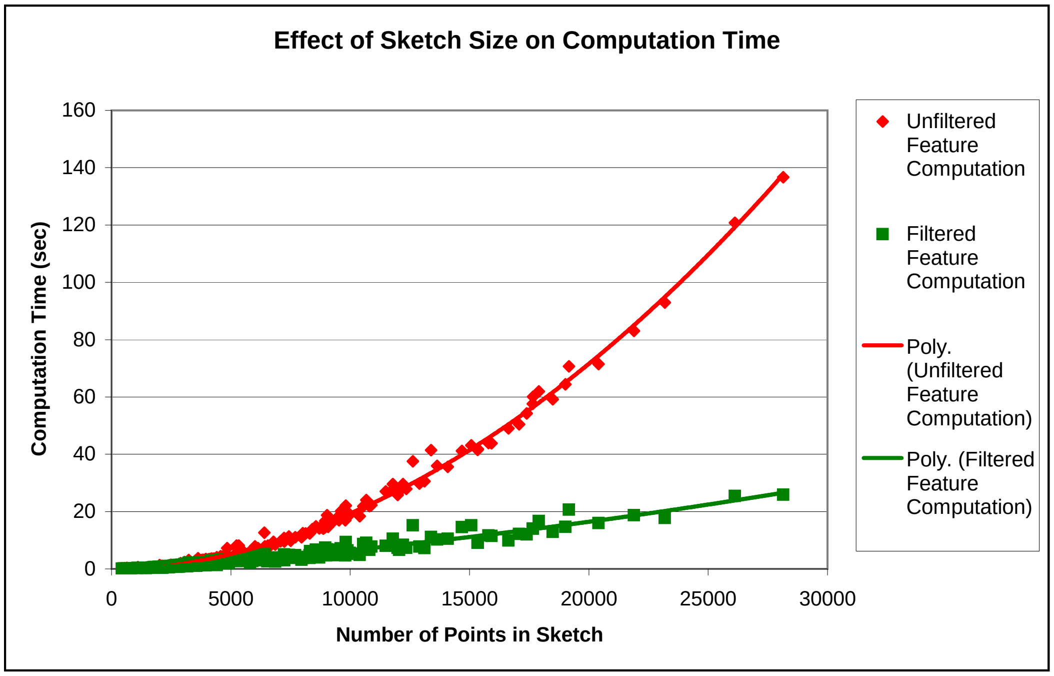


Figure 5.35: Computation time for all features in a sketch as a function of the number of points in the sketch. Using a conservative filter of 50mm can significantly reduce computation times. Both trend lines are polynomial (n^2) fits. Each data point represents the computation time for a sketch.

If two strokes are far enough apart, there is no need to compute the pairwise features, because the strokes are very unlikely to be from the same cluster. Thus, we do not compute the features for stroke pairs whose bounding boxes are greater than 5000 himetric units (50 mm) apart at their closest points. We chose 50 mm as a conservative estimate of the maximum d_{min} between a pair of strokes that are part of the same object. We have found that filtering does not decrease the accuracy of the grouper; the results presented in the previous sections used classifiers trained using features from filtered pairs of strokes. As Figure 5.35 shows, the process is still $O(n^2)$, but the constant coefficient is lower than without filtering.

Once the features are computed, the actual classification of an instance (single-stroke or pairwise) is very fast, and is negligible in relation to the feature computation time. For the largest sketch in the corpus (712 strokes), the user spent 20 minutes drawing the solution to the statics problem, yet the feature computation time was only about 140 seconds (11.1% of the drawing time)[2]. Without filtering, feature computation time was always less than drawing time. Computation time ranged from 0.4%-12.0% of the time taken to draw the entire sketch, with a median time of 2.0%. When filtering is used, the times drop to a range of 0.3%-3.6%, with a median of 1.1% of the time to draw the entire sketch. The features are computed incrementally as the user draws. Therefore, only the features for the last drawn stroke need to be computed before the program can group the final sketch. While this method is fast enough for an online system (grouping as the user draws), we have spent little time optimizing the computation, and believe that significant speed gains are possible. These gains likely would not be noticeable to the user, but would consume fewer computation cycles and power. One of the easiest ways to do this would be to down-sample the points in each stroke, which could reduce n significantly.

[2]All speed tests run on an HP TC4400 Tablet PC - Intel Core 2 Duo T7200 @ 2.00GHz with 2GB RAM, running Windows XP SP3 (Tablet PC Edition). This is the same type of computer that was used for collecting the ink for statics and digital circuit sketches.

Chapter 6

Discussion

We have developed a fast and accurate method for grouping strokes into objects. By classifying individual strokes into classes, we create separation between objects of the same class, thereby making grouping easier. To group the strokes within a given class, we have created two different types of pairwise classifiers (TPC and IPC). The pairwise classifiers are used to determine pairs of strokes that should be joined. A chainer then combines joined stroke pairs to form clusters.

Once the clusters have been created, our algorithm is finished. The clusters can then be recognized and used in a higher-level program. Accurate and efficient clustering is a very difficult part of sketch understanding, one that has kept sketch-based approaches from being used in many applications. Our work helps to overcome this roadblock. As discussed in Chapter 7, there have been a number of sketch recognition systems created, however almost all of them use recognition to obtain accurate groupings. These systems often search through many erroneous clusterings, using recognition to find a valid set of clusters. Our approach can complement these systems by providing an accurate set of initial clusters, thereby improving the system as a whole.

Our system quickly classifies and groups, once the features have been computed. Feature computation is done as the user draws. This saves time when it matters: at the time that the user has finished sketching and is expecting the system to recognize the sketch and provide feedback or interaction.

6.1 Single-Stroke Classification

Single-stroke classification is an important part of our method. It simplifies pairwise comparisons, leading to more efficient and accurate grouping. We have developed a classifier which uses adaptive boosting (AdaBoost) with decision trees and a set of features that extends previous efforts at single-stroke classification. Our classifier accurately classifies text versus non-text strokes (97.2% on digital circuits). In direct comparison to the entropy method described in [3] and Microsoft's© commercial InkAnalyzer algorithm, our classifier performed significantly better in our digital-circuits and both of our statics domains, while the entropy method provided slightly better results in the family-tree domain. On their own data set, Bhat and Hammond report 92% accuracy for their entropy method. Other work has reported single-stroke classification accuracies on their own data sets: Patel *et al.* ([30]) reports approximately 70% accuracy, while Bishop *et al.* ([5]) report approximately 95% accuracy, and Qi *et al.*'s ([33]) report approximately 96% accuracy.

While previous stroke classifiers have performed two-way classification, our classifier is shown to be accurate for three or more classes. When classifying strokes into multiple categories, it is important to select good class distinctions. A good set of classes is one in which each class has characteristics that differentiate it from the others, allowing the classifier to accurately learn the difference between them.

6.1.1 Picking Classes

In the case of complete statics solution sketches, we have examined the effect of different sets of classes. Using only two classes, text and non-text, our classifier is 92.1% accurate. However classifying into these two classes does not help grouping. In fact, performing text versus non-text classification before grouping results in *worse* accuracy than not classifying strokes at all. In this situation, the single-stroke classifier introduces errors without significantly simplifying grouping. More classes are needed to improve grouping. Using four classes: *Body*, *Arrow*, *Text*, and *Other*, the single-stroke classifier is slightly less accurate (88.8%) than with two classes, yet the grouping accuracy increases dramatically. Adding still more classes was detrimental to both single-stroke and grouping accuracy.

In general, it is important to select a set of classes that is differentiable, and that separates objects within a class. Having too many classes can make the classes indistinguishable. For instance, there was a large amount of confusion between *Label* and *Equation* strokes when using eight classes. Conversely, having too few classes results in objects that are not adequately separated from each other after classification. An example of this is when *Arrows* and *Bodies* are part of the same class, as they are in the non-text class for two-way classification. In general, it is best to classify strokes into as many classes as can be accurately distinguished. If accurate classification is not possible, it is sometimes better to perform grouping without stroke classification.

Additionally, for optimal accuracy, the class distribution should be roughly equal. Having an unbalanced data set leads to a classifier that is good at classifying one class at the expense of the others. For example, with statics sketches, there were many more text strokes than non-text strokes. The classifier became very good at identifying *Text*, while it performed poorly on the other classes. The class distribution can not always be controlled, however the classes should be chosen such that they are as balanced as possible.

The goal of single-stroke classification is to maximally separate the shapes, unfortunately, there is no *a priori* way to pick the best set of classes. We have found that using a general shape type usually gives a good starting point. For example in digital circuits, using the class *Gates* worked better than having one class for *and* gates, one for *or* gates, etc. If the shapes in a class are too close together, causing them to incorrectly be clustered, more classes are needed to separate them. This is not always possible, as is evidenced by the *Links* in family trees (Figure 2.1(c)), as well as *Wires* in digital circuits. Finding a good set of classes is the most difficult and time-consuming part of designing the system for a new domain.

6.1.2 Extension to New Domains

Once a good set of classes has been identified, and sketches have been labeled, our method can be easily applied to new domains. There is no additional coding necessary. To extend our method, the features must be computed from the sketches, and a new set of classifiers can then be trained.

We have demonstrated our method in four domains: digital circuits, family trees, complete statics solutions, and statics solutions without equations. Our single-stroke classifier performs best on digital circuits. For family tree diagrams, there are only 27 sketches, compared to 192 digital circuit diagram sketches, which significantly reduces the amount of training data. Some of the family-tree sketches and almost all of the statics sketches had significantly more strokes and many looked "messier" than the digital circuits; see Appendices A, B, and C for examples. The lack of training data and the complexity of sketches likely contributed to poorer classification accuracy for family trees and statics compared to digital circuits.

The domains we have considered have many different characteristics, and represent a fairly wide range of diagrammatic sketches. Digital circuit sketches have very little text, and have sets of objects (gates) connected by wires that can overlap. Family tree diagrams usually have more text, and the text objects contain multiple characters. Family trees are similar to organizational charts in their construction, having enclosed objects linked by various connectors – arrows, straight lines, and jagged lines. Statics solutions do not have the same types of connected objects; instead there are objects (bodies) with many other types of objects interacting with them. Arrows can overlap with bodies, while text labels are often found around or inside of bodies. These sketches also have large amounts of text in the form of equations.

Given the high performance on this wide range of sketches, we believe our methods will work well for many other types of diagrammatic sketches, such as flow charts, UML diagrams, and organizational charts. For our method, we have assumed that a single stroke is not part of more than one object; this is not the case in some domains, such as analog circuit diagrams. Our method could potentially work for analog circuits if the strokes were first segmented, and the single-stroke classification was changed to single-segment classification.

6.1.3 Features

Some domains may benefit from the development of additional features. While our approach strives to be domain independent, some domains have unique characteristics that are not captured by our current set of features. For our four domains, we have shown that perceptual and size-based features are often very important. Even still, the feature rankings varied quite widely between the domains.

To examine the benefit of additional features in complete statics solution sketches, we added six extra features. Doing so increased the single-stroke classification accuracy from 88.8% to 89.5% overall. Similarly, for statics without equations, accuracy increased from 80.3% to 80.7% overall. Despite these small increases, the overall grouping accuracies did not measurably improve. It seems that much of the information that was gained by adding these features was already accounted for in others. In other words, they did not contain much *new* information. While the addition of these six features gave little increase in accuracy, it does not mean that other features can not provide a large benefit. Additional features likely need to address different aspects of a stroke in order to be effective.

6.1.4 Single-Stroke Classification Importance

One of the major insights in this work is that if strokes can be accurately classified, the sketch can be accurately grouped. For instance, when the strokes in statics sketches without equations are perfectly classified, grouping accuracy using the IPC-MD method is 93.0% ink found and 85.5% perfect clusters, compared to 83.9% ink found and 61.6% perfect clusters when using our single-stroke classifier, which is 88.8% accurate.

In general, increasing single-stroke classification accuracy directly leads to better grouping. This is probably the most effective way to improve grouping, and is the clearest path forward for improving our algorithm.

6.2 Grouping

We use pairwise classifiers to join strokes in an accurate and efficient manner. By using pairwise classifiers we do not need to consider a large search space, thus keeping computation tractable.

6.2.1 Classifiers

In most domains, the single-stroke classification is good enough at separating the objects that a simple pairwise classifier works well. The *Thresholded Pairwise Classifier* (TPC) accurately groups some strokes, however it is not as flexible as our other methods. The thresholds do not have enough information to accurately group strokes such as *Wires*, which often overlap each other, and *bodies*, which often have long pauses between strokes.

There are many subtleties to grouping. For instance, it can be easy to overlook the fact that the classifiers act on *pairs* of strokes. However, it is not the pairwise classification accuracy that is important, but rather the accuracy of the final clusters of strokes. Chaining stroke pairs which have a stroke in common makes it unnecessary to actually classify every pair from the same object as being joined. This means that the program can recover from false-negative joins. The opposite error, a false-positive join, can be much more damaging because it is unrecoverable. This cost difference seems to make this situation amenable to a cost-sensitive classifier, however our preliminary studies of this indicate that there is little or no increase in performance, even with the false-positive cost being two orders of magnitude greater than the false-negative cost. This insensitivity to cost suggests that there are pairs of strokes that should not be joined which have feature values that are virtually identical to many positive join examples.

One example situation where *NearJoin* and *NoJoin* instances have very similar characteristics is in digital circuits when a *notbubble* is added to the input of a logic gate, such as in Figure 6.1. These should be two separate objects: an inverter (*not*) and a *nor* logic gate. However the pairwise features for the circle forming the inverter and the back stroke of the *nor* gate (which is intersecting it), are very similar to those of the circle on the right side of the *nor* gate and the stroke intersecting it. There may be more cases similar to this which will be very difficult to fix with domain independent methods.

Besides the TPC method, we also use an *Inductive Pairwise Classifier* (IPC)

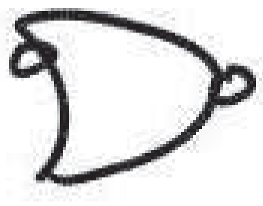


Figure 6.1: This example shows two different gates, the *notbubble* on the input side of a *nor* gate. The grouping classifier has difficulty learning the difference between the *notbubble* that should be joined on the output side of the gate, and the *notbubble* on the input side that should not be joined to the gate.

to group strokes together. This method is more flexible, however there is more work required to properly train the classifier. Using only *Join* and *NoJoin* labels, the IPC is actually less accurate than the TPC. This is likely because the IPC is trained with information that indicates that every stroke pair from the same object needs to be joined. This is not the case, because chaining can join distant strokes via intermediate ones. To improve accuracy, the *Joins* need to be split up into *NearJoins* and *FarJoins*; this allows the classifier to reliably learn the difference between *NearJoins* (the most important cases) and *NoJoins*. Combining the *FarJoins* with the *NoJoins* makes grouping slightly less accurate for most domains. All three join classes should be kept for optimal classifier learning.

We experimented with two different techniques for labeling stroke pairs from the same object as either *NearJoin* or *FarJoin* (stroke pairs from different objects are always labeled as *NoJoin*). The *Minimum Distance* (MD) method uses a set of conditions to determine if the pair is a *NearJoin* or *FarJoin*; if any of the three conditions is met, it is labeled as a *NearJoin*. The second labeling method is called *Iterative Re-Labeling* (IRL). It uses rules to update the labels for the pairwise training examples based on errors in the post-chaining clusters.

The MD and IRL methods have comparable accuracy. The advantage of the MD method is that it is easier to implement, and is quicker to train. The MD method uses a hard-coded threshold of 200 himetric units; this value is likely *not* optimal for all situations. The IRL method does not require a threshold. It can be seeded with most any set of labels, and within a few iterations will arrive at a near optimal solution. In fact, we found that seeding the IRL with very little information regarding *FarJoins* consistently produced high accuracy. By labeling very few instances as *FarJoin* in the seed, the IRL is able to determine which labels are important without bias.

6.2.2 Single-Stroke Classification Errors

Grouping accuracy is greatly affected by the output of the single-stroke classifier. The percentage of ink found is almost directly proportional to single-stroke accuracy, while the perfect cluster accuracy is even more dependent on accurate single-stroke classification. When single-stroke accuracy is too low, it is better to simply skip it. In these instances, it is better to perform pairwise classification on all nearby stroke pairs.

Here, we examine the differences in errors made for complete statics sketches with and without single-stroke classification. The grouping accuracies are approximately equal, however the individual errors are different. When using the single-stroke classifier, there are more clusters with a missing stroke. This is mostly due to the stroke being misclassified, and therefore unable to be appropriately grouped. Without the single-stroke classifier, there are many more clusters that are merges of two shapes. This is especially true with characters.

About two-thirds of the clusters that are incorrect using one approach are correct using the other. By combining the output from grouping with and without single-stroke classification, we may be able to improve clustering compared to a single approach on its own. However, there does not seem to be a distinct pattern to the errors, making it difficult to decide which clustering interpretation to use if they conflict.

6.2.3 Improving Accuracy

We can improve the performance of single-stroke and pairwise classifiers by incorporating user-specific training examples. We examined this for digital circuits and complete statics solutions by performing a sketch-holdout validation. In doing so, we achieved about a 10% reduction in error. For digital circuits, the single-stroke classification accuracy rose from 93.6% to 94.7%. The grouping accuracy improved as well, from 90.9% ink found and 79.0% perfect clusters, to 92.1% ink found and 80.6% perfect clusters. The results are similar for statics solution sketches.

