SKETCH RECOGNITION OF DIGITAL INK DIAGRAMS

A thesis presented in partial fulfilment of the requirements for the degree of
Doctor of Philosophy
in
Computer Science
at Massey University, Palmerston North,
New Zealand.

Amirhossein Ghodrati

2020

Contents

List of Tables

List of Figures

Abstract

Sketch recognition of digital ink diagrams is the process of automatically identifying hand-drawn elements in a diagram. This research focuses on the simultaneous grouping and recognition of shapes in digital ink diagrams. In order to recognise a shape, we need to group strokes belonging to a shape, however, strokes cannot be grouped until the shape is identified. Therefore, we treat grouping and recognition as a simultaneous task.

Our grouping technique uses spatial proximity to hypothesise shape candidates. Many of the hypothesised shape candidates are invalid, therefore we need a way to reject them. We present a novel rejection technique based on novelty detection. The rejection method uses proximity measures to validate a shape candidate. In addition, we investigate on improving the accuracy of the current shape recogniser by adding extra features. We also present a novel connector recognition system that localises connector heads around recognised shapes.

We perform a full comparative study on two datasets. The results show that our approach is significantly more accurate in finding shapes and faster on process diagram compared to Stahovich et al. (2014), which the results show the superiority of our approach in terms of computation time and accuracy. Furthermore, we evaluate our system on two public datasets and compare our results with other approaches reported in the literature that have used these dataset. The results show that our approach is more accurate in finding and recognising the shapes in the FC dataset (by finding and recognising 91.7% of the shapes) compared to the reported results in the literature.

Acknowledgements

I would like to express my appreciation to my supervisors Dr. Rachel Blagojevic, Prof. Hans Guesgen and Prof. Stephen Marsland for their constant guidance throughout my PhD. I wish to acknowledge the support of my family and friends. I'm grateful for the support of my parents and without their support I would not be where I am.

Chapter 1

Introduction

1.1 Motivation

People often sketch when trying to solve a problem or express an idea. Sketching allows people to visually represent their ideas (Johnson et al. 2009). Sketching plays a key role in conceptual designs, enabling one to naturally express ideas and concepts (Delaye 2014). Hand-drawn diagrams, in particular, are frequently used for externalising ideas and documenting existing phenomena. Pen and paper offers an unconstrained space for drawing such diagrams, it is quick to use, and allows for ambiguity (Buxton 2007).

Often we seek to digitize diagrams for ease of replication, storage, and sharing. However, using a mouse and keyboard to produce the digital diagrams is a cumbersome and tedious task. Sketching on computers has become easier and more natural with the use of a stylus (Sezgin et al. 2007). The availability of stylus input to the computer can offer similar advantages to pen and paper alongside standard computational support. The automatic recognition of these diagrams can allow for even greater advantages, such as beautification, intelligent editing, automated translation to alternative forms, and execution or animation of sketch models. For example, Figure 1.1 shows an example of intelligent editing for a sketched graph diagram in SketchNode (Plimmer et al. 2010) where the edges connected to a node are moved when a node is moved. Figure 1.2 shows an example of a graph diagram that is beautified after recognition of the sketch, and Figure 1.3 shows an example of auto-translation, where the web page of a sketched user interface is automatically generated.

Sketched diagram recognition is the process of automatically identifying hand-drawn elements of a diagram, not recognising the sketch itself. The automatic recognition requires recognising the individual elements in the sketch. For example in Figure 1.4, a sketched diagram recognition system could identify the three drawn shapes, i.e., square, arrow and diamond.

When a sketch is drawn with a stylus on a Tablet PC, each stroke (from pen-down

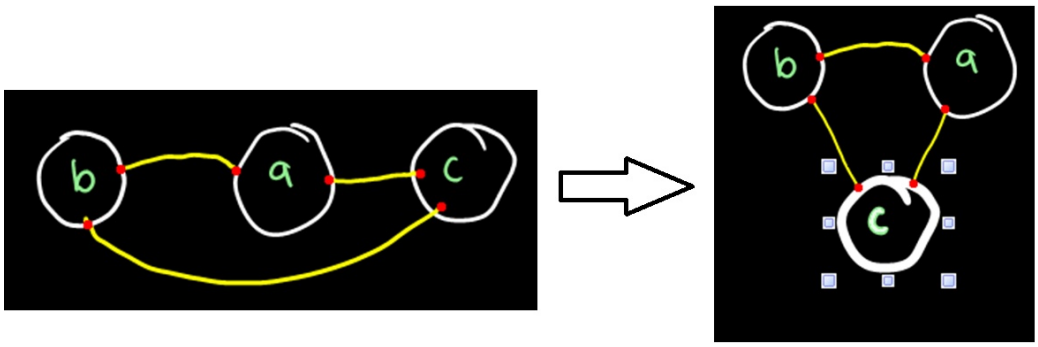


Figure 1.1: An example of intelligent editing in SketchNode (Plimmer et al. 2010)

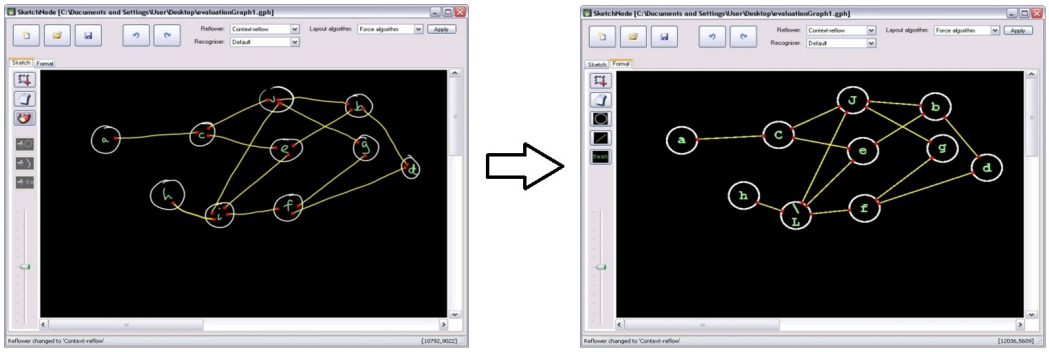


Figure 1.2: An example of beautification in SketchNode (Plimmer et al. 2010)

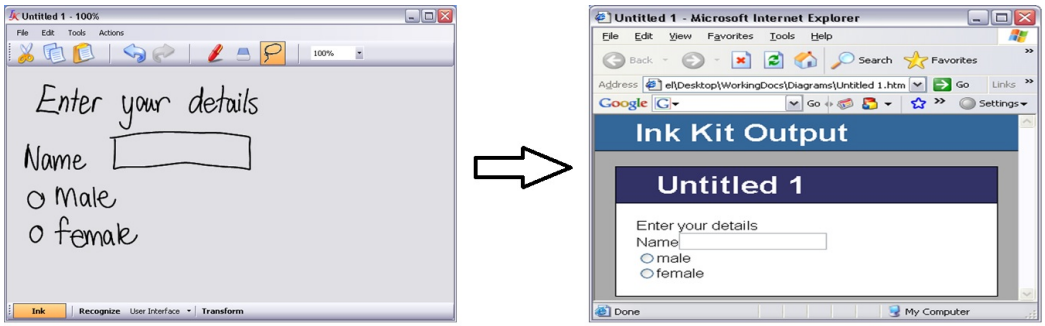


Figure 1.3: An example of auto translation

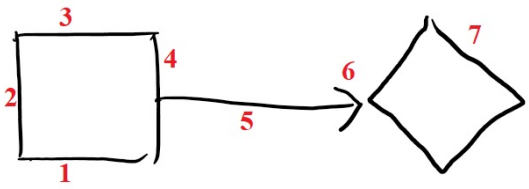


Figure 1.4: An example of a drawn sketch with the numbers representing the order of drawn strokes

to pen-up) is recorded as a series of (x, y) coordinates. Each stroke is sampled at high frequency with this point-level information, as well as pen pressure (if the screen is pressure sensitive), and time stamps. A sketch recognition system that deals with the drawn strokes on a Tablet PC is often referred to as "online". On the other hand, an "offline" sketch recognition system deals with the scanned image of a sketch. The *offline* systems require some pre-processing steps to extract information about the location of drawn strokes/primitives (Bresler et al. 2016*c*, Ray et al. 2019), while the *online* systems have the advantage of providing information about the location of strokes and their order of drawings.