We demonstrated that, for statics solutions, the single-stroke accuracy can be improved with the development of additional features. It is likely that additional features would improve pairwise classifier accuracy as well.

6.3 Future Work

While we are able to achieve high accuracy in most domains, there is still much room for improvement. The most promising path for future work is to improve single-stroke classification accuracy. Here, we present four potential methods for doing this.

The first method improves features that are currently used by the single-stroke classifier. For example, the *Inside a Closed Path* feature is currently computed by checking whether the stroke is contained within the bounding box of the closed path, which can be much larger than the closed path itself. By using the convex hull of the closed path, we could more accurately determine whether a stroke is actually *inside* the closed path. The convex hull area could also be used to more accurately compute ink density. It currently uses the area of its bounding box, which can change based on the stroke's orientation. The intersection features could also be improved so that they provide more information than just the number of intersections the stroke has. By changing the intersection features to include the angle at which the stroke is intersected, we may be able to better represent a stroke's context.

The second method develops new features for the single-stroke classifier. We demonstrated a small improvement for statics solutions by adding six features, and the accuracy could be further improved with yet more features. Future work should study how *people* differentiate objects, which may give insights into new ways to characterize the stroke.

The third method improves the inductive classifier. We found that AdaBoosted decision trees performed well, however, there are many classification techniques that we did not explore. Alternatively, we could try to improve the AdaBoosted classifier. While it would make the classifier more complex, adding more iterations may improve accuracy. Different base classifiers may also be more accurate than decision trees, and should be explored.

The forth method corrects single-stroke classification errors before grouping.

By adding a post-single-stroke classification step, we may be able to fix some of the incorrectly classified strokes. For example, we could add another classifier (after single-stroke classification) that uses features based on a stroke's neighbors and *their* classifications.

Besides improving single-stroke classification accuracy, further research should improve pairwise classification. It could be improved by adding new features, or improving the inductive classifier, similar to single-stroke classification. Also, using the output from the *Thresholded Pairwise Classifier* in conjunction with that of the *Inductive Pairwise Classifier* may be helpful in determining the pair's true classification. While we explored a number of pair labeling techniques, there are likely some optimizations left to be done.

We developed two different pairwise classifiers; future work should develop additional techniques. For example, agglomerative clustering may work well for grouping strokes after they have been separated. Since single-stroke classification creates separation between objects, some previously developed grouping techniques, which were too computationally expensive before, may become tractable.

Finally, we have evaluated our technique using four domains. Future work should extend this method to new domains, showing that our method is indeed extendable to a wide variety of situations.

Chapter 7

Literature Review

As the field of sketch understanding grows, there are an increasing number of systems that attempt sketch recognition. However, to our knowledge no one has previously created a general purpose method for grouping strokes, without using recognition, in freely-drawn diagrammatic sketches. In this chapter we discuss previous full sketch recognition systems [1, 5, 8, 10, 12, 16, 21, 19, 37, 40], symbol recognizers [6, 13, 15, 22, 31, 32, 34, 35, 45, 48], and single-stroke classifiers [3, 4, 5, 30, 33, 43, 44] as they relate to our work.

7.1 Full Sketch Recognition

Researchers have used a number of techniques for full sketch recognition systems, which group and recognize objects to interpret the sketch. Some systems avoid automatic grouping by requiring the user to manually select strokes that belong together. Other systems group strokes and recognize them using special heuristics that are unique to a domain. Many systems rely on search and recognition to interpret the sketch by enumerating and evaluating many hypotheses. In this section we discuss examples of each of these techniques.

One approach to sketch recognition is to simplify the problem by removing automated grouping. LaViola and Zeleznik do this in *MathPad²* by requiring the user to manually select the strokes in a group [19]. The system then attempts to recognize the selection as a string, which is to be evaluated by Matlab. While this is an effective means of obtaining the groups in a sketch, it places a burden on the user to either interrupt their sketching process or go back after completing the sketch and select all of the groups. Doing this can be quite tedious, especially if there is a large number of groups in the sketch.

Another (similar) approach is presented in [16]. Here, Hse and Newton "beautify" strokes for use in PowerPoint slides using the recognition technique described in [15]. However, like MathPad^2, this system requires the user to provide an explicit cue to group strokes together, although in this case it is a button click. After the user has drawn a symbol, (s)he clicks the 'recognize' button which takes all strokes drawn since the last click of the button and groups them together. While this may be adequate for some domains, this type of interaction is not desirable in a free-sketch environment.

A more automated, yet domain-specific, approach is AC-SPARC, presented by Gennari *et al.* [10]. It takes uninterrupted input from the user as (s)he draws an analog circuit diagram and automatically detects the groups of stroke segments (fragments) and recognizes them using a feature-based multi-stroke recognizer. AC-SPARC uses an "ink-density" heuristic, computed for the preceding n strokes, to determine boundaries between symbols. The algorithm requires that a symbol be finished before the next one is begun (i.e., interspersing of symbols is not allowed), and places an upper-bound on the number of segments in a cluster to simplify and enable accurate grouping. These constraints, along with relying solely on ink-density to locate clusters, prevent the system from being extended to other domains.

Kara *et al.* have developed two systems which attempt free-sketch recognition: SimuSketch [21] and VibroSketch [20]. SimuSketch is a graphical interface for Matlab's Simulink software package. It uses "mark-group-recognize" to understand the sketch; this technique is part of the inspiration for our work. The marker symbols, which are objects that can be easily found in the sketch, simplify grouping. In this application, arrows act as marker symbols. Users draw Simulink objects that are connected by arrows. The system automatically detects the marker arrows and associates the remaining strokes with the nearest arrow head or tail. Strokes that are associated with the same arrow endpoint or that have overlapping bounding boxes are clustered together to form the sketched Simulink objects. The mark-group-recognize approach is also used in VibroSketch to analyze vibratory systems. However, the marker symbols are no longer arrows, and instead are mechanical bodies and grounds. By removing the marker symbols from the vibratory sketch, the remaining objects (springs, dampers, and applied forces) are spatially separated from each other, making the grouping part significantly easier. VibroSketch groups the remaining objects using an agglomerative hierarchical algorithm. These two systems are domain-specific and required hand tuning to obtain effective marker symbols. Our work effectively generalizes the marking process by separating the sketch into multiple classes, and it improves the grouping.

Shilman *et al.* have developed a technique which performs a "layout analysis" to identify strokes to cluster together based on how well they line up [41]. It first forms strokes into "words," then "lines," and finally "paragraphs" using temporal information, followed by spatially-based merges. It then classifies the strokes as either text or graphics based on the number of fragments in the group and the linear regression error of the fragments' centroids. This work is similar to the work presented here in that it focuses on grouping (layout analysis), and does not go on to recognize the individual components. Instead it simply classifies the objects (and the strokes within them) as text or graphics. This technique is highly tuned for text and cannot easily be generalized to diagrammatic sketches.

In other work by Shilman *et al.*, a statistical visual language model is created for a domain to enable spatially-based parsing of ink strokes [39]. This method uses a hand-generated grammar for each new domain, which requires a domain expert to implement because the constraints between variables are hand-coded thresholds. Although new thresholds are later learned from examples which overwrite the hand-coded values, this type of system requires too much expert interaction for it to be considered an easily extensible method. In later work Shilman and Viola developed a method to group and recognize text and graphics in a purely spatial manner [40]. This technique forms a neighborhood graph of the strokes and uses A* search to find the solution which best explains all strokes in the sketch given a cost function and the recognition results. It uses a vision-based recognizer similar to [29] to evaluate hypothetical clusters, and is able to detect "garbage" stroke groupings. This technique has a few limitations, namely that there is a fixed threshold for determining the neighborhood graph, a restriction on the maximum size of a group, and what appears to be a computational complexity that is exponential in the number of strokes on the page.

Alvarado and Davis present an approach to full sketch recognition which uses dynamically constructed Bayes nets and a bottom-up plus top-down hypothesis generation and recognition scheme which uses context to aid understanding [1]. As part of the bottom-up step, the strokes are broken up into fragments which are then recognized as geometric primitives. The primitives are then hypothesized to be part of various shapes. The top-down step looks at hypothesized shapes and tries to match geometric primitives to any missing constraints to complete the hypothesis. The most likely set of non-conflicting hypotheses is used. The probabilistic framework makes this approach a good fit for sketch understanding because of its ability to handle ambiguity. This method suffers from two limitations. First, as the number of strokes in the sketch increases, the computation time required to form and evaluate all hypotheses grows to a level that is not manageable for interactive systems. Second, all strokes in a sketch require an explanation, a problem when the sketch contains overstroking, or erroneous/meaningless strokes.

Sezgin and Davis attempt to solve full sketch recognition using hidden Markov models (HMMs) to exploit the temporal nature of sketching [38]. However, this technique limits the user to drawing objects sequentially, puts a cap on the number of stroke primitives in an object, and requires the user to draw an object in a previously defined stroke order. They studied human drawing behavior as it relates to the number of unique stroke orderings and found that although the theoretical number of stroke orderings is quite large, only a small subset of those orderings is actually used, especially for a single user. Later work [37] was focused on removing the requirement to finish one shape before starting the next (i.e., allow interspersed drawings), and was accomplished by using dynamic Bayesian networks (DBNs) instead of HMMs. This work appears to provide an efficient full sketch recognition technique in an almost freely-drawn manner, however the system is unable to handle multiple interspersings. The system also requires that the strokes be broken down into meaningful primitives. Further, the system seems to be able to handle only line segment primitives (some of the features are based on properties of line segments), restricting it to domains with shapes composed entirely of line segments.

Feng *et al.* have created a system [8] for recognizing sketches that is similar to Sezgin and Davis' work [37]. They use a 2D dynamic programming approach to parse segments and then recognize them using either a neural network classifier (for circuit components) or a method based on least-squares fit error for connectors (wires). This approach has some limitations. Perhaps the biggest is the use of a specially tuned recognizer for connectors, making the approach more difficult to generalize than Sezgin *et al.*'s. The system also places a number of constraints on shapes to simplify the segment parsing (i.e., clustering). These include: a maximum number of stroke segments, a maximum distance between stroke segments, and a maximum number of time-jumps (interspersings).

Cowans and Szummer attempt simultaneous parsing and classification of stroke segments using a graphical model based on the potentials of the individual fragments and the pairwise potentials between fragments [7]. They report low error rates on labeling the fragments, however they do not report accuracies for grouping. The method restricts stroke fragments to straight lines and the computational complexity increases very rapidly with the maximum allowable clique (group) size.

7.2 Symbol Recognition

Symbol recognition is defined here as the process of recognizing an isolated set of one or more strokes. It can be split into two general types: single-stroke and multi-stroke. One of the most active areas of sketch recognition research has been the development of multi-stroke symbol recognizers. Most of these recognizers use either visual or shape information.

One of the most common approaches to multi-stroke recognition is to use the visual information. One example of this is by Kara and Stahovich, who created an image-based recognizer which compares an unknown symbol to exemplars and computes four similarity metrics between the bitmap images of the unknown and exemplar symbols [22]. More recently Ouyang and Davis created a visual approach to symbol recognition that extracts feature images corresponding to stroke orientations and endpoints. It then compares the set of feature images to templates that are stored hierarchically, which decreases computation during comparison [29].

A recognizer by Hse and Newton converts the visual representation of a symbol into features representing the Zernike Moments [15]. Calhoun *et al.* use a trainable multi-stroke recognizer which learns definitions in terms of the symbol's constituent geometric primitives (lines and arcs) [6]. Here, a stroke is segmented using its speed and curvature, and the segments are fit to geometric primitives. Taking a different approach, Lee *et al.* use a graph-based recognizer for multi-stroke symbols [25]. This recognizer uses statistical models for symbol definitions and the geometry as well as topology of the symbol to determine the best match. Hammond and Davis have developed a method which converts structural symbol descriptions into domain-specific recognizers [13]. They use their LADDER language to describe relations between geometric primitives in a symbol, which are used to identify an unknown symbol. These four approaches are representative of a larger body of non-visual multi-stroke symbol recognizers.

Single-stroke recognition techniques are typically simpler than multi-stroke techniques and are usually referred to as gesture recognizers. One of the earliest gesture recognizers was developed by Rubine [35]. It computes geometric and dynamic features of the stroke for input to a linear discriminator for classification. More recent work by Wobbrock *et al.* [45] uses a more structural description (resampled points along the stroke's path) and a simple template-matching algorithm to determine the most likely result. Because of the restriction to recognize single strokes, these types of techniques are not suited to full sketch recognition.

Primitive recognition is related to single-stroke symbol recognition. Here, a single stroke is segmented by one of a number of algorithms ([42, 46, 47]), and the resulting segments are then recognized as different geometric primitives – usually lines and arcs, however some recognizers include more complex primitives [31, 32, 34, 42, 48].

There is a large body of work on symbol recognition, however this work requires separate methods for grouping. Our work provides this grouping technique, and can be used in combination with these recognizers to create a full sketch recognition system.

7.3 Single-Stroke Classification

Identifying a stroke according to its general type is referred to here as single-stroke classification. It is similar to single-stroke recognition, and it could indeed be argued that some methods can be used for both. Here we make the distinction that single-stroke classification refers to labeling strokes in a full sketch, as compared to recognition which looks at strokes more in isolation. Additionally, classification differs from recognition in the specificity of the output. Classification identifies a *general* label while recognition identifies a *specific* label. For instance, a classifier might identify a stroke as a character, while a recognizer identifies it as the letter *a*.

The most common application of single-stroke classification is dividing text from graphics strokes [3, 5, 30, 44], allowing a system to send the text to a specialized handwriting recognition system, while the graphics are usually not processed further. Although classification is used for a different purpose than symbol and primitive recognition (7.2), there is a large amount of overlap between algorithms and features used for the different tasks.

Bishop *et al.* describe an algorithm for separating text from graphics using a combination of the features of the individual strokes and gaps between them [5]. An initial classification for each stroke is obtained with a multi-layer perceptron (MLP) using each stroke's geometric features as the input. The system then uses a hidden Markov model (HMM) to optimize the sequence of stroke states using the transition probabilities between class labels (uni-partite) or a combination of the stroke states and the gap states (bi-partite). This method achieved low error rates on two separate data sets, however the prior class distribution is weighted heavily to text.

Wang *et al.* improve on Bishop *et al.*'s work by addressing the class distribution issues [44]. They use an adjustment to the outputs of the MLPs to reflect *a priori* probabilities, yielding more accurate estimates of the posterior probabilities.

Using a collection of ink features, Patel *et al.* perform text and graphic classification for diagram recognition [30]. They report error rates as low as 10.8% for shapes and 8.8% for text, although it is unclear how they separated the training and test sets. With a new diagram set their error rates increased, although it is unclear how the classifier was trained. The algorithm computes many features (some of which are used or adapted in this dissertation) and uses their values as inputs to a decision tree classifier. However it should be noted that they restrict themselves to classifying text vs. non-text while we consider multiple classes, and that we report higher accuracies.

One of the most recent approaches to distinguishing text from graphics strokes is presented by Bhat and Hammond [3]. Here they calculate the "entropy" of the stroke as it relates to the sequence of angles between consecutive points. This approach achieved high accuracy in their tests. However, it usually performed worse than our method on the domains presented in this dissertation (see Section 5.1).

Not all previous stroke classification has been used for text versus non-text. Bischel *et al.* present a method to separate gesture from non-gesture strokes in unconstrained multi-modal descriptions of mechanical devices [4]. This method is similar to ours in that there are many features computed which are sent to a trained MLP classifier, however the feature set, classifier, and the number of output classes differ. Many of the features used are derived from transcripts of device description (verbal communication between users), which is a modality that is not available to our system.

Recognizing the importance of context in stroke classification, Szummer and Qi developed a conditional random field (CRF) model of site and interaction potentials to classify object and connector stroke fragments in hand-drawn diagrams [43]. This approach assumes that text has already been separated from graphics, and allows the classifications of stroke fragments to influence the classifications of neighboring strokes.

Improving on their previous technique, Qi *et al.* use a Bayesian CRF, with model averaging, which avoids the over-fitting problems that can occur with maximum likelihood (ML) and maximum a posteriori (MAP) trained CRFs [33].

Our work goes beyond each of the methods here by presenting results of multi-class labeling of strokes. It is also of comparable or better accuracy than previous methods. Further, we are the first to use single-stroke classification as a way to create spatial and temporal separation between objects to facilitate efficient and accurate grouping.

Chapter 8

Conclusions

We have developed a novel and effective method to group strokes into objects in hand-drawn diagrams. In this chapter we review the contributions detailed in this dissertation.

- Our two-step grouping technique is the first to demonstrate that separating the strokes in a sketch into classes, using a general classification algorithm, is an effective method for enabling accurate stroke grouping. Single-stroke classification is important because it creates spatial and temporal separation between objects.
- This work is the first to develop general-purpose techniques for stroke grouping independent of recognition. To aid future work, we have provided results for two grouping methods to serve as benchmarks. Additionally, our grouping accuracy metrics are meaningful ways to describe a cluster's correctness, and are based on the length of the strokes in the cluster and the number of incorrectly grouped strokes.

- Our single-stroke classification technique is used to separate the strokes in a sketch, and is the first reported work to classify strokes into three or more classes. Additionally, it performs as well as or better than two previous state-of-the-art techniques for text versus non-text classification. Our set of features extends previous work and includes many contextual features, which were found to be some of the most important features for accurate classification.

- Finally, we have developed a method for grouping strokes in polynomial time using pairwise classification of strokes to determine which pairs belong to the same object. This is in contrast to many previous methods which require exponential time or that limit drawing freedom.

Our two-step grouping technique is able to accurately cluster strokes much faster than the user can draw, thus it can be used in an interactive system. We have shown that separating the strokes in a sketch into several classes significantly reduces the grouping complexity. Our techniques are general, and can be applied to a new domain with little effort and no hand-coding. Further, they do not impose unnatural constraints on how a user draws, such as requiring a shape to be finished before beginning a new shape, or limiting the user to certain primitive shapes. Instead, our techniques learn how people naturally draw from training examples. There has been much progress in sketch recognition and interpretation, but previous approaches have lacked a robust and efficient, general-purpose grouping algorithm. Our work is an important step towards solving this challenging step, and provides a framework upon which sketch-based tools can be created.

Appendix A

Digital Circuit Diagrams

A.1 Examples of Single-Stroke Classifier Performance

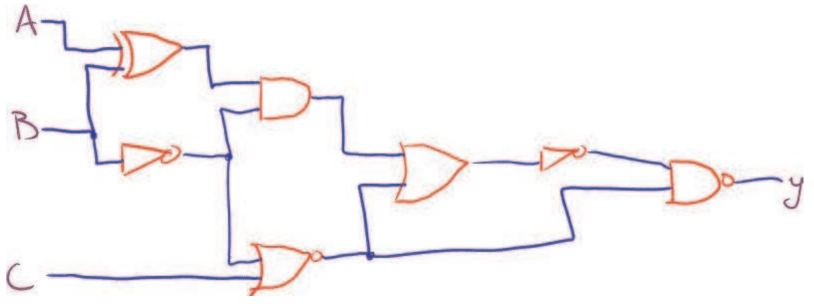


Figure A.1: Example of the single-stroke classifier performing well on a digital circuit sketch.

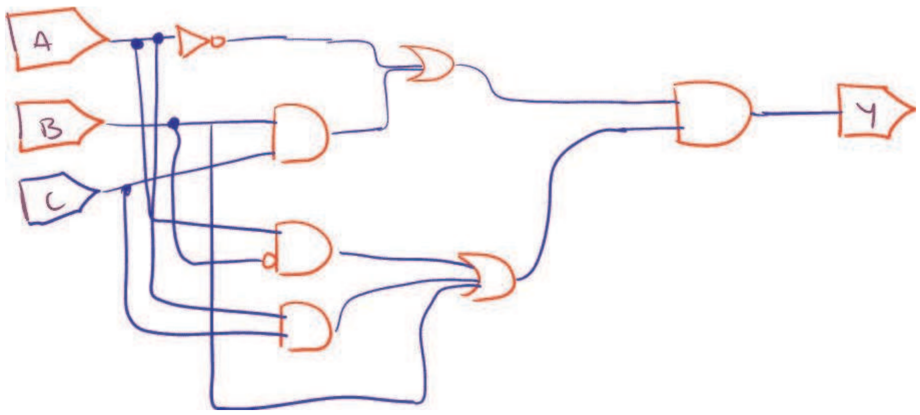


Figure A.2: Example of the single-stroke classifier performing poorly on a digital circuit sketch.

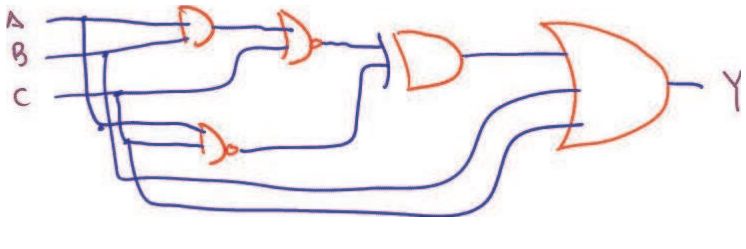


Figure A.3: Example of average performance of the single-stroke classifier on a digital circuit sketch.

A.2 Sample User-Study Sketches

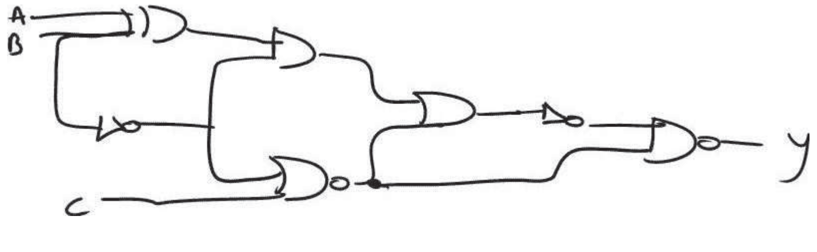


Figure A.4: Sample digital circuit sketch, drawn on a Tablet PC by user number 2, while copying a circuit.

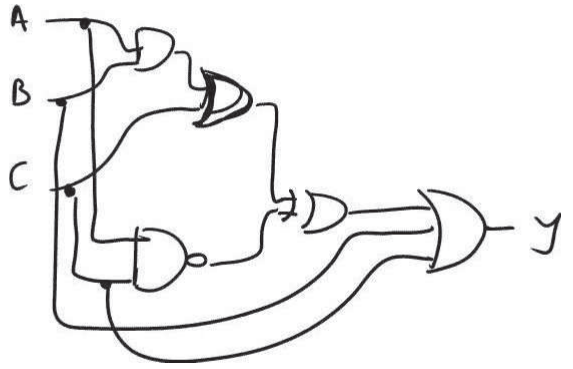


Figure A.5: Sample digital circuit sketch, drawn on a Tablet PC by user number 2, while synthesizing a circuit from an equation.

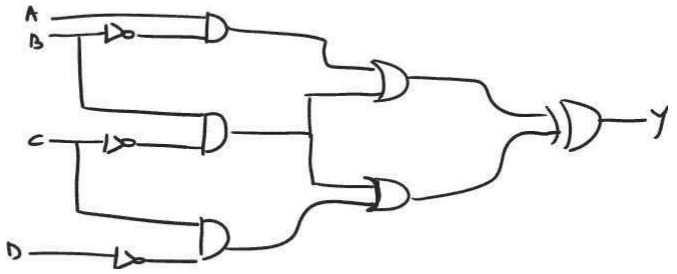


Figure A.6: Sample digital circuit sketch, drawn using an ink pen on a digitizing tablet by user number 2, while copying a circuit.

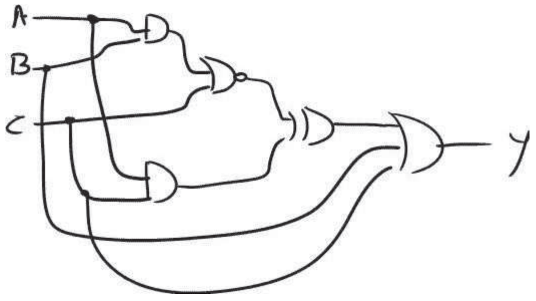


Figure A.7: Sample digital circuit sketch, drawn using an ink pen on a digitizing tablet by user number 2, while synthesizing a circuit from an equation.

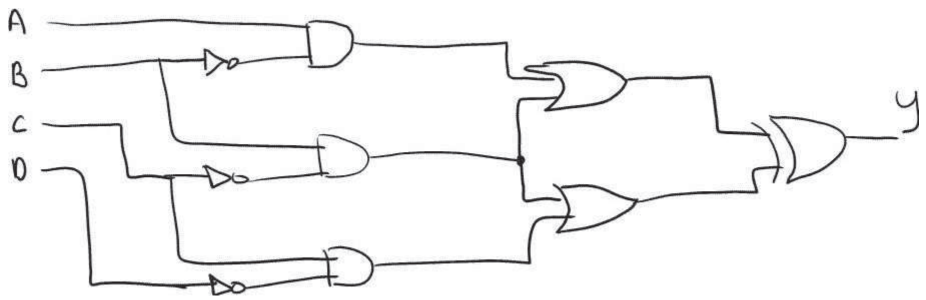


Figure A.8: Sample digital circuit sketch, drawn on a Tablet PC by user number 5, while copying a circuit.

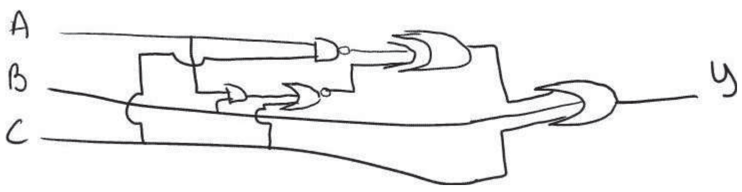


Figure A.9: Sample digital circuit sketch, drawn on a Tablet PC by user number 5, while synthesizing a circuit from an equation.

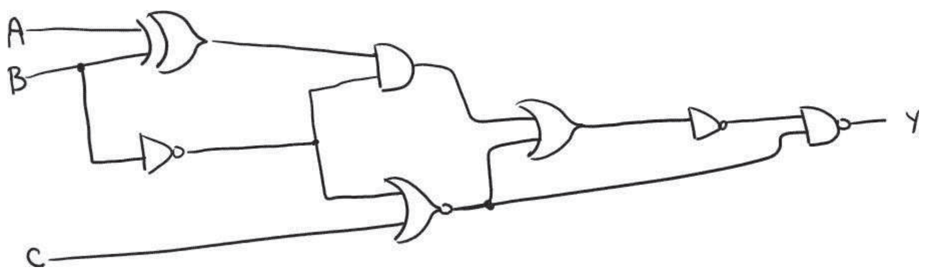


Figure A.10: Sample digital circuit sketch, drawn on a Tablet PC by user number 7, while copying a circuit.

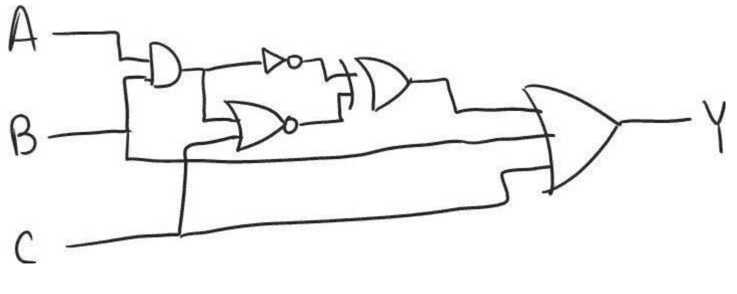


Figure A.11: Sample digital circuit sketch, drawn on a Tablet PC by user number 10, while synthesizing a circuit from an equation.

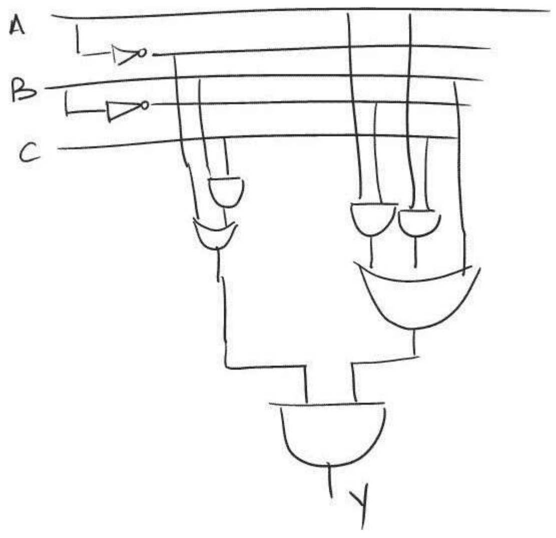


Figure A.12: Sample digital circuit sketch, drawn on a Tablet PC by user number 14, while synthesizing a circuit from an equation.

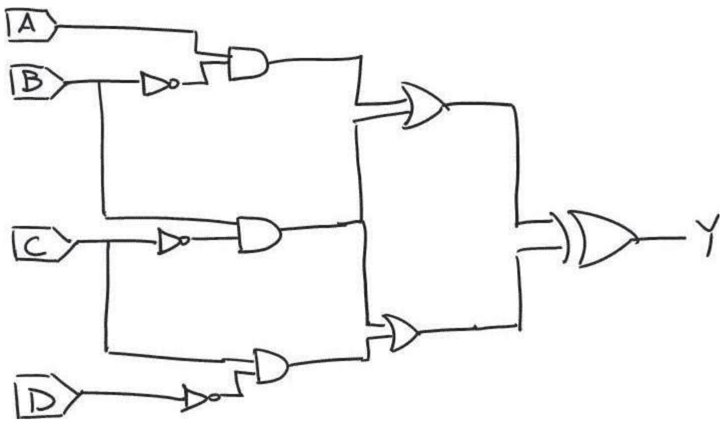


Figure A.13: Sample digital circuit sketch, drawn on a Tablet PC by user number 16, while copying a circuit.

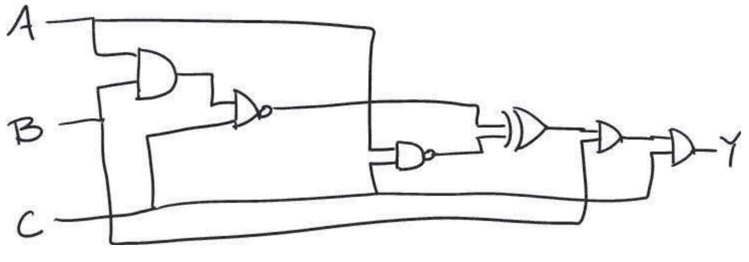


Figure A.14: Sample digital circuit sketch, drawn on a Tablet PC by user number 16, while synthesizing a circuit from an equation.

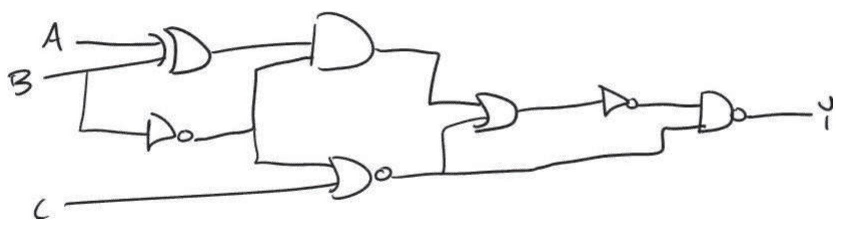


Figure A.15: Sample digital circuit sketch, drawn on a Tablet PC by user number 23, while copying a circuit.

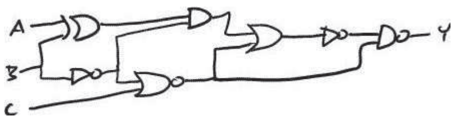


Figure A.16: Sample digital circuit sketch, drawn using an ink pen on a digitizing tablet by user number 23, while copying a circuit.

Appendix B

Statics Solutions

B.1 User Study Prompts

Compute the force supported by the pin at *A* for the slip-joint pliers under a grip load of *P*.

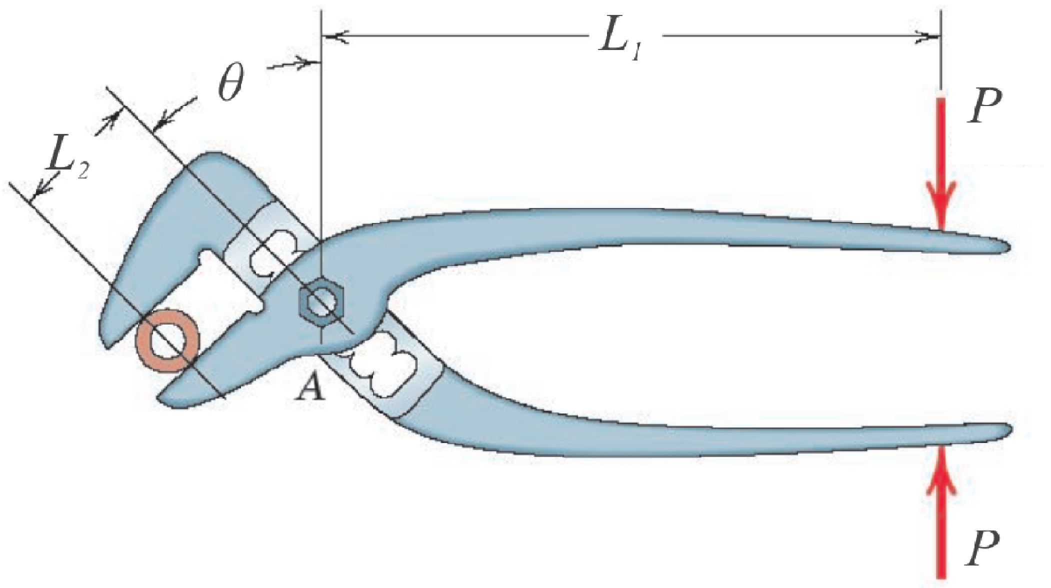


Figure B.1: Prompt for a statics problem used in the study. Image and prompt are from Engineering Mechanics: Statics [27], copyright John Wiley and Sons, Inc., used with permission. The image and prompt were modified, replacing all numeric values with variables.

The simple crane supports the load with weight W. Determine the tension T in the cable and the magnitude of the pin reaction at O.