Sketched diagram recognition systems can be categorised in three principal approaches: bottom-up, top-down and a combination of both (Patel et al. 2007). In the bottom-up approach, the system begins the recognition at the stroke level, followed by a progressive grouping of strokes into more complex shapes. The top-down approach first analyses the sketch's structure and then uses this information to aid recognition of the composite parts. The hybrid approach combines the recognition of each individual stroke with the layout information for recognition.

Bottom-up approaches are more widely-used (Deufemia et al. 2014). A bottom-up recognition engine typically takes primitives (lines and curves), or strokes as input, separates the writing from the drawing, groups the shape primitives together to form distinct diagram elements, recognises shapes, and finally infers the semantics of the diagram.

In this thesis we describe a bottom-up system that groups and recognises the shapes in a sketched diagram. This requires a "grouper" to determine which strokes together form a distinct shape, and a "recogniser" to identify what the shape is. For example, in Figure 1.4 a grouper should determine that strokes 1,2,3 and 4 together form a single shape, and the recogniser should determine it is a square.

1.2 Aims

We believe an ideal online sketch recognition system should have the following characteristics:

- **Accurate:** Accurately find and recognise all the drawn shapes in a diagram.
- **Computationally inexpensive:** Find and recognise all the drawn shapes fast.
- **Domain independent:** The recognition should not be limited to a specific diagram domain. This means that the underlying system should be general enough to be trained for a variety of diagram domains. We expect labelled examples to

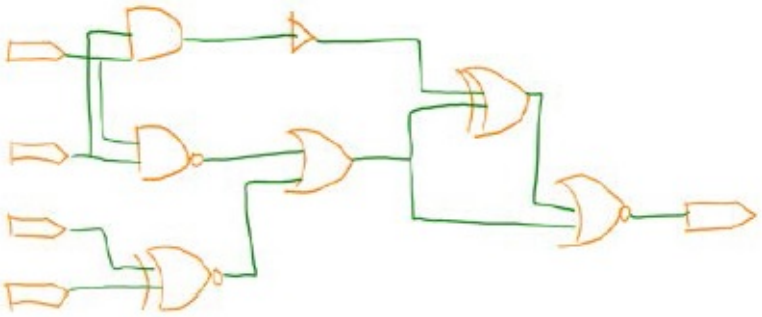


Figure 1.5: An example of a digital circuit diagram in (Stevens et al. 2013)

be provided for the system to learn from. Such information does not make the system domain dependent.

- **Supportive of a free sketch environment:** This should allow the users to draw the diagrams as freely as they do on pen and paper. The system should not put any restriction on how users draw the diagrams, such as restricting the user to draw each shape with a single stroke.

Reviewing the literature shows that a sketch recognition system with these characteristics is missing. In this work we seek to move the field of sketch recognition closer to the above ideals, particularly for grouping and shape recognition.

1.3 Background

Accurately grouping strokes into potential shapes is not a trivial problem. For a systematically drawn diagram, using spatial and/or temporal heuristics may be sufficient to identify the individual groups, since objects are drawn sequentially and in clearly delineated positions. However, this is often not the case, and there may not be clear spatial or temporal boundaries between elements. One reason for this difficulty is that users often have an interspersed drawing style, where they start drawing a new shape before completing the previous one (Sezgin & Davis 2007*b*). Another reason is that diagrams frequently contain highly connected components, e.g., the circuit diagram in Figure 1.5, where shapes are in very close proximity to one another or even overlapping. This issue is more evident in connected diagrams, in which connectors (e.g. arrows) are in close proximity to the drawn shapes. The naïve approach to grouping, of examining all possible combinations of strokes in the diagram, would require exponential time for the number of strokes.

Some methods avoid the problem of grouping completely by placing constraints on how objects should be drawn. For example, users have been restricted to drawing

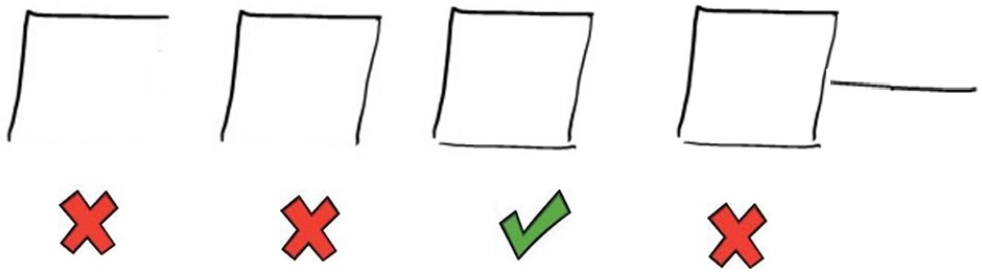


Figure 1.6: An example of some invalid shape candidates and a valid one

each shape with a single stroke (Rubine 1991, Wobbrock et al. 2007, Reaver et al. 2011*a*, Herold & Stahovich 2012, Plimmer et al. 2012), or asked to provide cues such as clicking a button, or pausing for a period of time, in order to show that the current shape is finished (Hse & Newton 2005). Other systems require a temporally contiguous sequence of strokes to be drawn for a shape (Gennari et al. 2005). Although such constraints simplify the grouping process, they do little to preserve a flexible, free-sketch environment that follows on from the user experience of sketching with pen and paper.

There is a potential chicken-and-egg problem with grouping and shape recognition (Arandjelović & Sezgin 2011, Peterson et al. 2010). Identifying a group of strokes that form a shape requires recognition of the shape, while recognising a shape may require that the correct group of strokes are presented to the recogniser. Therefore, in this PhD we treat them as simultaneous tasks. The simultaneous grouping and recognition involves a continuous interaction between grouper and recogniser. The grouper we propose uses spatial information to hypothesise shape candidates and the recogniser rejects a candidate if it is not a valid shape otherwise accepts the candidate and determines what the shape is. For example, Figure 1.6, shows an example of some hypothesised shape candidates where only one is valid.

Providing domain knowledge to the system can aid the recognition process, but at the cost of making the system domain dependent. Domain knowledge can be provided in various forms. A labelled dataset for training a sketch recognition system can be considered domain knowledge. However, a system that only takes a labelled dataset as domain knowledge can be flexible enough to be applied to different domains, without the need to make domain-specific changes to the underlying system. In Chapter 2, we review some systems that use specific domain information for recognition, and so are not applicable to all domains. For example, some methods use hard-coded shape description and shape relations (Alvarado & Davis 2006), while some perform structural analysis for a specific domain, such as flowcharts (Bresler et al. 2013*a*, 2016*a*) or chemical drawings (Ouyang & Davis 2007, Kang et al. 2014) to match the recognition with the

specifications of that domain. Such systems need to be redesigned for a new domain, and are therefore not domain independent.

Sketched diagrams are usually comprised of three semantically distinct set of strokes: text, shapes and connectors. The separation, grouping and recognition of text is an area of research that has largely been investigated (Bishop et al. 2004, Bhat & Hammond 2009, Blagojevic et al. 2011, Qi et al. 2005, Delaye & Liu 2012, Van Phan & Nakagawa 2016). In our sketch recognition pipeline, the text separation process is carried out before the shape recognition process and therefore is out of the scope of this PhD. We assume the text strokes are accurately excluded using a divider such as that designed by Blagojevic et al. (2011) and sent to a separate text recognition system. However, in Chapter 7, we show that using such dividers would affect the results as the text separation is not a completely solved problem.

The connectors are inherently different from the regular shapes. Their appearance can vary markedly between examples, and they have requirements that are not true for regular shapes, such as signifying relationships between shapes in a diagram. Therefore, we treat them separately.

The connectors are usually in close spatial proximity of shapes, which makes it difficult to use the spatial information to separate them from the shapes. For example, Figure 1.5 shows an example of a connected diagram where shapes and connectors are in a very close spatial proximity of each other. For non-connected diagrams, the process of grouping and recognition can be simpler, as the shapes are already spatially far from each other. For example, Figure 1.7 shows an example of a drawn user interface where shapes are relatively far from each other. In this PhD, we have focused on connected diagrams since they are more challenging, and unconnected ones are just a subset of them.

1.4 Thesis Objectives

The goal of this thesis is to build a domain independent sketch recognition system that supports a free sketching environment. We aim for the system to work quickly (without having the user to wait long to see the results) with high accuracy rates. Given the findings from our review of the existing methods of sketch recognition systems (see Chapter 2), we believe that a simultaneous grouping and recognition system is a promising way forward. To achieve this we have three sub-objectives:

- Design and implement a grouping algorithm that uses heuristics to hypothesise shape candidates and simultaneously works with a recogniser.
- Design and implement a rejection system for invalid shape candidates, to work with the shape recogniser.

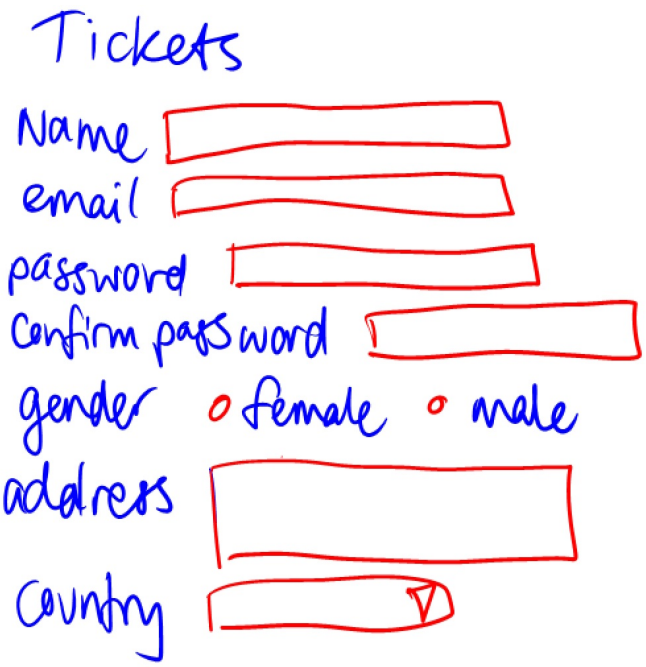


Figure 1.7: An example of a user interface

- Design and implement a connector recognition system.

In our sketch recognition system, we first build a simultaneous grouping and recognition system to find and recognise the shapes, followed by a connector recognition process. The contributions of this work are:

- A deterministic grouping algorithm that uses heuristics to hypothesise shape candidates. The grouping algorithm works simultaneously with the recogniser.
- A shape rejection system based on novelty detection techniques (Marsland 2003, Pimentel et al. 2014). In chapter 2, we review the different techniques used for rejecting invalid shape candidates. This area has received little attention thus far, particularly the use of novelty detection for rejection.
- A connector recognition system. The existing connector recognition systems either put a lot of restrictions on how connectors should be drawn or are designed specifically for a particular type of connector (e.g. designed for arrows). Our connector recognition system does not have such limitations.

1.5 Thesis Outline

The remainder of this thesis is as the following:

Chapter 2 contains a review of existing sketch recognition systems, first with a focus on grouping, followed by a review of existing shape recognisers. Next, methods used for rejecting invalid shape candidates are summarised. Finally, existing connector recognition systems are reviewed.

Chapter 3 describes the our system design, in which each component of the sketch recognition system is described in more detail. Additionally, we describe the datasets we used during the development of our sketch recognition system.

Chapter 4 provides details of the grouping algorithm. The performance of the grouper is then evaluated in terms of accuracy and time on various datasets.

Chapter 5 describes the details of the recogniser we chose as the shape recogniser. The experiments used to evaluate the performance of the recogniser on various datastes are presented. This is followed by some experiments with modifications to the feature representation for improving the recogniser's accuracy. Next, we describe our proximity-based rejection method with the different metrics used for measuring the proximity. We also provide two solutions for dealing with incomplete shapes that are not rejected. This is followed by a description of a pre-processing outlier detection method. Finally, we perform various experiments to measure the accuracy of each proposed method.

Chapter 6 provides details of our approach to connector localisation and grouping. We first describe how we learn about the common part of the connectors, followed by a description of connector grouping algorithm.

Chapter 7 provides the results of the final evaluation of our sketch recognition system on different datasets. We first perform a full comparative study, comparing our results to that of an existing method from the literature. This is followed by comparing our results with other approaches on two publicly available datasets. Finally, we provide some discussion on the performance of the system.

We conclude in Chapter 8 and outline possible future work.

Chapter 2

Literature Review

Sketch recognition systems are designed to automatically interpret hand-drawn diagrams on a computer. Performing recognition often requires a pipeline consisting of some or all of the following steps: separating the sketch into writing and drawing strokes; grouping strokes and recognising shapes and identifying domain-specific components.

The separation of text from the rest of the diagram is out of the scope of this PhD. Grouping and recognition of the shapes usually requires two components: grouper and recogniser. The grouper decides which strokes belong to the same shape and the recogniser identifies the label for the group of strokes. In some sketch recognition systems, the connectors are treated differently as they are inherently different from the regular shapes.

This chapter presents a review of different components of a sketch recognition system for digital ink diagrams. It begins by reviewing the sketch recognition systems with a focus on the role of grouping. This is followed by existing shape recognisers, followed with methods used for rejecting invalid shape candidates. Finally, in Section 2.5 we will review existing connector recognition systems.

2.1 Evaluation Metrics

Different metrics for evaluating a sketch recognition system are used in the literature. The most commonly used metric is the total shape accuracy, which is calculated by dividing the number of correctly grouped and recognised shapes over the total number of shapes in a dataset. Some approaches report the accuracy as the average of accuracies of diagrams in a dataset (Stahovich et al. 2014). Some report stroke level accuracies, which shows the portion of correctly recognised strokes regardless of their group (Bresler et al. 2016*b*). In this chapter, we report the accuracies as the total shape accuracy, as is the most commonly used method, unless otherwise is stated.

Without a full comparative study of each technique, it is difficult to judge the relative

accuracy of different approaches, particularly when different datasets are used; here we comment on them based on the reported results - which in many cases, indicates their performance in the best case scenario.

2.2 The Role of Grouping in Sketch Recognition

There is a potential chicken-and-egg problem with grouping and shape recognition (Arandjelović & Sezgin 2011, Peterson et al. 2010). Identifying a group of primitives that form a shape requires recognition of the shape, while recognising a shape may require that the correct group of primitives are presented to the recogniser. Therefore, one of the two main approaches identified in the literature is to perform grouping and recognition simultaneously. Techniques for simultaneous grouping and recognition are described in Section 2.2.1. An alternative approach is to follow a sequence and perform the grouping independently of the shape recognition. This is typically based on either a clustering method that groups primitives together, or a sequential optimisation that optimise a cost function; these methods are reviewed in Section 2.2.2.