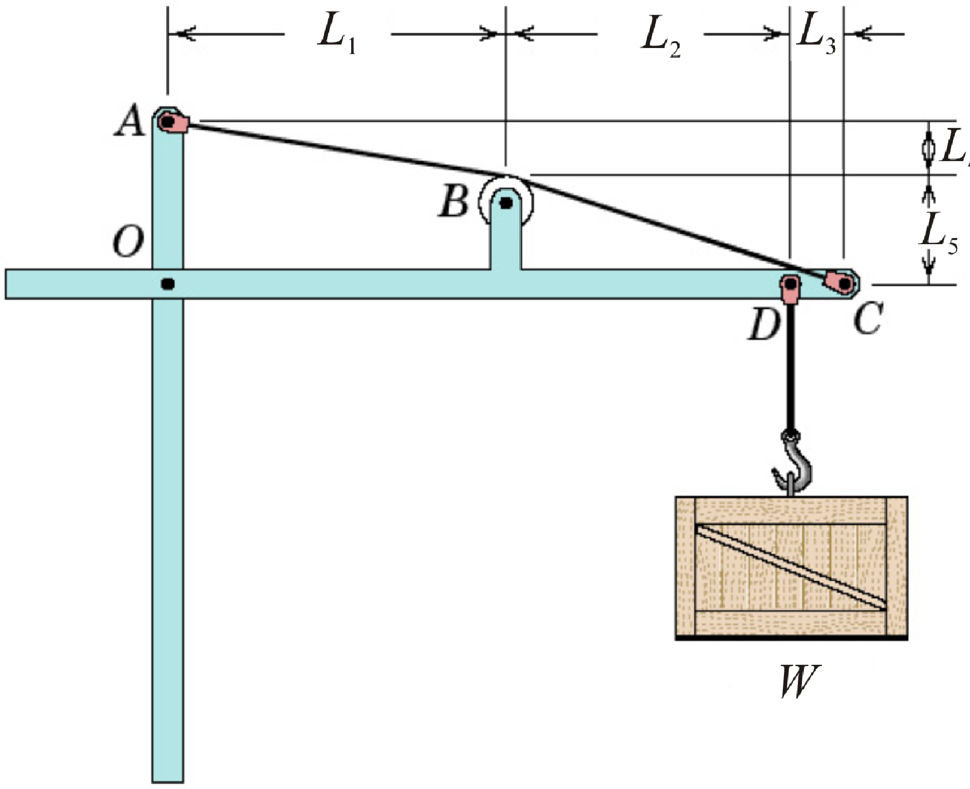


Figure B.2: Prompt for a statics problem used in the study. Image and prompt are from Engineering Mechanics: Statics [27], copyright John Wiley and Sons, Inc., used with permission. The image and prompt were modified, replacing all numeric values with variables.

When the crank *AB* is vertical, the beam *CD* is horizontal and the cable makes an angle θ with the horizontal. Compute the moment *M* required for equilibrium of the frame.

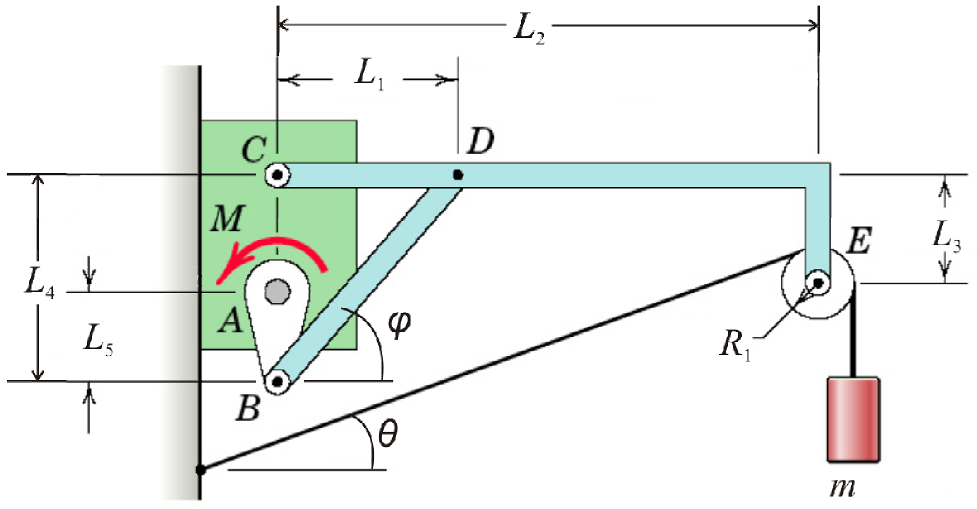


Figure B.3: Prompt for a statics problem used in the study. Image and prompt are from Engineering Mechanics: Statics [27], copyright John Wiley and Sons, Inc., used with permission. The image and prompt were modified, replacing all numeric values with variables.

Determine the compression force *C* exerted on the can for an applied force *P* when the can crusher is in the position shown. Note that there are two links *AB* and two links *AOD*, with one pair of linkages on each side of the stationary portion of the crusher. Also, pin *B* is on the vertical centerline of the can. Finally, note that small square projections *E* of the moving jaw move in recessed slots of the fixed frame.

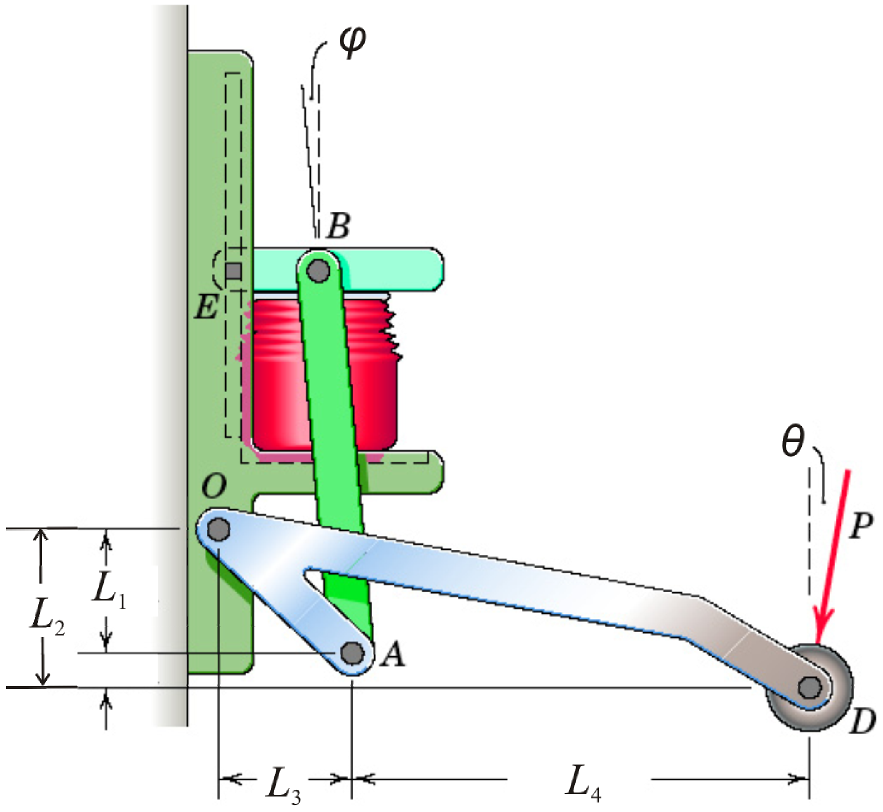


Figure B.4: Prompt for a statics problem used in the study. Image and prompt are from Engineering Mechanics: Statics [27], copyright John Wiley and Sons, Inc., used with permission. The image and prompt were modified, replacing all numeric values with variables.

The car hoist allows the car to be driven onto the platform, after which the rear wheels are raised. If the loading from both rear wheels is *W*, determine the force in the hydraulic cylinder *AB*. Neglect the weight of the platform itself. Member *BCD* is a right-angle bell crank pinned to the ramp at *C*.

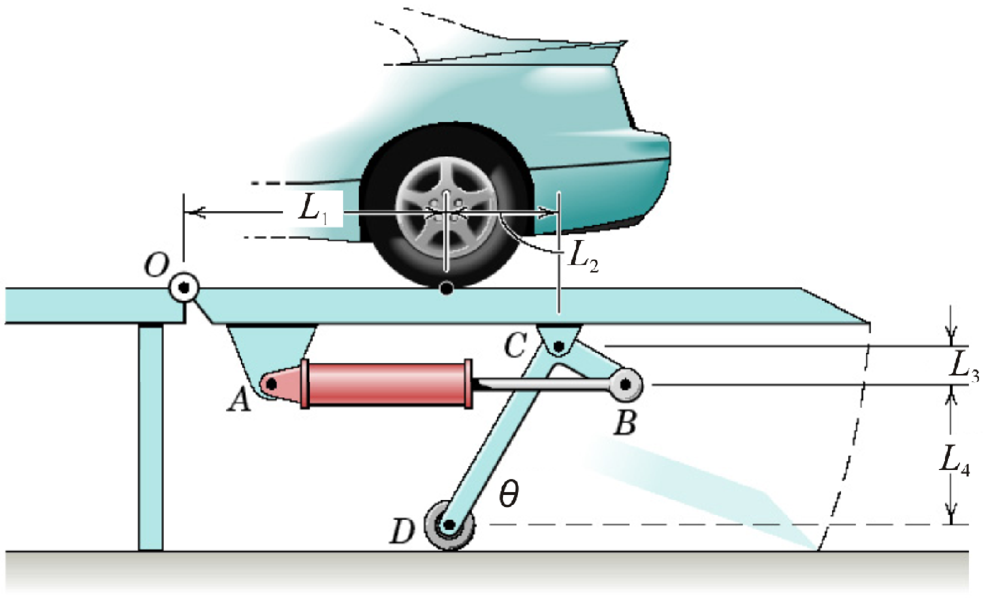


Figure B.5: Prompt for a statics problem used in the study. Image and prompt are from Engineering Mechanics: Statics [27], copyright John Wiley and Sons, Inc., used with permission. The image and prompt were modified, replacing all numeric values with variables.

B.2 Examples of Single-Stroke Classifier Performance

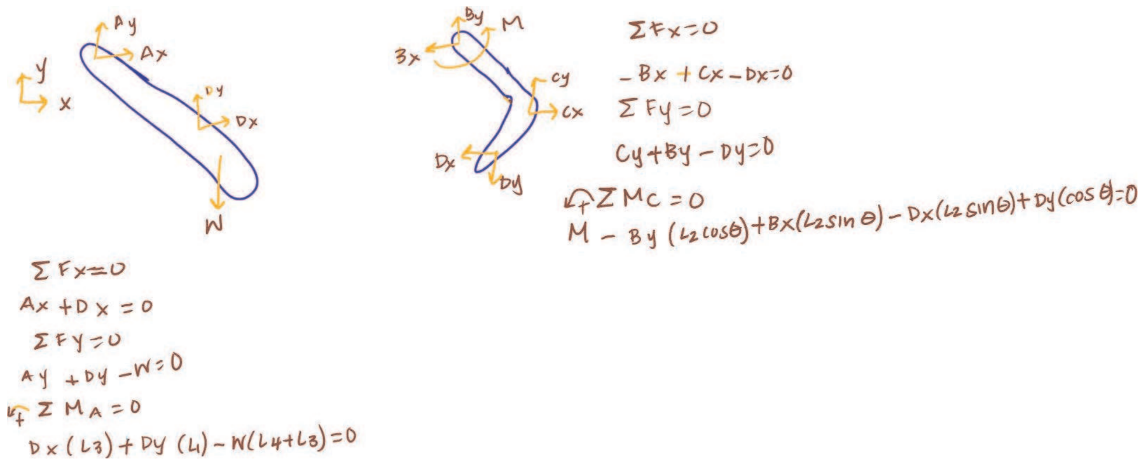


Figure B.6: Example of the single-stroke classifier performing well on a complete statics solution sketch.

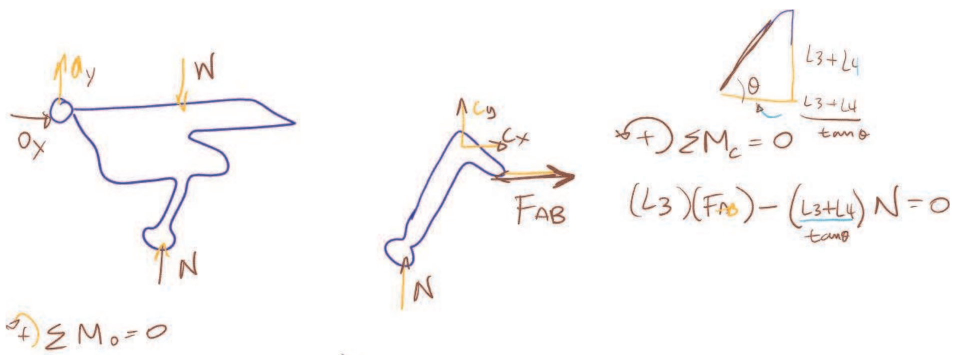
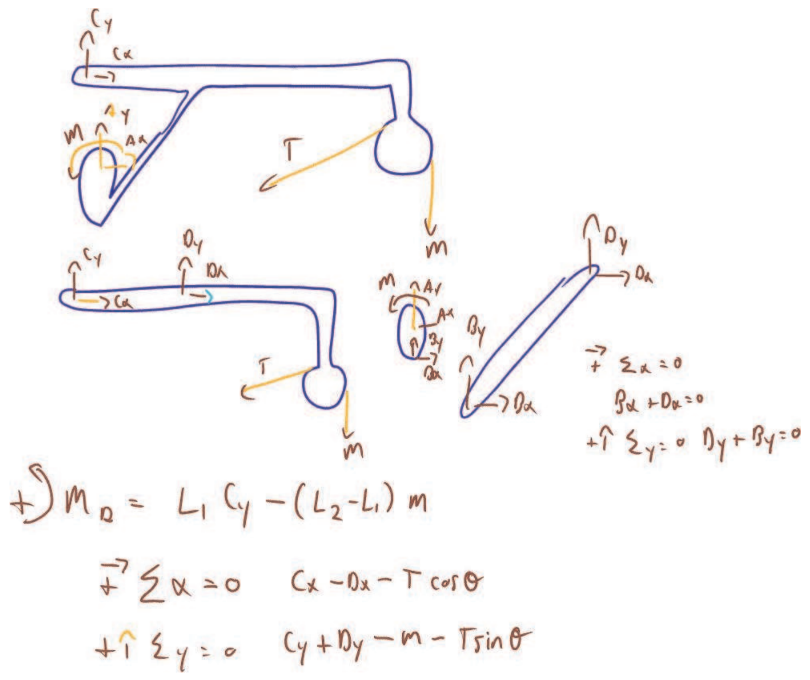


Figure B.7: Example of the single-stroke classifier performing poorly on a complete statics solution sketch.



C_y
C_x
A_y
A_x
m
T
m
C_y
D_y
D_x
C_x
T
m
m
A_y
A_x
B_y
B_x
B_y
B_x
D_y
D_x
$\overset{\rightarrow}{+} \Sigma x = 0$
$B_x + D_x = 0$
$+\uparrow \Sigma y = 0 \quad D_y + B_y = 0$
$+\circlearrowright M_D = L_1 C_y - (L_2 - L_1) m$
$\overset{\rightarrow}{+} \Sigma x = 0 \quad C_x - D_x - T\cos\theta$
$+\uparrow \Sigma y = 0 \quad C_y + D_y - m - T\sin\theta$

Figure B.8: Example of the average performance of the single-stroke classifier on a complete statics solution sketch.