2.2.1 Simultaneous Grouping and Recognition

Many groupers use a shape recogniser as a way of guiding the search process to only group viable shape candidates. We have identified three conceptual approaches from the literature, based on how they use the information from the recogniser. The first of these are negative example methods that use the recogniser to reject potential shapes that are not recognised. The second approach is based on grammars and languages. These methods work by defining a logical grammar to describe acceptable shapes. The third technique is based on optimisation, where usually the temporal order of shapes are captured for grouping primitives. We have also included a section on other simultaneous methods that do not fall into the previous categories. The general process of simultaneous grouping and recognition is shown in Figure 2.1. Domain knowledge (displayed in dashed box) is mainly used in the grammar and language based approaches.

Negative Examples

The most common method of enabling a classifier to reject shape candidates is to include the invalid shapes that should be ignored in the training set, as part of a ‘no-match’ class. Several sub-classes of shapes can be part of this class. The simultaneous grouping and recognition process then becomes a method for selecting candidate shapes using a classifier that returns either the class representing a shape, or ‘no-match’. The risk of this approach is that it assumes that the ‘no-match’ class is complete.

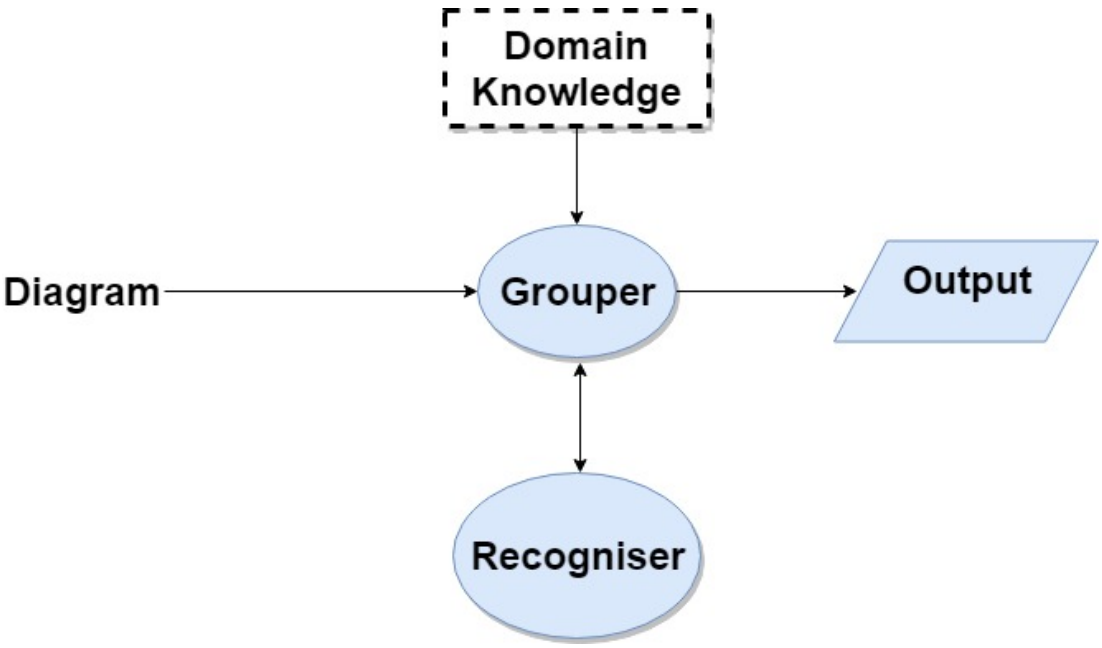


Figure 2.1: Simultaneous grouping and recognition process

Ouyang & Davis (2007) produced early work using negative examples in the domain of chemical structure diagrams. All the combinations of up to 7 sequential strokes were generated and classified using a Support Vector Machine (SVM) classifier, either as a valid or invalid shape. The training data included examples of both valid and invalid candidates. Once the valid shape candidates were identified, the system ranked the overlapping candidates using recognition and context scores, and iteratively selected the best ones. Finally, domain knowledge was used to check whether the structure was chemically sound for the domain of molecular compounds. Later in the work by Ouyang & Davis (2009*a*), direction features were incorporated (Ouyang & Davis 2009*b*) for the shape representation, and used a joint probability to select the final set of shapes.

Bresler et al. (2013*b*) selected candidate shapes using all possible sets of up to five spatially proximal primitives for a flowchart dataset (Awal et al. 2011). Negative examples are included. However, as the number of negative examples is much higher than the number of valid shapes, the negative examples are clustered into several clusters. A multi-class Support Vector Machine (SVM) is used to accept valid candidates and reject the invalid ones. Since training the classifier does not efficiently equip the classifier to reject negative examples, Bresler et al. (2013*a*) select the final set of shape candidates by solving an optimisation problem. The optimisation problem is formulated to model the relations in a diagram. The model of relations is based on the connection points of the shapes. For example, for the flowchart diagram domain, four connection points are defined for the non-arrow shapes and two connection points for each arrow. Three

shape relations are then defined:

- Conflict: Two candidates share one or more strokes, or two arrows are connected to the same connection point of a shape.
- Overlap: Two shape candidates have overlapping bounding boxes.
- Endpoint: Each arrow requires existence of both shapes it connects.

All pairs of candidates get a score based on these relations and each shape candidate gets a score based on the classifier's confidence level. Selecting the best combination of shape candidates is then formulated as a max-sum problem, in which the combination with the highest score is sought. This optimisation technique is specifically designed for arrow-connected diagrams with specific requirements such as two arrows not pointing to the same connection point of a shape. Therefore, this approach is considered domain dependent and would not be applicable to all domains.

Later Bresler et al. (2014) used the trajectory-based normalization and direction features introduced by Liu & Zhou (2006) for shape representation, and improve their approach by separating the arrow detection unit as a process after recognising the regular shapes. Having too many shape candidates makes the recogniser slow (Bresler et al. 2015*b*). To reduce the number of shape candidates, Bresler et al. (2015*b*, 2016*a*) use the Single Linkage Agglomerative Clustering (SLAC) algorithm of Delaye & Lee (2015) for grouping. Bresler et al. (2016*a*) use synthetically generated data to balance the valid and negative classes and avoid performing clustering on the negative class. This approach has achieved the best accuracy in grouping and recognition on the flowchart (Awal et al. 2011) (with 84.2% accuracy) and Finite Automata (FA) (with 98.5% accuracy) (Bresler et al. 2014) datasets. The reported results also show that their approach is quite fast (0.78 seconds and 0.69 seconds on the flowchart and FA datasets respectively). However, as mentioned earlier, this approach is domain dependent since the optimisation problem is formulated for arrow-connected diagrams with specific requirements.

In these methods, the recognisers need to learn about very large invalid classes, which requires a large amount of data, since there are many ways that negative examples can be represented. Selecting the final set of shape candidates through optimization techniques is also a computationally expensive task. Moreover, the optimisation and final checks usually require domain knowledge, limiting these approaches to the specified domains.

A summary of work using negative examples can be found in Table 2.1. The accuracy column in this table reports the statistics from each paper. We report the accuracy of grouping and recognition of each approach, unless otherwise stated. All the approaches

also support interspersed drawing[1] and are domain independent [2], unless stated otherwise in the limitations column. The proximity column shows what type of vicinity between strokes are being considered (spatial, temporal or both).

	Proximity	**Accuracy**	**Limitations**
Bresler et al. (2013*b*)	Spatial	91.9% on flowcharts (Awal et al. 2011)	Limited size of strokes per shape
Bresler et al. (2013*a*)	Spatial, temporal	88.7% in grouping on flowcharts (Awal et al. 2011)	Domain dependent
Bresler et al. (2014)	Spatial, temporal	82.8% on flowcharts (Awal et al. 2011), 94% for FA (Finite Automata) dataset	Limited to arrow connected diagrams; domain dependent
Bresler et al. (2015*b*)	Spatial, temporal	84.2% of flowcharts (Awal et al. 2011), 95.5% for FA (Bresler et al. 2014)	Requires shape relations; domain independent
Ouyang & Davis (2007)	Temporal	85% of molecular compounds diagrams without domain knowledge and 89% with domain knowledge	Does not allow interspersed drawing
Ouyang & Davis (2009*a*)	Spatial, temporal	97% of the Molecular diagrams, 91% of Electrical circuit diagrams	Computationally expensive training and recognition process

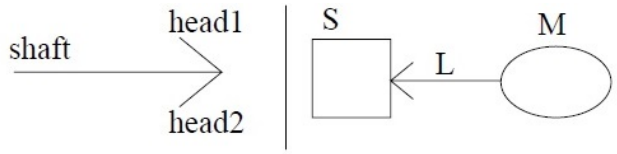
Table 2.1: Summary of work using negative examples. The accuracy column in this table reports the statistics from each paper.

Grammar and Language

Grammar and language techniques have been explored in wider sketch recognition tasks, including grouping. These approaches typically define a language to describe shapes, relationships, and their constraints. The language is then used as a key part of grouping primitives and recognising the groups (see the summary in Table 2.2). Variations on the form of languages exist, such as those described by Alvarado & Davis (2004) where

[1] Interspersed drawing refers to the drawing behaviour of starting a new shape before completing the previous ones.

[2] An approach is domain dependent if it is only applicable to a specific diagram domain.



DEFINE ARROW
(Subshapes
L1,L2,L3: (Line *shaft head1 head2*))
(Constraints
C1: (coincident *shaft.p1 head1.p1*)
C2: (coincident *shaft.p1 head2.p2*)
C3: (equal-length *head1 head2*)
C4: (shorter *head1 shaft*)
C5: (acute-angle *head1 shaft*)
C6: (acute-angle *head2 shaft*)
DEFINE MOTHER-SON
(Subshapes
M,S,L: (Female *M*) (Male *S*) (Child-link *L*)
(Constraints
C1: (touches *L.head S*)
C2: (touches *L.tail M*))

Figure 2.2: The description of the shape "arrow" in the family tree domain from Alvarado & Davis (2004).

a hierarchical shape language is used (see Figure 2.2 for the definition of an arrow in the family tree domain), while in the work done by Julca-Aguilar et al. (2017) a set of predefined graph grammars are used to generate potential shape candidates and their possible relations. Groups are constructed based on how well they conform to the shape descriptions, where the descriptions rely heavily on spatial and temporal proximity.

Groups are further evaluated in various ways. Costagliola et al. (2005) use a parser to generate multiple parse trees, each providing a possible interpretation of the sketch. Each tree is assigned a probability based on the fitting error of strokes and the accuracy of shape relationships. The most likely tree is chosen as the output. In SketchREAD (Alvarado & Davis 2004) a Bayesian Network evaluates the strongest interpretation of the diagram. Given the descriptions and constraints, the system also tries to find the missing parts from partially drawn shapes by looking into spatially and temporally close strokes. Further improvements to SketchREAD have been made (Alvarado & Davis 2006) using dynamically constructed Bayesian networks to determine how well each shape candidates fits the data. Julca-Aguilar et al. (2017) use a classifier to prune the search space by rejecting groups below some level of confidence; this may include groups representing negative example classes. The remaining shape hypotheses are then chosen by optimising a cost function that considers the likelihood score of shapes and their relationships.

Computational time is still a significant issue for these approaches, particularly if

	Proximity	**Accuracy**	**Limitations**
Costagliola et al. (2005)	Spatial, temporal	Not tested	Requires shape definition; hard to extend; domain dependent
Alvarado & Davis (2004)	Spatial, temporal	77% of family tree diagrams, 62% of digital circuit diagrams	Requires structural definition of shapes and their relationships for a specific domain; domain dependent
Hammond & Davis (2009)	Spatial, temporal	100% accuracy for Japanese Kanji, Military course of action and Biology diagrams	Requires shape definition; domain dependent
Julca-Aguilar et al. (2017)	Spatial, temporal	85.5% on flowcharts (Awal et al. 2011) when texts are manually excluded	Relies on the grammar; domain dependent

Table 2.2: Summary of work using grammar and language. The accuracy column in this table reports the statistics from each paper

interspersed drawing is allowed. Hammond & Davis (2009) explore these issues by proposing a grouping method that examines all possible shape combinations, but uses an indexing technique to reduce computation time. Although they achieve 100% recognition accuracy, the computation time in the worst case is exponential. Also, the recognition is limited to the constraints defined by the language, making the system domain dependent.

The main limitation of these methods are that the language must be defined by an expert, for each domain. It is also difficult to encode the levels of ambiguity in these languages that we know are present in sketched diagrams. The accuracy of these approaches is therefore limited by the difficulties in defining a language.

Optimisation

In optimisation approaches, grouping and recognition are performed simultaneously, and a model of the primitives is optimised to group and recognise shapes. The main technique demonstrated in the literature is to use time-based models (Sezgin & Davis 2005, 2007*a*, 2008) to group and recognise shapes i.e. those that only consider temporal information (see the summary in Table 2.3).

In the work by Sezgin & Davis (2005) the temporal order of shapes are modelled with different Hidden Markov Models (HMMs). To interpret a diagram, a graph showing the temporal order of primitives is produced, with the addition of edges that represent possible shape candidates (weighted with the log-likelihood of it matching a known

shape). The shortest path between the first and last primitive (node) of the diagram is used to determine the optimal grouping and recognition of shapes, where each edge in the shortest path represents a valid shape. In later work by Arandjelović & Sezgin (2011), a similar approach was used to construct the graph. However in this work, the optimal shape candidates are determined using dynamic programming, with information combined from time-based and image-based recognisers. In addition, a one-class SVM classifier is used to reject invalid shape candidates. Sezgin & Davis (2007*a*) also extended their earlier work (Sezgin & Davis 2005), by using a dynamic Bayesian network where object-level patterns (the sequence of drawn objects) are considered, as well as the temporal ordering of primitives. An extra node is added for each primitive to denote if the shape is complete. The model then tries to maximise the joint likelihood of stroke-level and object-level patterns, resulting in an optimal grouping and recognition of the sketch.

Since these approaches only rely on temporal order, interspersed drawing cannot be supported. This issue has been resolved for grouping by Sezgin & Davis (2008) who considered an additional "switching node" (MUX) for each observation that indicates whether the user has interspersed any two objects, but drawing order is still considered when recognising shapes. The optimisation techniques are generally computationally expensive and the drawing order can affect the accuracy.

Other Simultaneous Methods

There are also other approaches that perform grouping and recognition simultaneously (see Table 2.4). In the work of Hammond & Paulson (2011), all strokes are first given to PaleoSketch (Hammond & Davis 2009), a low-level recogniser capable of recognising nine primitives (such as lines, arcs and curves). From the recognised primitives, a neighbourhood graph is constructed, and then used to search for connected components using Tarjan's algorithm (Tarjan 1972). To find valid shapes, all sub-graphs of each connected component are recognised using the neural network version of PaleoSketch (Paulson & Hammond 2008). The group must then pass a false-positive removal stage in order to be deemed a correct grouping. The false-positive removal stage follows five rules to determine if a group of strokes should be accepted. These rules are set empirically through observations. This work is computationally expensive and has limitation in supporting complex shape recognition.