B.3 Sample User-Study Sketches

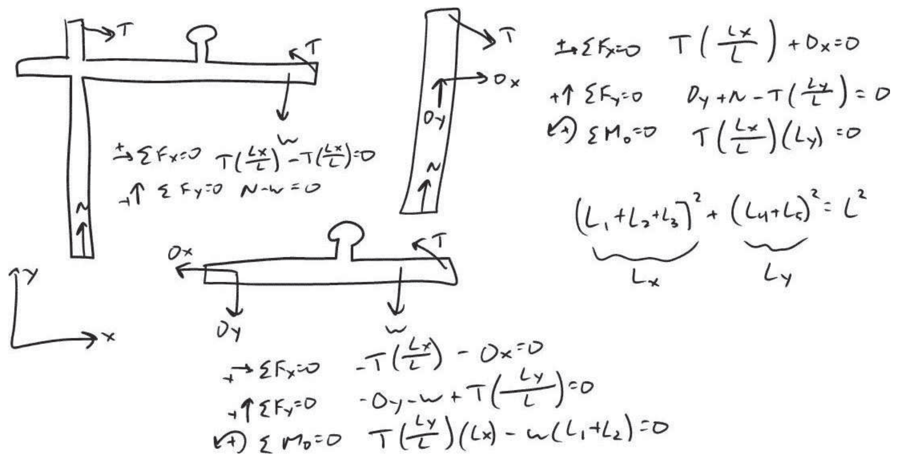

T
T
w
$\xrightarrow{+} \Sigma F_x = 0 \quad T(\frac{Lx}{L}) - T(\frac{Lx}{L}) = 0$
$+\uparrow \Sigma F_y = 0 \quad N - w = 0$
N
T
Ox
Oy
N
$\xrightarrow{+} \Sigma F_x = 0 \quad T(\frac{Lx}{L}) + Ox = 0$
$+\uparrow \Sigma F_y = 0 \quad Oy + N - T(\frac{Ly}{L}) = 0$
$\circlearrowleft + \; \Sigma M_o = 0 \quad T(\frac{Lx}{L})(Ly) = 0$
$\underbrace{(L_1 + L_2 + L_3)}_{Lx}^2 + \underbrace{(L_4 + L_5)}_{Ly}^2 = L^2$
y
x
Ox
T
Oy
w
$\xrightarrow{+} \Sigma F_x = 0 \quad -T(\frac{Lx}{L}) - Ox = 0$
$+\uparrow \Sigma F_y = 0 \quad -Oy - w + T(\frac{Ly}{L}) = 0$
$\circlearrowleft + \; \Sigma M_o = 0 \quad T(\frac{Ly}{L})(Lx) - w(L_1 + L_2) = 0$

Figure B.9: Sample of a statics solution sketch, drawn by user 1, for the crane problem shown in Figure B.2

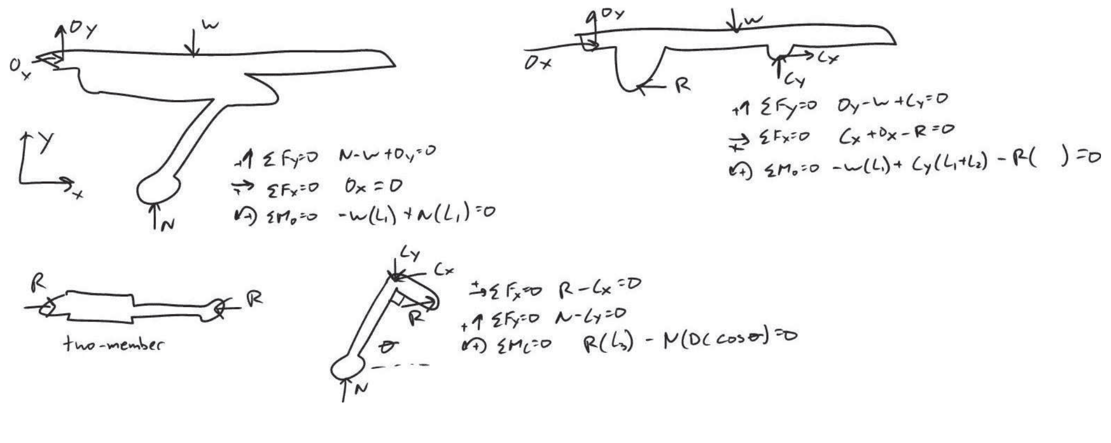
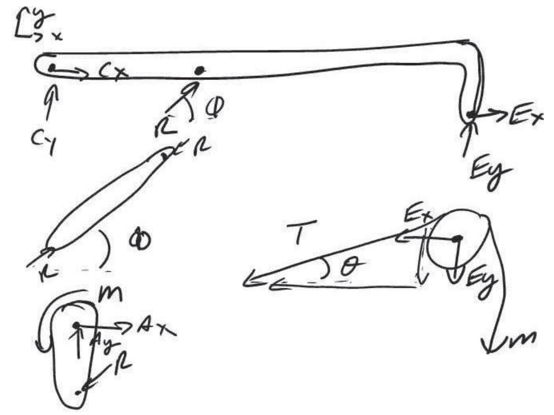

Oy
w
Ox
y
x
N
$+\uparrow \Sigma F_y = 0 \quad N - w + O_y = 0$
$\xrightarrow{+} \Sigma F_x = 0 \quad Ox = 0$
$\circlearrowleft + \; \Sigma M_o = 0 \quad -w(L_1) + N(L_1) = 0$
Oy
w
Ox
R
Cy
Cx
$+\uparrow \Sigma F_y = 0 \quad Oy - w + Cy = 0$
$\xrightarrow{+} \Sigma F_x = 0 \quad Cx + Ox - R = 0$
$\circlearrowleft + \; \Sigma M_o = 0 \quad -w(L_1) + Cy(L_1 + L_2) - R(\quad) = 0$
R
R
two-member
Cy
Cx
R
θ
N
$\xrightarrow{+} \Sigma F_x = 0 \quad R - Cx = 0$
$+\uparrow \Sigma F_y = 0 \quad N - Cy = 0$
$\circlearrowleft + \; \Sigma M_C = 0 \quad R(L_3) - N(DC \cos\theta) = 0$

Figure B.10: Sample of a statics solution sketch, drawn by user 1, for the car hoist problem shown in Figure B.5



Pulley E

$\overset{+}{\uparrow}\Sigma F_y = 0$

$-mg - E_y - T\sin\theta = 0$

$\xrightarrow{+}\Sigma F_x = 0$

$-T\cos\theta - E_x = 0$

$\overset{\circlearrowleft}{+}\Sigma M_E = 0$

$-mg(R_1) + T\cos(\theta)(R_1) + T\sin\theta(R_1) = 0$

Cam AB

$\circlearrowleft\Sigma M_A = 0$

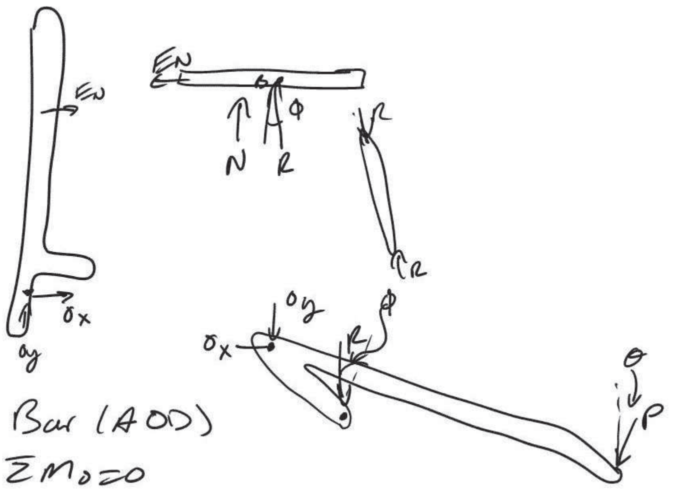
$M - R\cos(\phi)(L_5) = 0$

Bar CDE

$\overset{\circlearrowleft}{+}\Sigma M_C = 0$

$E_x(L_3) + E_y(L_2) + R\sin(\phi) = 0$

Figure B.11: Sample of a statics solution sketch, drawn by user 2, for the crank problem shown in Figure B.3



Bar (AOD)

$\Sigma M_O = 0$

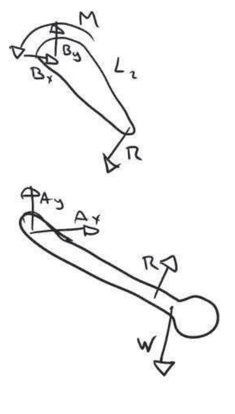
$-R\cos\phi(L_3) - R\sin\phi(L_4) - P\cos\theta(L_3+L_4) - P\sin\theta(L_2) = 0$

Bar EB

$\Sigma F_y = 0$

$N + R\cos\phi = 0 \quad ; \; N = C$

Figure B.12: Sample of a statics solution sketch, drawn by user 2, for the can crusher problem shown in Figure B.4



$\circlearrowleft^{+} \Sigma M_B = -M + R\sin\theta L_2 = 0$

$M = R\sin\theta L_2$

$\Sigma F_y = B_y - R\cos\theta = 0$

$\Sigma F_x = B_x - R\sin\theta = 0$

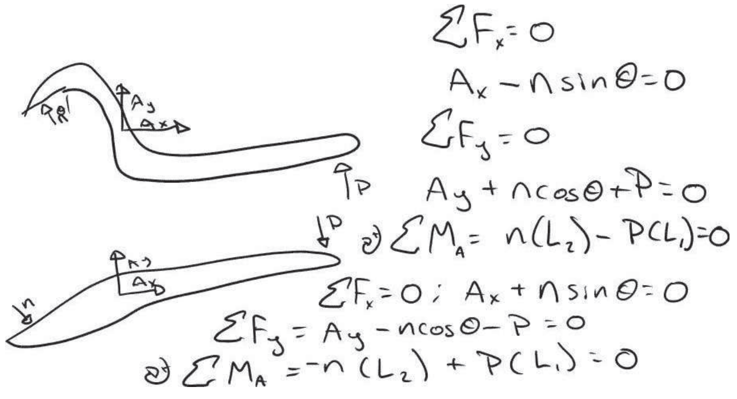
$\circlearrowleft^{+} \Sigma M_A = -R\cos\theta(L_1) + W\sin\left(\sin^{-1}\frac{L_1}{L_3}\right)(L_3+L_4) = 0$

$L_1 \quad L_3 \quad \sin\theta = \frac{L_1}{L_3}$

$\Sigma F_y = A_y + R\cos\theta - W = 0$

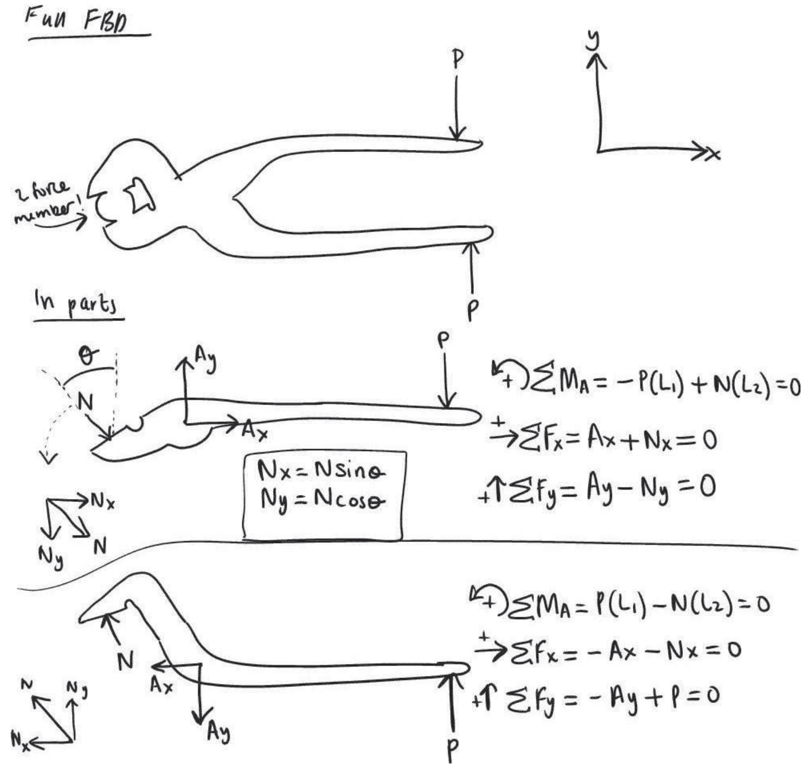
$\Sigma F_x = A_x + R\sin\theta = 0$

Figure B.13: Sample of a statics solution sketch, drawn by user 5, for the landing gear problem shown in Figure 4.3



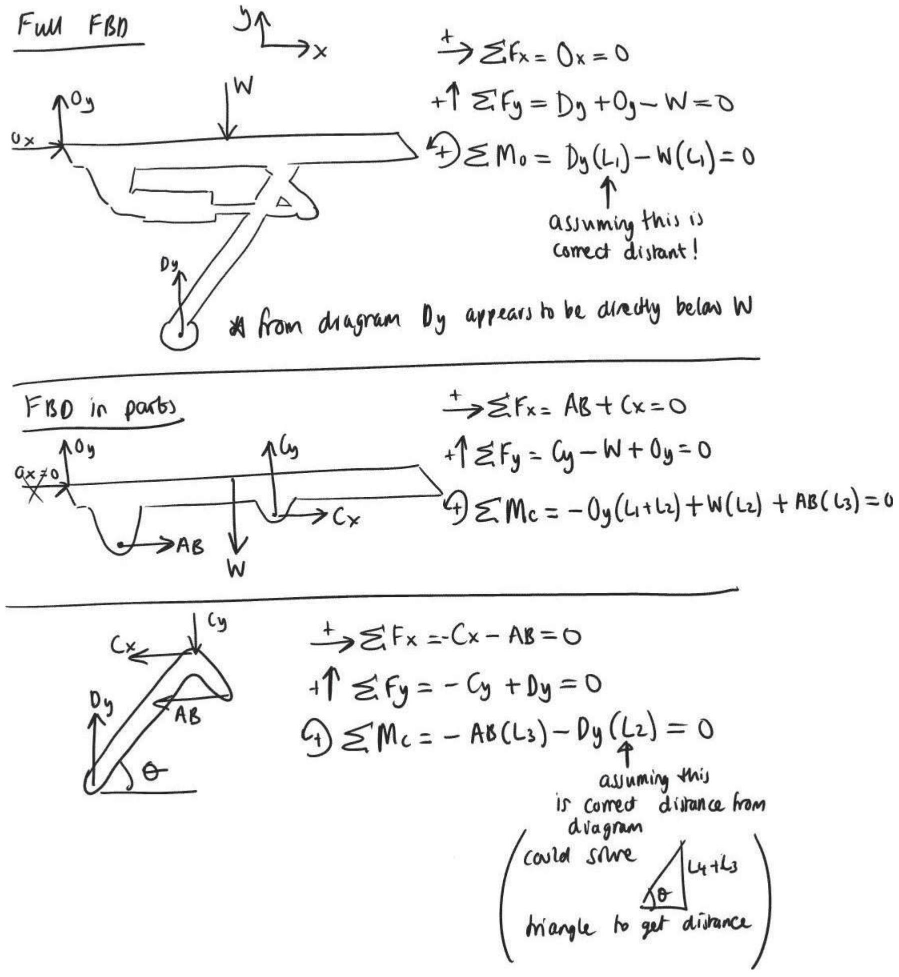
$\sum F_x = 0$
$A_x - n\sin\theta = 0$
$\sum F_y = 0$
$A_y + n\cos\theta + P = 0$
$\sum M_A = n(L_2) - P(L_1) = 0$
$\sum F_x = 0;\ A_x + n\sin\theta = 0$
$\sum F_y = A_y - n\cos\theta - P = 0$
$\sum M_A = -n(L_2) + P(L_1) = 0$
A_y
A_x
n
P

Figure B.14: Sample of a statics solution sketch, drawn by user 5, for the slip-joint pliers problem shown in Figure B.1



Full FBD
2 force member!
P
P
y
x
In parts
θ
N
A_y
A_x
P
$\sum M_A = -P(L_1) + N(L_2) = 0$
$\sum F_x = A_x + N_x = 0$
$\sum F_y = A_y - N_y = 0$
$N_x = N\sin\theta$
$N_y = N\cos\theta$
N_x
N_y
N
$\sum M_A = P(L_1) - N(L_2) = 0$
$\sum F_x = -A_x - N_x = 0$
$\sum F_y = -A_y + P = 0$
N
A_x
A_y
P
N
N_y
N_x

Figure B.15: Sample of a statics solution sketch, drawn by user 7, for the slip-joint pliers problem shown in Figure B.1



$\xrightarrow{+} \sum F_x = O_x = 0$

$+\uparrow \sum F_y = D_y + O_y - W = 0$

$\circlearrowleft + \sum M_o = D_y(L_1) - W(L_1) = 0$

assuming this is correct distant!

* from diagram D_y appears to be directly below W

$\xrightarrow{+} \sum F_x = AB + C_x = 0$

$+\uparrow \sum F_y = C_y - W + O_y = 0$

$\circlearrowleft + \sum M_C = -O_y(L_1+L_2) + W(L_2) + AB(L_3) = 0$

$\xrightarrow{+} \sum F_x = -C_x - AB = 0$

$+\uparrow \sum F_y = -C_y + D_y = 0$

$\circlearrowleft + \sum M_C = -AB(L_3) - D_y(L_2) = 0$

assuming this is correct distance from diagram

(could solve triangle to get distance)

Figure B.16: Sample of a statics solution sketch, drawn by user 7, for the car hoist problem shown in Figure B.5

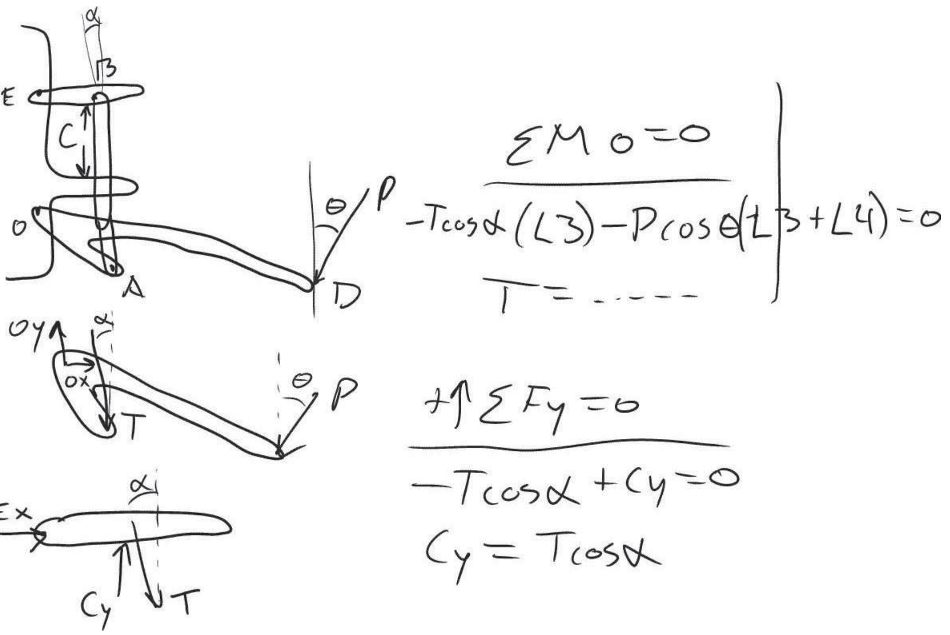


Figure B.17: Sample of a statics solution sketch, drawn by user 11, for the can crusher problem shown in Figure B.4

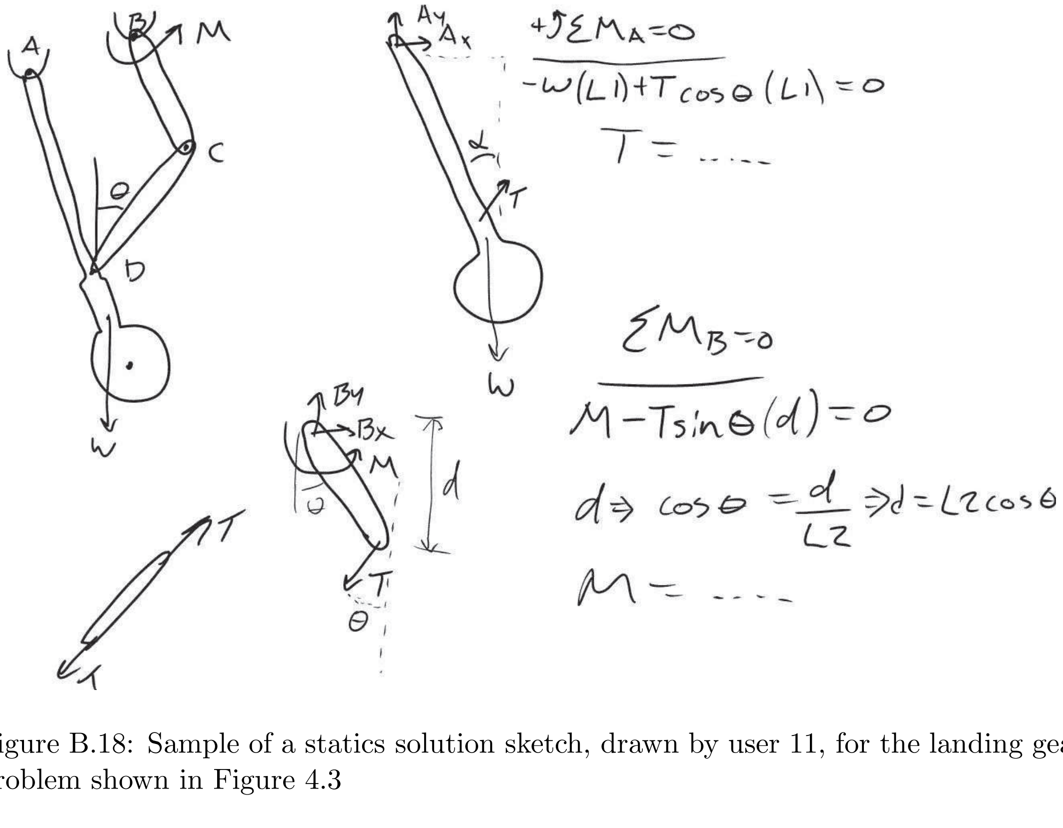


Figure B.18: Sample of a statics solution sketch, drawn by user 11, for the landing gear problem shown in Figure 4.3

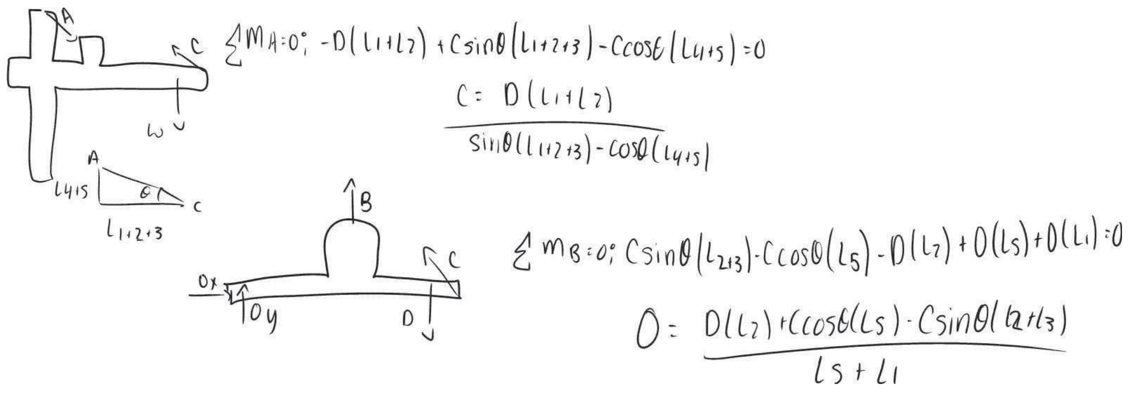

A
C
W
A
θ
C
L4+5
L1+2+3
B
C
Dx
Dy
D
∑MA=0; -D(L1+L2)+Csinθ(L1+2+3)-Ccosθ(L4+5)=0
C= D(L1+L2) / sinθ(L1+2+3)-cosθ(L4+5)
∑MB=0; Csinθ(L2+3)-Ccosθ(L5)-D(L2)+D(L5)+D(L1)=0
D= D(L2)+Ccosθ(L5)-Csinθ(L2+L3) / L5+L1

Figure B.19: Sample of a statics solution sketch, drawn by user 15, for the crane problem shown in Figure B.2

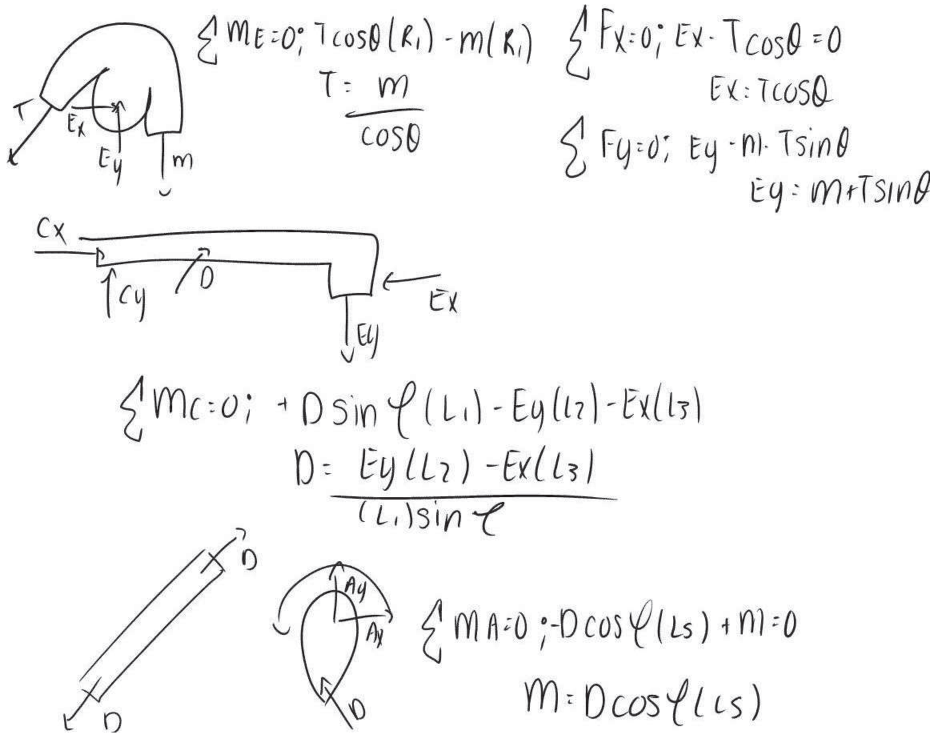


Figure B.20: Sample of a statics solution sketch, drawn by user 15, for the crank problem shown in Figure B.3

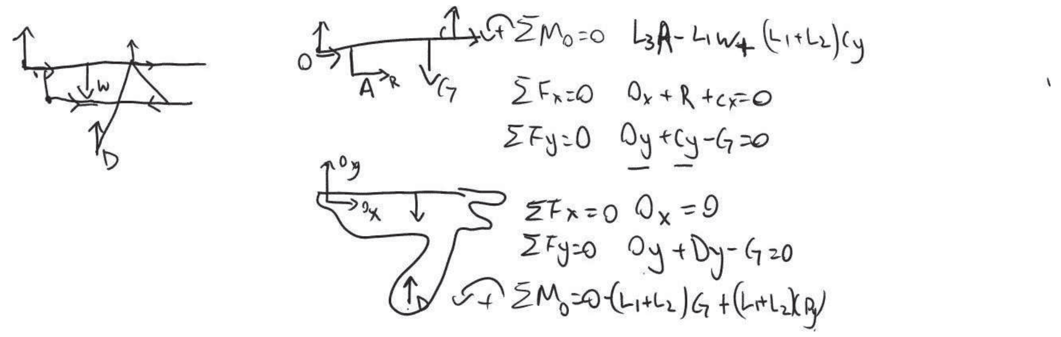

W
D
O
A
R
G
$\Sigma M_O = 0$ $L_3 A - L_1 W + (L_1+L_2) C_y$
$\Sigma F_x = 0$ $O_x + R + C_x = 0$
$\Sigma F_y = 0$ $O_y + C_y - G = 0$
O_y
O_x
D
$\Sigma F_x = 0$ $O_x = 0$
$\Sigma F_y = 0$ $O_y + D_y - G = 0$
$\Sigma M_O = 0$ $(L_1+L_2)G + (L_1+L_2)(R)$

Figure B.21: Sample of a statics solution sketch, drawn by user 16, for the car hoist problem shown in Figure B.5

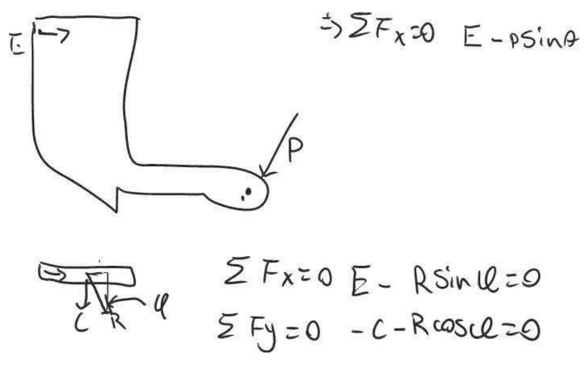

E
$\Sigma F_x = 0$ $E - P\sin\theta$
P
$\Sigma F_x = 0$ $E - R\sin\varphi = 0$
C
R
φ
$\Sigma F_y = 0$ $-C - R\cos\varphi = 0$

Figure B.22: Sample of a statics solution sketch, drawn by user 16, for the can crusher problem shown in Figure B.4

Appendix C

Family Tree Diagrams

C.1 Examples of Single-Stroke Classifier Performance

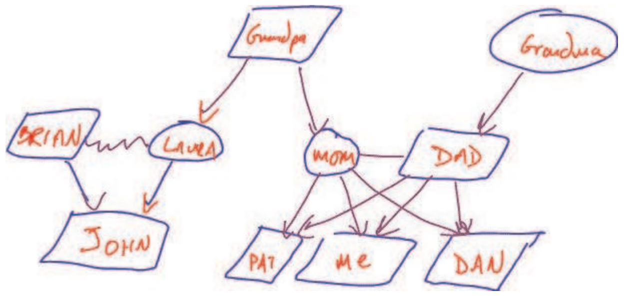


Figure C.1: Example of the single-stroke classifier performing well on a family tree sketch.

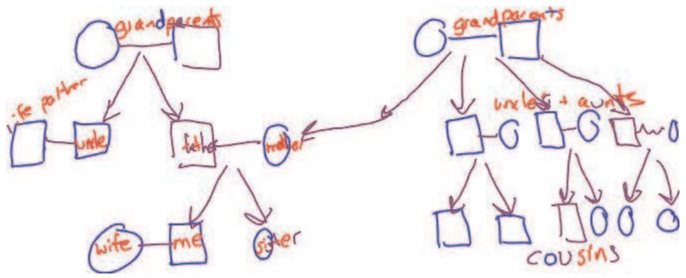


Figure C.2: Example of the single-stroke classifier performing poorly on a family tree sketch.

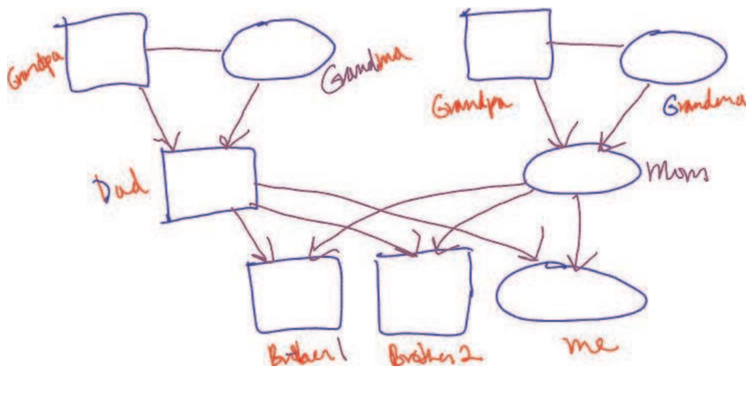


Figure C.3: Example of average performance of the single-stroke classifier on a family tree sketch.

C.2 Sample User-Study Sketches

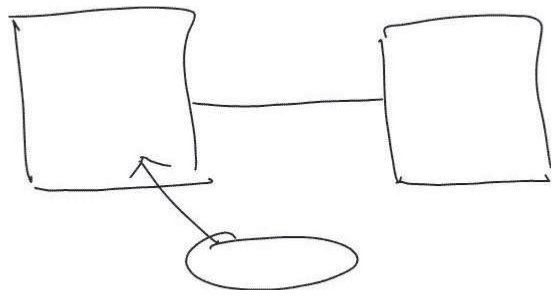


Figure C.4: Sample family tree sketch from user number 1.

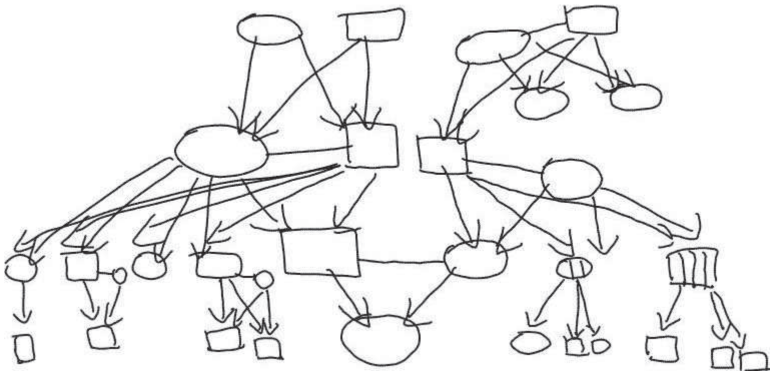


Figure C.5: Sample family tree sketch from user number 1.

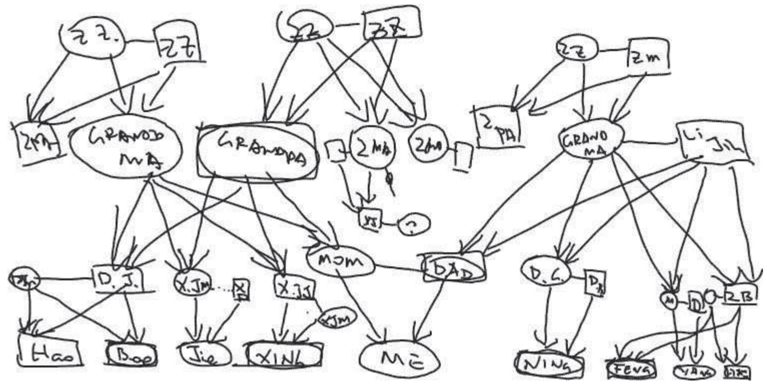


Figure C.6: Sample family tree sketch from user number 1.

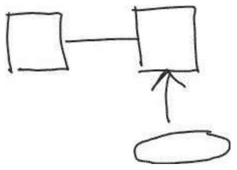


Figure C.7: Sample family tree sketch from user number 2.

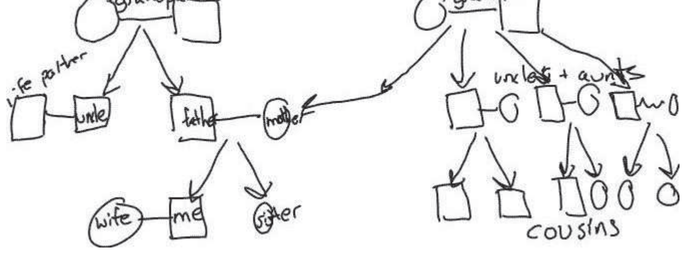


Figure C.8: Sample family tree sketch from user number 2.

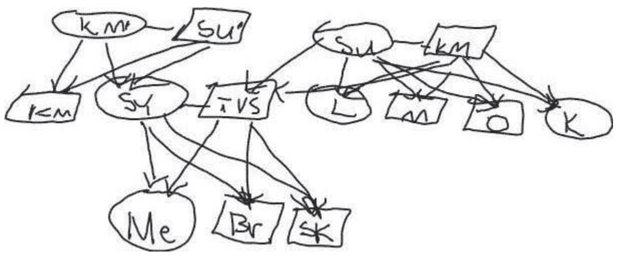


Figure C.9: Sample family tree sketch from user number 4.