Johnston & Alvarado (2013) explore grouping for the domain of digital circuit diagrams. In their approach, the strokes are first classified into *gate, text* or *connector* using the method described by Peterson et al. (2010). To group the *gate* strokes, a neighbourhood graph is constructed by placing a bounding box around each stroke to locate spatially close strokes. Connected components of the graph are found by performing a

	Proximity	**Accuracy**	**Limitations**
Sezgin & Davis (2005)	Temporal	96.5% recognition rate on 88 objects from different domains including military course of action, stick-figures, and mechanical engineering	Relies on temporal pattern of how objects are drawn; does not allow interspersed drawing
Arandjelović & Sezgin (2011)	Temporal	36.3% accuracy in grouping and recognition of full Course of Action diagrams (partial recognition of a diagram is not counted)	Relies on temporal pattern of how objects are drawn; Limited temporal window; does not allow interspersed drawing
Sezgin & Davis (2007*a*)	Temporal	Ranging from 77.4% to 93% for different users on Circuit diagrams	Relies on temporal pattern of how objects are drawn; does not allow interspersed drawing
Sezgin & Davis (2008)	Temporal	Ranging from 87.7% to 95.6% for different users on circuit diagrams	Computationally expensive

Table 2.3: Summary of groupers using optimisation. The accuracy column in this table reports the statistics from each paper

	Proximity	**Accuracy**	**Limitations**
Hammond & Paulson (2011)	Spatial	89.7% weighted accuracy for Military course of action, circuit diagrams, chemistry diagrams, mechanical engineering diagrams	Computationally expensive
Johnston & Alvarado (2013)	Spatial, temporal	94.2% accuracy on digital circuit diagrams	Limited to spatial bounding box size; the results are reported for the case that text and connectors are manually excluded

Table 2.4: Summary of other simultaneous techniques. The accuracy column in this table reports the statistics from each paper

series of breadth-first-searches. All possible subsets of each connected component (up to size 5) are evaluated by calculating their Hausdorff distance to the template shapes. Each shape is represented with the direction features introduced by Ouyang & Davis (2009*b*). The subset that matches most closely to a known shape goes through a domain specific check using an encoding language before finalising a shape. Although this approach has performed well in the experiments on digital circuit diagrams (producing a recognition rate of 94%), all the *text* and *connector* strokes are excluded manually. Having the connectors excluded greatly simplifies the grouping and recognition problem. Moreover, domain knowledge needs to be given to the system, which makes it domain dependent.

2.2.2 Sequential Grouping and Recognition

These approaches follow a pipeline of grouping and recognition as separate tasks occurring sequentially and independently. We have further categorised these sequential approaches into optimisation and clustering techniques. The general processes of sequential grouping and recognition are shown in Figure 2.3 and Figure 2.4. Figure 2.3 shows the pipeline of sequential optimisation and hard-clustering approaches. Hard-clustering approaches do not have the optimisation task (showed in the dashed box). Figure 2.4 also shows the pipeline of Probabilistic Graphical Model (PGM) based recognition systems. The difference between these two pipelines is that in PGM-based approaches, the grouping occurs after the recognition, whereas in the sequential optimisation and hard-clustering approaches, the grouping task is performed before the recognition.

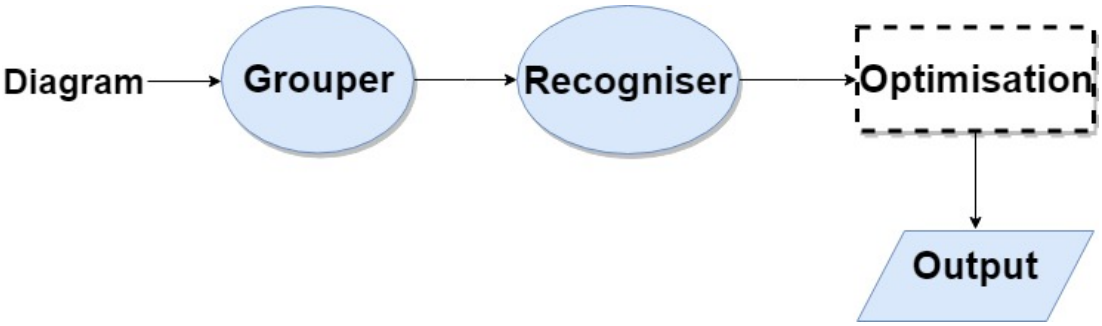


Figure 2.3: Sequential optimisation based and hard-clustering based grouping and recognition process

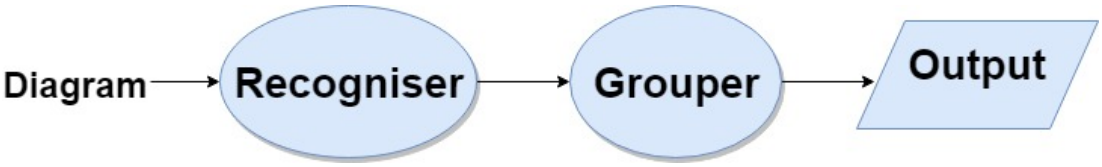


Figure 2.4: Sequential PGM-based grouping and recognition process

Sequential Optimisation

Sequential optimisation approaches generally follow a sequence of group-recognise-optimise. Spatial/temporal information about the primitives are used to generate possible shape candidates, each candidate is recognised, and finally the best set of shape candidates is chosen by optimising a given cost function (see Table 2.5 for a summary of work using sequential optimisation).

Shilman et al. (2004) approach the grouping and recognition of characters and shapes as an optimisation problem. The spatial proximity is represented by constructing a neighbourhood graph of strokes, with the strokes being nodes, and the edges between them representing spatial contiguity (within some pre-defined threshold). All connected components of the graph up to size k are found, where each component hypothesises a shape candidate. The aim of the recognition framework is to find the best grouping and recognition of strokes to optimise a global cost function. The cost function considers the shape recognition cost, the combination cost of two subsets of the graph, and the constraint cost that checks the connectivity of subgraphs. The rendered image of candidate strokes, along with some context images, are used as the input to the recognition system. Context images are rendered from strokes connected to some of candidate's strokes in the neighbourhood graph, which are used to aid the identification of invalid candidates. Each of these principal images is modified with four additional features including curvature, orientation and end-point information. In order to avoid any redundant computation, Dynamic Programming is employed to reuse already computed cost functions. A multi-class variant of the confidence rated boosting algorithm is used

for the classification. Shilman & Viola (2004) extended the work of (Shilman et al. 2004) by using A^* search to prune away parts of the search space that could not possibly lead to a viable solution. The experimental results demonstrate that the proposed system can achieve 97% grouping/recognition for 13 shape classes collected from 19 different users, although these are isolated shapes rather than full diagrams. On full flowchart diagrams, Shilman & Viola (2004) report a 85% grouping/recognition rate.

In a similar manner, Feng et al. (2009) generate shape candidates based on temporal proximity and maximum size, but with two additional constraints. Temporal proximity is considered by allowing a maximum number of time jumps between primitives within a group, to account for the interspersed nature of sketching. Also, a maximum overlap ratio is used as an extra spatial constraint, where groups with a significant overlap of primitives are merged, as they are considered part of the same shape. Each shape candidate is first checked to see if it is a connector using Least Square Fitting Error (LSFE). If the candidate is not a connector then a neural network (that is trained with genuine and synthetic data) is used for the classification. Each candidate is assigned a score based on the shape resemblance (based on the LSFE if is a connector, otherwise the class probability distribution provided by the neural network) and the connectivity requirements of the shape. The connectivity requirements are employed to aid eliminating the pseudo-shapes (either containing strokes from multiple shapes or an incomplete shape) as the classifier has limited capability in rejecting outliers only based on the produced class probabilities. A cost function based on the shape resemblance and the connectivity requirements of the shape is then optimized. This approach is specifically designed for circuit diagrams.