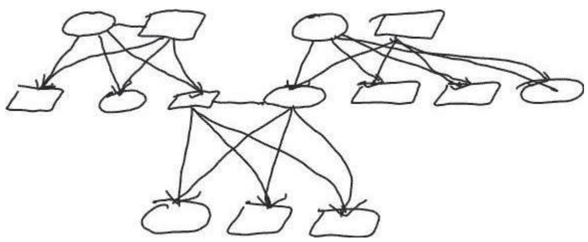


Figure C.10: Sample family tree sketch from user number 4.

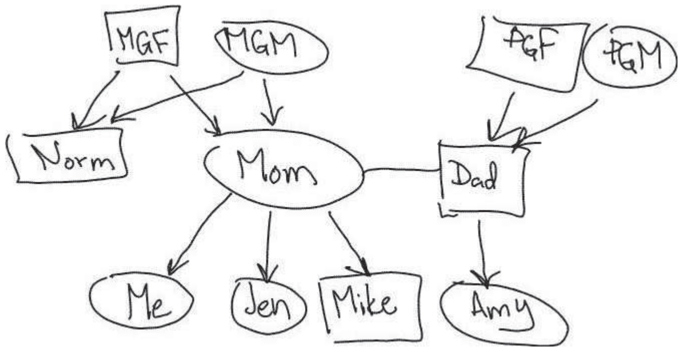


Figure C.11: Sample family tree sketch from user number 5.

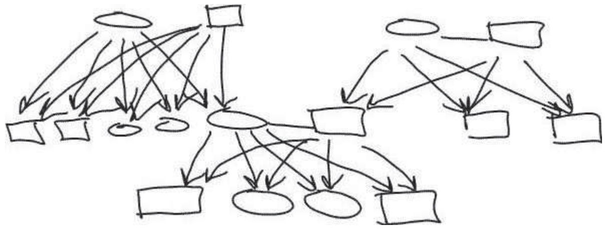


Figure C.12: Sample family tree sketch from user number 5.

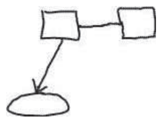


Figure C.13: Sample family tree sketch from user number 8.

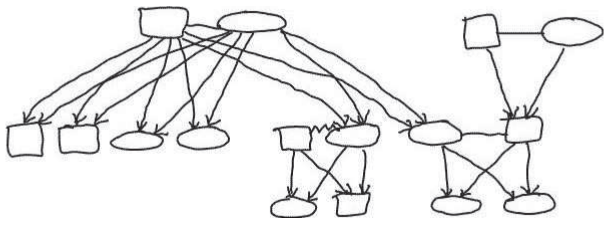


Figure C.14: Sample family tree sketch from user number 8.

Appendix D

Tabulated Results

D.1 Single-Stroke Feature Importance

Feature Importance Digital Circuits - Single-Stroke Classification		
Average Merit	Average Rank	Attribute Name
0.391 +- 0.003	1 +- 0	Part of a Closed Path
0.211 +- 0.004	2 +- 0	Inside a Closed Path
0.173 +- 0.019	3 +- 0	Distance To Left or Right Edge
0.115 +- 0.007	4.2 +- 0.4	Number of LX Intersections
0.108 +- 0.001	4.8 +- 0.4	Bounding Box Width
0.09 +- 0.009	7 +- 1.79	Time to Previous Stroke
0.086 +- 0.001	7.1 +- 0.54	Bounding Box Height
0.083 +- 0.007	8.7 +- 2.1	Arc Length
0.081 +- 0.003	9 +- 1.26	End Point to Arc Length Ratio
0.079 +- 0.002	9.4 +- 0.92	Number of XX Intersections
0.078 +- 0.003	10.3 +- 1	Time to Draw Stroke
0.073 +- 0.004	11.5 +- 0.92	Bounding Box Area
0.059 +- 0.002	13 +- 0	Sum of Thetas
0.052 +- 0.004	14.3 +- 0.46	Time to Next Stroke
0.048 +- 0.001	14.8 +- 0.6	Self Enclosing
0.043 +- 0.003	15.9 +- 0.3	Number of LL Intersections
0.038 +- 0.001	17 +- 0	Path Density
0.031 +- 0.001	18.9 +- 0.7	Minimum Pen Speed
0.029 +- 0.001	20.8 +- 1.08	Number of Self Intersections
0.03 +- 0.002	21 +- 1.67	Distance To Top or Bottom Edge
0.029 +- 0.001	21.1 +- 1.22	Sum of Abs Value of Thetas
0.028 +- 0.002	21.9 +- 1.81	(Max - Min) Pen Speed
0.027 +- 0.003	22.8 +- 2.52	Average Pen Speed
0.026 +- 0.007	22.9 +- 3.91	Sum of Squared Thetas
0.027 +- 0.001	23.5 +- 1.69	Sum of Sqrt of Thetas
0.024 +- 0.001	25.4 +- 0.92	Maximum Pen Speed
0.02 +- 0.001	26.7 +- 0.46	Number of XL Intersections

Table D.1: Ranked list of features for single-stroke classification in digital circuits.

Feature Importance Family Trees - Single-Stroke Classification		
Average Merit	Average Rank	Attribute Name
0.443 +- 0.01	1.1 +- 0.3	Inside a Closed Path
0.425 +- 0.006	1.9 +- 0.3	Part of a Closed Path
0.38 +- 0.006	3 +- 0	Self Enclosing
0.199 +- 0.018	5.1 +- 1.14	Bounding Box Area
0.196 +- 0.011	5.1 +- 1.22	End Point to Arc Length Ratio
0.191 +- 0.011	5.6 +- 0.92	Distance To Left or Right Edge
0.176 +- 0.01	6.3 +- 0.9	Bounding Box Width
0.158 +- 0.003	8.2 +- 0.6	Arc Length
0.151 +- 0.01	8.7 +- 0.46	Bounding Box Height
0.131 +- 0.006	10.8 +- 0.6	Time to Draw Stroke
0.129 +- 0.015	11.6 +- 2.01	Time to Previous Stroke
0.124 +- 0.008	12 +- 1.26	Time to Next Stroke
0.118 +- 0.006	13 +- 1.34	Number of XX Intersections
0.116 +- 0.006	13 +- 0.89	Path Density
0.104 +- 0.006	15.1 +- 0.83	Sum of Sqrt of Thetas
0.103 +- 0.004	15.7 +- 0.64	Distance To Top or Bottom Edge
0.085 +- 0.006	17.2 +- 0.87	Number of LX Intersections
0.083 +- 0.003	17.6 +- 0.49	Sum of Thetas
0.066 +- 0.005	19.6 +- 0.66	Number of XL Intersections
0.062 +- 0.004	19.7 +- 0.64	Average Pen Speed
0.049 +- 0.006	21.9 +- 1.04	Sum of Abs Value of Thetas
0.047 +- 0.006	22 +- 1.1	Maximum Pen Speed
0.045 +- 0.005	22.7 +- 0.78	(Max - Min) Pen Speed
0.041 +- 0.011	23.1 +- 1.58	Sum of Squared Thetas
0.022 +- 0.002	25.1 +- 0.3	Number of LL Intersections
0.017 +- 0.002	25.9 +- 0.3	Minimum Pen Speed
0 +- 0	27 +- 0	Number of Self Intersections

Table D.2: Ranked list of features for single-stroke classification in family trees.

Feature Importance Statics - Single-Stroke Classification		
Average Merit	Average Rank	Attribute Name
0.091 +- 0.002	1 +- 0	Inside a Closed Path
0.083 +- 0.001	2 +- 0	Number of XX Intersections
0.073 +- 0.003	3.2 +- 0.4	Bounding Box Area
0.068 +- 0.004	4 +- 0.63	Arc Length
0.066 +- 0.002	4.8 +- 0.4	Bounding Box Width
0.061 +- 0.002	6.2 +- 0.4	Time to Draw Stroke
0.055 +- 0.003	7 +- 0.63	Average Pen Speed
0.053 +- 0.001	7.8 +- 0.4	Bounding Box Height
0.044 +- 0.001	9 +- 0	Sum of Sqrt of Thetas
0.036 +- 0.001	10.7 +- 0.78	Part of a Closed Path
0.036 +- 0.002	11.3 +- 1.49	Time to Previous Stroke
0.035 +- 0.001	12.3 +- 0.9	Sum of Abs Value of Thetas
0.035 +- 0.001	12.8 +- 0.87	Number of LL Intersections
0.033 +- 0.003	14.4 +- 2.62	Maximum Pen Speed
0.032 +- 0.002	15.2 +- 1.4	(Max - Min) Pen Speed
0.031 +- 0.001	15.9 +- 1.04	End Point to Arc Length Ratio
0.031 +- 0.001	16 +- 1.34	Time to Next Stroke
0.03 +- 0.001	17.4 +- 0.8	Self Enclosing
0.026 +- 0.001	19.1 +- 0.3	Number of XL Intersections
0.024 +- 0.001	20.3 +- 0.64	Distance To Left or Right Edge
0.024 +- 0.001	20.6 +- 0.49	Sum of Squared Thetas
0.021 +- 0.001	22.1 +- 0.3	Number of LX Intersections
0.018 +- 0.001	23.3 +- 1	Distance To Top or Bottom Edge
0.015 +- 0	24.8 +- 0.6	Sum of Thetas
0.015 +- 0.001	24.8 +- 0.98	Number of Self Intersections
0.014 +- 0.002	25.4 +- 1.43	Path Density
0.013 +- 0	26.6 +- 0.66	Minimum Pen Speed

Table D.3: Ranked list of features for single-stroke classification in complete statics solutions.

Feature Importance Statics (NoEqn) - Single-Stroke Classification		
Average Merit	Average Rank	Attribute Name
0.163 +- 0.001	1 +- 0	Bounding Box Area
0.152 +- 0.01	2.5 +- 1.02	Bounding Box Height
0.149 +- 0.003	3 +- 0.63	Part of a Closed Path
0.143 +- 0.005	3.7 +- 0.64	Arc Length
0.134 +- 0.004	5 +- 0.77	Bounding Box Width
0.128 +- 0.003	6.3 +- 0.64	Number of XX Intersections
0.126 +- 0.003	6.5 +- 0.5	Time to Draw Stroke
0.111 +- 0.002	8.4 +- 0.49	Sum of Sqrt of Thetas
0.109 +- 0.003	8.7 +- 0.64	Time to Previous Stroke
0.1 +- 0.003	10.1 +- 0.3	Self Enclosing
0.094 +- 0.003	11.2 +- 0.6	Sum of Abs Value of Thetas
0.088 +- 0.002	12.2 +- 0.6	Time to Next Stroke
0.08 +- 0.003	13.7 +- 0.64	End Point to Arc Length Ratio
0.077 +- 0.014	14.4 +- 2.65	Average Pen Speed
0.075 +- 0.003	14.4 +- 0.66	(Max - Min) Pen Speed
0.071 +- 0.003	15.7 +- 0.64	Maximum Pen Speed
0.068 +- 0.004	16.2 +- 1.17	Sum of Squared Thetas
0.043 +- 0.002	18.2 +- 0.4	Number of XL Intersections
0.04 +- 0.001	18.8 +- 0.4	Sum of Thetas
0.037 +- 0.002	20 +- 0	Distance To Left or Right Edge
0.033 +- 0.001	21.4 +- 0.49	Inside a Closed Path
0.033 +- 0.001	21.6 +- 0.49	Path Density
0.027 +- 0.001	23.8 +- 0.87	Minimum Pen Speed
0.028 +- 0.001	23.9 +- 0.7	Number of Self Intersections
0.027 +- 0.001	24.3 +- 0.78	Number of LL Intersections
0.017 +- 0.001	26 +- 0	Distance To Top or Bottom Edge
0.015 +- 0.001	27 +- 0	Number of LX Intersections

Table D.4: Ranked list of features for single-stroke classification in statics solutions without equations.

D.2 Pairwise Feature Importance

Feature Importance, Pairwise - Digital Circuits - Gate		
Average Merit	Average Rank	Attribute Name
.619 +- .003	1 +- 0	Part of Same Closed Path
.393 +- .002	2.5 +- 0.5	Minimum Distance
.378 +- .042	3 +- 1.18	Closeness Small
.341 +- .023	4.1 +- 0.7	Centroid Distance
.321 +- .03	4.9 +- .94	Closeness Large
.298 +- .017	5.8 +- .75	Minimum Endpoint to Endpoint Distance
.269 +- .029	7.3 +- 1.19	Minimum Endpoint to Any point Distance
.256 +- .015	7.7 +- .78	Time Gap
.235 +- .01	8.7 +- .64	Maximum Distance
.129 +- .005	10 +- 0	X-Overlap
.111 +- .003	11 +- 0	Y-Overlap
.009 +- 0	12 +- 0	Ratio XL
.008 +- 0	13 +- 0	Ratio LL

Table D.5: Ranked list of features for pairwise classification of gates in digital circuits.

Feature Importance, Pairwise - Digital Circuits - Wire		
Average Merit	Average Rank	Attribute Name
.301 +- .005	1 +- 0	Part of Same Closed Path
.189 +- .015	2.8 +- 1.08	Closeness Small
.189 +- .012	3 +- .77	Minimum Endpoint to Any point Distance
.182 +- .004	3.3 +- .64	Minimum Distance
.156 +- .015	4.9 +- .3	Closeness Large
.108 +- .001	6 +- 0	Minimum Endpoint to Endpoint Distance
.07 +- .006	7.4 +- .49	X-Overlap
.065 +- 002	7.6 +- .49	Y-Overlap
.038 +- .001	9.1 +- .3	Centroid Distance
.034 +- .002	9.9 +- .3	Maximum Distance
.029 +- .002	11 +- 0	Time Gap
.012 +- .002	12 +- 0	Ratio LL
.006 +- 0	13 +- 0	Ratio XL

Table D.6: Ranked list of features for pairwise classification of wires in digital circuits.

Feature Importance, Pairwise - Digital Circuits - Label		
Average Merit	Average Rank	Attribute Name
.882 +- .004	1 +- 0	Minimum Distance
.647 +- .015	2.3 +- .46	Closeness Small
.612 +- .037	3.1 +- .7	Minimum Endpoint to Any point Distance
.594 +- .027	3.6 +- .66	Centroid Distance
.551 +- .009	5.1 +- .3	Y-Overlap
.528 +- .01	5.9 +- .3	Minimum Endpoint to Endpoint Distance
.511 +- .005	7 +- 0	Part of Same Closed Path
.428 +- .022	8 +- 0	Maximum Distance
.307 +- .03	9.4 +- .49	Time Gap
.28 +- .003	9.4 +- .49	Closeness Large
.077 +- .002	11 +- 0	X-Overlap
.026 +- .002	12 +- 0	Ratio LL
.022 +- .002	13 +- 0	Ratio XL

Table D.7: Ranked list of features for pairwise classification of labels in digital circuits.

Feature Importance, Pairwise - Family Trees - People		
Average Merit	Average Rank	Attribute Name
.809 +- .007	1 +- 0	Part of Same Closed Path
.64 +- .087	2.3 +- .46	Minimum Endpoint to Endpoint Distance
.582 +- .091	2.7 +- .46	Minimum Distance
.428 +- .015	4 +- 0	Closeness Small
.363 +- .006	5 +- 0	Minimum Endpoint to Any point Distance
.338 +- .005	6 +- 0	Centroid Distance
.252 +- .003	7 +- 0	Closeness Large
.201 +- .011	8 +- 0	Time Gap
.141 +- .008	9.1 +- .3	Maximum Distance
.131 +- .003	9.9 +- .3	X-Overlap
.082 +- .003	11+- 0	Y-Overlap
.067 +- .007	12 +- 0	Ratio LL
.015 +- .003	13 +- 0	Ratio XL

Table D.8: Ranked list of features for pairwise classification of people in family trees.

Feature Importance, Pairwise - Family Trees - Text		
Average Merit	Average Rank	Attribute Name
.469 +- .004	1 +- 0	Part of Same Closed Path
.322 +- .004	2.3 +- .46	Time Gap
.305 +- .01	3.2 +- .4	Closeness Small
.301 +- .002	3.8 +- .4	Closeness Large
.286 +- .003	5.2 +- .4	Minimum Distance
.28 +- .005	5.8 +- .4	Minimum Endpoint to Any point Distance
.269 +- .007	7.1 +- .3	Minimum Endpoint to Endpoint Distance
.256 +- .003	7.9 +- .3	Centroid Distance
.244 +- .002	9 +- 0	Maximum Distance
.144 +- .006	10 +- 0	Y-Overlap
.106 +- .002	11 +- 0	X-Overlap
.012 +- .001	12 +- 0	Ratio XL
.009 +- .001	13 +- 0	Ratio LL

Table D.9: Ranked list of features for pairwise classification of text in family trees.

Feature Importance, Pairwise - Family Trees - Label		
Average Merit	Average Rank	Attribute Name
.548 +- .003	1 +- 0	Part of Same Closed Path
.306 +- .004	2 +- 0	Time Gap
.275 +- .004	3.1 +- .3	Closeness Small
.271 +- .004	3.9 +- .3	Closeness Large
.247 +- .005	5.2 +- 0 .4	Minimum Distance
.242 +- .003	6.5 +- 1.02	Minimum Endpoint to Endpoint Distance
.241 +- .004	6.7 +- .64	Minimum Endpoint to Any point Distance
.229 +- .007	8 +- .63	Centroid Distance
.214 +- .016	8.6 +- .92	Maximum Distance
.093 +- .002	10 +- 0	Y-Overlap
.075 +- .002	11 +- 0	X-Overlap
.009 +- 0	12 +- 0	Ratio XL
.006 +- 0	13 +- 0	Ratio LL

Table D.10: Ranked list of features for pairwise classification of labels in family trees.

Feature Importance, Pairwise - Statics - Body		
Average Merit	Average Rank	Attribute Name
.395 +- .021	1.1 +- .3	Closeness Large
.367 +- .021	2.3 +- .64	Minimum Distance
.355 +- .004	2.6 +- .49	Closeness Small
.298 +- .016	4.5 +- .5	Minimum Endpoint to Endpoint Distance
.297 +- .007	4.5 +- .5	Minimum Endpoint to Any point Distance
.214 +- .014	6 +- 0	X-Overlap
.151 +- .015	7.5 +- 1.02	Y-Overlap
.141 +- .008	8.2 +- .75	Centroid Distance
.136 +- .005	8.4 +- .49	Time Gap
.124 +- .003	9.9 +- .3	Maximum Distance
.103 +- .002	11 +- 0	Part of Same Closed Path
.086 +- .004	12 +- 0	Ratio LL
.047 +- .002	13 +- 0	Ratio XL

Table D.11: Ranked list of features for pairwise classification of bodies in complete statics solutions.

Feature Importance, Pairwise - Statics - Arrow		
Average Merit	Average Rank	Attribute Name
.327 +- .003	1 +- 0	Part of Same Closed Path
.254 +- .016	2.3 +- .46	Time Gap
.233 +- .011	2.8 +- .6	Minimum Distance
.217 +- .012	4.4 +- .49	Minimum Endpoint to Any point Distance
.215 +- .002	4.5 +- .67	Minimum Endpoint to Endpoint Distance
.185 +- .004	6 +- 0	Closeness Small
.173 +- .003	7.3 +- .46	Closeness Large
.161 +- .019	7.7 +- .46	Centroid Distance
.101 +- .003	9.5 +- .5	X-Overlap
.101 +- .005	9.6 +- .66	Maximum Distance
.091 +- .003	10.9 +- .3	Y-Overlap
.025 +- .001	12 +- 0	Ratio LL
.013 +- .001	13 +- 0	Ratio XL

Table D.12: Ranked list of features for pairwise classification of arrows in complete statics solutions.

Feature Importance, Pairwise - Statics - Text		
Average Merit	Average Rank	Attribute Name
.479 +- .004	1 +- 0	Part of Same Closed Path
.049 +- .004	2.4 +- .66	Closeness Large
.048 +- .003	3 +- .45	Closeness Small
.041 +- 0	4.6 +- .49	Minimum Distance
.044 +- .006	4.9 +- 2.07	Time Gap
.04 +- 0	5.6 +- .49	Centroid Distance
.038 +- .001	6.6 +- .49	Minimum Endpoint to Any point Distance
.036 +- 0	7.9 +- .3	Minimum Endpoint to Endpoint Distance
.034 +- .001	9 +- 0	Maximum Distance
.027 +- .003	10 +- 0	Y-Overlap
.015 +- .001	11 +- 0	X-Overlap
.002 +- 0	12 +- 0	Ratio LL
.001 +- 0	13 +- 0	Ratio XL

Table D.13: Ranked list of features for pairwise classification of text in complete statics solutions.

Feature Importance, Pairwise - Statics - Other		
Average Merit	Average Rank	Attribute Name
.249 +- .018	1 +- 0	Part of Same Closed Path
.179 +- .014	2 +- 0	Time Gap
.146 +- .012	3 +- 0	Ratio LL
.117 +- .003	4.4 +- .66	Closeness Large
.116 +- .007	5.1 +- .94	Closeness Small
.107 +- .005	6.5 +- 1.02	Minimum Endpoint to Endpoint Distance
.107 +- .005	6.8 +- 1.25	Centroid Distance
.104 +- .006	7.5 +- 1.12	Minimum Endpoint to Any point Distance
.1 +- .003	9 +- .45	Minimum Distance
.095 +- .005	9.7 +- .64	Maximum Distance
.08 +- .002	11 +- 0	Y-Overlap
.73 +- .001	12 +- 0	Ratio XL
.061 +- .003	13 +- 0	X-Overlap

Table D.14: Ranked list of features for pairwise classification of other in complete statics solutions.

Feature Importance, Pairwise - Statics (NoEqn) - Body		
Average Merit	Average Rank	Attribute Name
.412 +- .012	1.4 +- .49	Minimum Distance
.404 +- .033	1.6 +- .49	Closeness Large
.356 +- .004	3 +- 0	Closeness Small
.305 +- .007	4.4 +- .49	Minimum Endpoint to Endpoint Distance
.3 +- .016	4.6 +- .49	Minimum Endpoint to Any point Distance
.24 +- .004	6 +- 0	X-Overlap
.179 +- .005	7 +- 0	Centroid Distance
.144 +- .007	8.4 +- .49	Maximum Distance
.144 +- .011	9.1 +- .83	Y-Overlap
.138 +- .002	9.5 +- .67	Time Gap
.1 +- .003	11 +- 0	Part of Same Closed Path
.087 +- .002	12 +- 0	Ratio LL
.05 +- .002	13 +- 0	Ratio XL

Table D.15: Ranked list of features for pairwise classification of bodies in statics solutions without equations.

Feature Importance, Pairwise - Statics (NoEqn) - Arrow		
Average Merit	Average Rank	Attribute Name
.291 +- .006	1 +- 0	Part of Same Closed Path
.224 +- .017	2.1 +- .3	Time Gap
.181 +- .004	3.7 +- 1	Minimum Distance
.176 +- .001	4.3 +- .64	Closeness Small
.172 +- .006	4.9 +- 1.04	Closeness Large
.169 +- .014	5.8 +- 1.33	Minimum Endpoint to Any point Distance
.168 +- .016	6.2 +- 1.4	Minimum Endpoint to Endpoint Distance
.12 +- .009	8 +- 0	Centroid Distance
.094 +- .005	9.1 +- .3	X-Overlap
0.085 +- .005	10.2 +- .6	Y-Overlap
.079 +- .004	10.7 +- .46	Maximum Distance
.03 +- .003	12 +- 0	Ratio LL
.015 +- .001	13 +- 0	Ratio XL

Table D.16: Ranked list of features for pairwise classification of arrows in statics solutions without equations.

Feature Importance, Pairwise - Statics (NoEqn) - Label		
Average Merit	Average Rank	Attribute Name
.417 +- .002	1 +- 0	Part of Same Closed Path
.302 +- .013	2.3 +- .46	Closeness Small
.285 +- .017	2.8 +- .6	Minimum Distance
.266 +- .009	4.3 +- .64	Minimum Endpoint to Any point Distance
.259 +- .006	5.7 +- 1	Closeness Large
.256 +- .006	6 +- 1.1	Minimum Endpoint to Endpoint Distance
.256 +- .009	6.3 +- 1.42	Centroid Distance
.25 +- .003	7.9 +- .7	X-Overlap
.246 +- .004	8.7 +- .64	Time Gap
.218 +- .002	10 +- 0	Maximum Distance
.126 +- .005	11 +- 0	Y-Overlap
.016 +- .001	12 +- 0	Ratio LL
.004 +- .001	13 +- 0	Ratio XL

Table D.17: Ranked list of features for pairwise classification of labels in statics solutions without equations.

Feature Importance, Pairwise - Statics (NoEqn) - Other		
Average Merit	Average Rank	Attribute Name
.271 +- .015	1 +- 0	Part of Same Closed Path
.179 +- .014	2 +- 0	Time Gap
.153 +- .006	3 +- 0	Ratio LL
.121 +- .003	4,2 +- .4	Closeness Large
.111 +- .008	4.9 +- .54	Closeness Small
.096 +- .004	6.3 +- 1	Minimum Endpoint to Endpoint Distance
.094 +- .002	7.6 +- .66	Minimum Endpoint to Any point Distance
.093 +- .003	8.1 +- 1.14	Minimum Distance
.091 +- .004	8.5 +- 1.28	Centroid Distance
.087 +- .004	9.4 +- .92	Maximum Distance
.082 +- .002	11 +- 0	Y-Overlap
.069 +- .001	12 +- 0	Ratio XL
.058 +- .007	13 +- 0	X-Overlap

Table D.18: Ranked list of features for pairwise classification of other in statics solutions without equations.

Bibliography

[1] Christine Alvarado and Randall Davis. Dynamically constructed Bayes nets for multi-domain sketch understanding. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, page 33, New York, NY, USA, 2007. ACM.

[2] Christine Alvarado and Michael Lazzareschi. Properties of real-world digital logic diagrams. In *PLT '07: Proceedings of the First International Workshop on Pen-Based Learning Technologies*, page 12, Washington, DC, USA, 2007. IEEE Computer Society.

[3] Akshay Bhat and Tracy Hammond. Using entropy to distinguish shape versus text in hand-drawn diagrams. In *IJCAI'09: Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 1395–1400, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[4] David Bischel, Thomas Stahovich, Eric Peterson, Randall Davis, and Aaron Adler. Combining speech and sketch to interpret unconstrained descriptions of mechanical devices. In *IJCAI'09: Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 1401–1406, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[5] Christopher M. Bishop, Markus Svensen, and Geoffrey E. Hinton. Distinguishing text from graphics in on-line handwritten ink. In *IWFHR '04: Proceedings of the Ninth International Workshop on Frontiers in Handwriting Recognition*, pages 142–147, Washington, DC, USA, 2004. IEEE Computer Society.

[6] Chris Calhoun, Thomas F. Stahovich, Tolga Kurtoglu, and Levent Burak Kara. Recognizing multi-stroke symbols. In *AAAI Spring Symposium on Sketch Understanding*, pages 15–23. AAAI Press, 2002.

[7] Philip J. Cowans and Martin Szummer. A graphical model for simultaneous partitioning and labeling. In *Proc. AI & Statistics*, 2005.

[8] Guihuan Feng, Christian Viard-Gaudin, and Zhengxing Sun. On-line hand-drawn electric circuit diagram recognition using 2d dynamic programming. *Pattern Recognition*, 42(12):3215–3223, 2009.

[9] Martin Field, Sam Gordon, Eric Peterson, Raquel Robinson, Thomas Stahovich, and Christine Alvarado. The effect of task on classification accuracy: using gesture recognition techniques in free-sketch recognition. In *SBIM '09: Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, pages 109–116, New York, NY, USA, 2009. ACM.

[10] Leslie Gennari, Levent Burak Kara, Thomas F. Stahovich, and Kenji Shimada. Combining geometry and domain knowledge to interpret hand-drawn diagrams. *Computers & Graphics*, 29(4):547–562, 2005.

[11] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

[12] Tracy Hammond and Randall Davis. Tahuti: A geometrical sketch recognition system for UML class diagrams. In *AAAI Spring Symposium on Sketch Understanding*, pages 59–66, 2002.

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

[14] Achim G. Hoffmann, Rex Bing Hung Kwok, and Paul Compton. Using subclasses to improve classification learning. In *EMCL '01: Proceedings of the 12th European Conference on Machine Learning*, pages 203–213, London, UK, 2001. Springer-Verlag.

[15] Heloise Hse and A. Richard Newton. Sketched symbol recognition using Zernike moments. In *ICPR '04: Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 1*, pages 367–370, Washington, DC, USA, 2004. IEEE Computer Society.

[16] Heloise Hwawen Hse and A. Richard Newton. Recognition and beautification of multi-stroke symbols in digital ink. *Computers & Graphics*, 29(4):533–546, 2005.

[17] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[18] Joseph J. LaViola Jr. An initial evaluation of Mathpad^2: A tool for creating dynamic mathematical illustrations. *Computers & Graphics*, 31(4):540–553, 2007.

[19] Joseph J. LaViola Jr. and Robert C. Zeleznik. Mathpad^2: a system for the creation and exploration of mathematical sketches. *ACM Trans. Graph.*, 23(3):432–440, 2004.

[20] Levent Burak Kara, Leslie Gennari, and Thomas F Stahovich. A sketch-based interface for the design and analysis of simple vibratory mechanical systems. In *ASME International Design Engineering Technical Conferences*, 2004.

[21] Levent Burak Kara and Thomas F. Stahovich. Hierarchical parsing and recognition of hand-sketched diagrams. In *UIST*, pages 13–22, 2004.

[22] Levent Burak Kara and Thomas F. Stahovich. An image-based trainable symbol recognizer for sketch-based interfaces. In *AAAI Fall Symposium Series 2004: Making Pen-Based Interaction Intelligent and Natural*, pages 99–105. AAAI Press, 2004.

[23] Levent Burak Kara and Thomas F. Stahovich. An image-based, trainable symbol recognizer for hand-drawn sketches. *Computers & Graphics*, 29(4):501–517, 2005.

[24] J. J. LaViola. *Mathematical sketching: A new approach to creating and exploring dynamic illustrations.* PhD thesis, Brown University, 2005.

[25] WeeSan Lee, Levent Burak Kara, and Thomas F. Stahovich. An efficient graph-based recognizer for hand-drawn symbols. *Computers & Graphics*, 31(4):554–567, 2007.

[26] Yun Luo. Can subclasses help a multiclass learning problem? In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 214 –219, 4-6 2008.

[27] J. L. Meriam and L. G. Kraige. *Engineering Mechanics: Statics.* Wiley, 6th edition, 2006.

[28] Mike Oltmans, Christine Alvarado, and Randall Davis. ETCHA sketches: Lessons learned from collecting sketch data. In *In Making Pen-Based Interaction Intelligent and Natural. AAAI Fall Symposium*, pages 134–140. Press, 2004.

[29] Tom Y. Ouyang and Randall Davis. A visual approach to sketched symbol recognition. In *IJCAI'09: Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 1463–1468, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[30] Rachel Patel, Beryl Plimmer, John Grundy, and Ross Ihaka. Ink features for diagram recognition. In *SBIM '07: Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling*, pages 131–138, New York, NY, USA, 2007. ACM.

[31] Brandon Paulson and Tracy Hammond. Paleosketch: accurate primitive sketch recognition and beautification. In *IUI '08: Proceedings of the 13th international conference on Intelligent user interfaces*, pages 1–10, New York, NY, USA, 2008. ACM.

[32] Brandon Paulson, Pankaj Rajan, Pedro Davalos, Ricardo Gutierrez-Osuna, and Tracy Hammond. What!?! No Rubine features?: Using geometric-based features to produce normalized confidence values for sketch recognition. In *Visual Languages and Human Centric Computing Workshop: Sketch Tools for Diagramming*, 2008.

[33] Yuan Qi, Martin Szummer, and Thomas P. Minka. Diagram structure recognition by Bayesian conditional random fields. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2*, pages 191–196, Washington, DC, USA, 2005. IEEE Computer Society.

[34] Shengfeng Qin. Intelligent classification of sketch strokes. In *Computer as a Tool, 2005. EUROCON 2005.The International Conference on*, volume 2, pages 1374 –1377, 21-24 2005.

[35] Dean Rubine. Specifying gestures by example. *Computers & Graphics*, 25(4):329–337, 1991.

[36] Eric Saund and Edward Lank. Stylus input and editing without prior selection of mode. In *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 213–216, New York, NY, USA, 2003. ACM.

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

[38] Tevfik Metin Sezgin and Randall Davis. HMM-based efficient sketch recognition. In *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*, pages 281–283, New York, NY, USA, 2005. ACM.

[39] Michael Shilman, Hanna Pasula, Stuart Russell, and Richard Newton. Statistical visual language models for ink parsing. In *In AAAI Spring Symposium on Sketch Understanding*, pages 126–132. AAAI Press, 2002.

[40] Michael Shilman and Paul Viola. Spatial recognition and grouping of text and graphics. In *SBIM '04: Proceedings of the 1st Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 91–95, 2004.

[41] Michael Shilman, Zile Wei, Sashi Raghupathy, Patrice Simard, and David Jones. Discerning structure from freeform handwritten notes. In *ICDAR '03: Proceedings of the Seventh International Conference on Document Analysis and Recognition*, page 60, Washington, DC, USA, 2003. IEEE Computer Society.

[42] Thomas F. Stahovich. Segmentation of pen strokes using pen speed. In *AAAI Fall Symposium Series 2004: Making Pen-Based Interaction Intelligent and Natural*, 2004.

[43] Martin Szummer and Yuan Qi. Contextual recognition of hand-drawn diagrams with conditional random fields. In *IWFHR '04: Proceedings of the Ninth International Workshop on Frontiers in Handwriting Recognition*, pages 32–37, Washington, DC, USA, 2004. IEEE Computer Society.

[44] Xin Wang, Manoj Biswas, and Sashi Raghupathy. Addressing class distribution issues of the drawing vs writing classification in an ink stroke sequence. In *SBIM '07: Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling*, pages 139–146, New York, NY, USA, 2007. ACM.

[45] Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *UIST '07: Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 159–168, New York, NY, USA, 2007. ACM.

[46] A. Wolin, B. Paulson, and T. Hammond. Sort, merge, repeat: an algorithm for effectively finding corners in hand-sketched strokes. In *SBIM '09: Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, pages 93–99, New York, NY, USA, 2009. ACM.

[47] Yiyan Xiong and Joseph J. LaViola, Jr. Revisiting shortstraw: improving corner finding in sketch-based interfaces. In *SBIM '09: Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, pages 101–108, New York, NY, USA, 2009. ACM.

[48] Bo Yu and Shijie Cai. A domain-independent system for sketch recognition. In *GRAPHITE '03: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 141–146, New York, NY, USA, 2003. ACM.