In the work by Ouyang & Davis (2011), candidate shapes are generated using temporal and spatial proximity. Recognition and optimisation of the solution is performed using a Conditional Random Field (CRF). The CRF model used here constructs a hierarchical model that combines features measured from three levels: raw ink points, primitives and candidate shapes. The joint model captures the relationship between these levels. Each primitive has a shape candidate node that represents all possible shape candidates that includes that primitive. The inference process selects the best shape candidate (with the maximum likelihood) for each primitive. The results show that the proposed algorithm is able to detect and classify 97.4% of shapes in a chemical drawing dataset. However, the inference process is computationally expensive.

Although optimisation approaches have yielded reasonable recognition rates, the search and optimisation process is still computationally expensive.

	Proximity	**Accuracy**	**Limitations**
Shilman et al. (2004)	Spatial	94% of mathematical expressions	Computationally expensive
Shilman & Viola (2004)	Spatial	97% of isolated shapes of HHreco Hse & Newton (2004), 85% on synthesized flowchart diagrams	Computationally expensive
Feng et al. (2009)	Spatial, temporal - constrained to max # of time jumps	90.29% accuracy in class-level recognition of digital circuit diagrams	Fine tuned to circuit diagrams; domain dependent
Ouyang & Davis (2011)	Temporal, spatial	97.4% of molecular diagrams	Employs domain knowledge about chemical structures; Computationally expensive

Table 2.5: Summary of work using sequential optimisation. The accuracy column in this table reports the statistics from each paper

Clustering-Based Groupers

Clustering-based groupers capture some pairwise features for each pair of strokes in the diagram. Based on the pairwise features, they decide which strokes together form a distinct object. We further separate this approach into hard clustering methods, and those that use Probabilistic Graphical Models (PGMs).

Hard clustering Delaye & Lee (2015) propose a grouping method based on the Single Linkage Agglomerative Clustering (SLAC) algorithm. The algorithm starts by defining each stroke as its own cluster and successively merges the two closest clusters unless the distance between the closest clusters is above a threshold. The merged strokes of each cluster at the cutting threshold represent a symbol grouping. The proposed algorithm learns to minimize the distance between strokes within the same shape while maximizing the distance between strokes from different clusters, and adjusts the threshold accordingly. The closeness of the clusters is calculated through a weighted distance function that captures the pairwise stroke spatial and temporal distances. The weights of the pairwise features and the cutting threshold are learned through an optimisation process using the LBFGS second-order gradient descent method (Liu & Nocedal 1989). The proposed algorithm is evaluated on various datasets including flowcharts, finite automata, mathematical notation, synthesized diagram from these datasets and free-form documents, with the grouping accuracy ranging from 75% to 95.55% for different datasets.

In order to solve the problem of exhaustive search in traditional methods, a grouping technique has been designed with two levels of classification (Peterson et al. 2010). This strategy reduces the problem of exhaustive search into a classification problem. The initial step is the single stroke classification, which classifies strokes into coarse classes depending on the context. For example, in the domain of digital circuit diagrams, each stroke is classified as being part of a wire, gate, or text. Two methods are used to perform the grouping on each coarse class. One uses a simple threshold based on the spatial distance and the elapsed time between two strokes. The other approach (IPC-MD) uses the AdaBoost classifier trained with 13 pairwise features to classify each pair as: "don't join", "far join" or "near join". Once all pairs are labelled, in an iterative process, pairs of strokes with "far join" or "near join" are clustered together. Stahovich et al. (2014) point out that the IPC-MD often confuses the "far join" class with "near join" and "no join" classes. To achieve a higher accuracy, they propose a new training approach (IPC-IR), which uses cluster accuracy to iteratively re-label the training data. The iterative labelling uses the IPC-MD method to assign labels to pairs of strokes and iteratively optimises the accuracy of the classifier. In the experiments, in cases where the differences in accuracy were statistically significant, the IPC-IR and IPC-MD methods outperformed the thresholding method, while achieving comparable performance to each other.

Stevens et al. (2013) add 11 new pairwise features to the 13 features introduced by Peterson et al. (2010) for the coarse level classification, which has led to an increase in the grouping accuracy. However, Lee et al. (2012) found that only 6 features that are introduced by Peterson et al. (2010) were useful for training a C4.5 decision tree to determine if two strokes belong to the same character for equilibrium equation recognition. This grouping approach is employed in a language and grammars based sketch recognition system (Costagliola, Rosa & Fuccella 2014, Costagliola et al. 2015) that uses local context for recognition.

In a similar manner to the two level classification, Alvarado (2007) first labels strokes as wire or gate (in the digital circuit domain) using Conditional Random Fields (CRF). A neighbourhood graph is then constructed on gate strokes where each connected component of the graph represents a shape. This algorithm is tested on 51 circuit diagrams with 5 different classes (wire, AND, OR, XOR and NOT gates), achieving 77% accuracy in grouping. Although these approaches avoid the exhaustive search and are domain independent, the accuracy of grouping is relatively low. The summary of these methods can be seen in Table 2.6.

Probabilistic Graphical Model based The following work differs from the hard clustering approaches as recognition of primitives as their corresponding shapes is performed before clustering primitives into groups. These approaches use Probabilistic

	Proximity	**Accuracy**	**Limitations**
Delaye & Lee (2015)	Spatial, temporal features	88% in grouping for flowchart dataset, 98% for FA dataset	Relatively low accuracy for grouping of flowcharts
Peterson et al. (2010)	Spatial and temporal features	79% grouping accuracy for circuit diagrams - 69% grouping accuracy for family tree diagrams	Relatively low accuracy for grouping
Stevens et al. (2013)	Spatial and temporal features	Overall 80.96% accuracy in grouping for class diagram, Digital circuit diagram, Family tree and Flowchart diagrams	Requires expensive training process - low accuracy
Stahovich et al. (2014)	Spatial and temporal features	79% grouping accuracy in digital circuits - 72% grouping accuracy in family tree diagrams	Requires expensive training process - Accuracy
Alvarado (2007)	Spatial	77% in grouping of digital circuit diagrams	Low accuracy

Table 2.6: Summary of work using hard clustering techniques. The accuracy column in this table reports the statistics from each paper.

Graphical Models (PGMs) for primitive recognition. Deufemia et al. (2014) propose a sketch recognition system based on Latent-Dynamic Conditional Random Field (LDCRF) to model the sub-structure of a shape by learning relationships between shape classes. In this approach, recognition and grouping is performed in two stages; first classifying each primitive using the trained LDCRF, and then applying unsupervised geometric distance-based clustering algorithm to group the primitives belonging to the same shape. To build the model, for each stroke some feature points (endpoints, corners or overlapping points) are calculated and for each feature point a set of features including orientation, length, distance, direction and time are calculated. In this model, feature points are considered to be the observations, while class membership is represented by hidden states. The model determines the more appropriate hidden state (class) for each observation, based on its features and the features of neighbouring observations. In the training process, the model parameters are learned to maximize the likelihood of labelled observation sequence. The inference process is to estimate a sequence of labels for the given observation sequence that maximizes the sum of marginal probabilities. Since the loops in the graph make the exact inference intractable, loopy belief propagation (Pearl 1988) is used to estimate the marginal probabilities. Once labels are predicted for all feature points, an unsupervised geometric distance-based clustering algorithm is

	Proximity	**Accuracy**	**Limitations**
Delaye (2014)	Spatial and temporal features	75.50% accuracy on flowchart dataset, 75.50% accuracy on FA	Computationally expensive inference process - Low accuracy for flowchart dataset
Deufemia et al. (2014)	Spatial, temporal	Ranging between 81.3% and 91% for electric circuit diagrams	Computationally expensive inference process
Wang et al. (2016)	Spatial, temporal	84.3% accuracy for flowchart dataset, 95.8% accuracy for FA	Computationally expensive inference process

Table 2.7: Summary of work using PGM-based techniques. The accuracy column in this table reports the statistics from each paper

applied to group the same symbol strokes. The algorithm is evaluated on the domain of electric circuit diagrams, achieving accuracy values between 81.3% and 91%.

Wang et al. (2016) deployed a Markov Random Field (MRF) at the primitive level to capture the local context (i.e., relationships between primitives). The stroke relationships are defined as 'same symbol' or 'others'. Having the inferred primitive classes and relationships from the model, grouping is done by merging neighbouring strokes from the same class with the 'same symbol' relationship. In the work of Wang et al. (2017) object relations are added to the system using grammatical descriptions of the domain to ensure the global consistency of the recognition. The grammar also forces the explanation of all available strokes; hence, if there are leftover strokes, a backtracking process explores another solution that can explain all the strokes.

In the work by Delaye (2014) two tree-structured CRFs are compared for predicting stroke classes; Minimum Spanning Tree (MST-CRF) and Hierarchical clustering tree model (H-CRF). After determining the class for each stroke through the MST-CRF, groupings of the sketch are obtained by cutting the tree where the weights of the edges are higher than a threshold, and where connected nodes have different shape interpretations. The structure used in the H-CRF is a dendrogram that is obtained by applying the SLAC algorithm described by Delaye & Lee (2015) to the sketch. The leaves of the dendrogram model the class of strokes and the nodes model the clusters of strokes. Cutting the dendrogram on edges with a weight less than a threshold produce the groupings of strokes. The experimental results show that the H-CRF outperforms the MST-CRF. These approaches usually have a computationally expensive inference process. The summary of these methods can be seen in Table 2.7.

2.2.3 Other methods

In addition to the methods in these categories, there are techniques that perform sketch recognition from a different perspective (see Table 2.8 for a summary of these methods). For example, there are techniques that perform grouping from a top-down perspective, where the entire diagram is considered as a whole and then broken down into pieces sequentially. Kara & Stahovich (2007) propose a mark-group recognition technique. This technique relies on "marker symbols" that can be accurately and inexpensively extracted from the input data. The set of strokes coming before the ends of the marker are considered to be shapes. These techniques rely on the existence of markers, and so cannot be applied to the domains that do not contain a marker. Saund & Lank (2003) decompose a sketch into sequences of contiguous line segments corresponding to line art, and "blobs" of dense ink corresponding to text. They use Gestalt principles (Goldmeier 1972) to group these objects into larger structures. The approach is computationally expensive for dense diagrams, and is intended to produce groupings suitable for interactive manipulation rather than object recognition (Lin 2014).

Chao et al. (2017) present a novel gaze-aided grouping method for flowchart diagrams. Gaze data is collected during the drawing of diagrams and is used to assign a heat value to every pixel in the sketching canvas. Since the heat values of arrow regions are low, as the eye spends less time there, the regions with high heat values are the potential parts to be searched for closed shapes.

Julca-Aguilar & Hirata (2018) propose a symbol detection and recognition system using a deep neural network. In this research Faster Region-based Convolutional Neural Network (Faster R-CNN) (Ren et al. 2017) is used to to find the object regions. First, each sketch is scaled so the largest dimension of its bounding box is equal to a fixed parameter, resulting in gray-scale images. The bounding boxes around each object is also determined and labelled accordingly. These images are then fed into a Faster R-CNN, which is composed of three components: feature extractor, regional proposal network (which generates bounding box coordinates), and a region classifier (for recognising each region). For the feature extractor they have considered different networks, such as Inception V2 (Ioffe & Szegedy 2015), Resnet 50 (He et al. 2016), Resnet 101 (He et al. 2016), and Inception Resnet V2 (Szegedy et al. 2017), resulting in different training and inference times and accuracy. They have evaluated their approach on two public datasets of mathematical expression (Mouchère et al. 2016) and flowchart (Awal et al. 2011). However, since they evaluate their method at the bounding box level (mean average precision (Everingham et al. 2010)), the results are not directly comparable with other methods.

	Proximity	**Accuracy**	**Limitations**
Kara & Stahovich (2007)	Spatial	70% in recognising arrows for modelling Matlab's Simulink package	Limited to arrow connected diagrams; domain dependent
Chao et al. (2017)	Spatial	71.15% in grouping of shapes (excluding arrows) in flowchart diagrams	Requires peripheral devices, low accuracy, limited to closed shapes
Julca-Aguilar & Hirata (2018)	-	-	Requires a large dataset for training; The training and inference process are computationally expensive
Schäfer & Stuckenschmidt (2019)	-	93.5% in grouping and recognition of flowchart diagrams	Designed specifically for flowchart diagram; domain dependent

Table 2.8: Summary of other grouping methods. The accuracy column in this table reports the statistics from each paper

(Schäfer & Stuckenschmidt 2019) propose an extension of Faster R-CNN for flowchart diagram recognition. The object detector in this approach is extended with an additional arrow keypoint predictor that predicts the arrow heads and tails locations. The network architecture is designed to take the arrow keypoints information into consideration, which restricts this approach to the flowchart domain. They have also proposed an extension of their work to support recognition of online flowchart diagrams that matches strokes with detected objects. This process is also done in a specific order that takes the flowchart diagram structure into account. Since this approach is designed for flowchart diagrams, the grouping and recognition rate is relatively high (93.5%) on the flowchart dataset.

2.2.4 Summary

We believe an ideal grouping strategy should have the following characteristics:

- High accuracy
- Computationally inexpensive (for training and testing)
- Domain independent
- Supportive of a free sketch environment

In fact, these characteristics extend to all parts of a sketch recognition engine. Using our analysis of grouping approaches presented in this section, we have summarised the

		Accuracy	Computationally Inexpensive	Domain Independent	Support Free Sketching Environment
Simultaneous	Negative Examples	∼	∼	✓	∼
	Grammar & Language	✗	✗	✗	∼
	Optimisation	✗	✗	✓	✗
Sequential	Sequential Optimisation	✓	✗	✓	∼
	Clustering: Hard Clustering	✗	✓	✓	✓
	Clustering: PGM-based	✗	✗	✓	✓

Table 2.9: Comparison of grouping approaches. ✓= Exhibits the characteristic, ✗=Does not exhibit the characteristic, ∼=Dependant on other factors

success each approach has had in exhibiting these characteristics (see Table 2.9). Of course, without a full comparative study of each grouping technique, it is difficult to judge the relative accuracy of different approaches, particularly when different datasets are used; here we comment on them based on the reported results - which in many cases, indicates their performance in the best case scenario.

The characteristics for approaches that use negative examples is largely dependent on the number of primitives allowed per shape. Generally, a limit is chosen that is optimal for the size of valid shapes in the training set. Increasing this limit would increase the number of negative examples that should be included in training exponentially. Therefore, the number of shape candidates (both for training and testing) grows when a larger number of primitives per shape is allowed. A very large training set with a very large invalid shape class has the potential to affect the accuracy of the classifier. A greater number of shape candidates would result in computationally expensive training and testing phases. Although Bresler et al. (2015*b*) reduce the number of shape candidates by using the method of Delaye & Lee (2015), the recogniser still does not perform well in rejection and a final structural analysis step is carried out.

Grammar and language based approaches have reported low accuracy, except the approach taken by Hammond & Davis (2009) which is highly constrained and has an exponential time computational complexity. These approaches are usually computationally expensive, as an exhaustive search is required to match the drawn strokes with the shape definitions of a language. They are also domain dependent, as the description and constraints are specifically defined by an expert for a particular domain. In addition, the sketching environment is limited by the drawing constraints imposed by the language, where any shape that does not conform to the language cannot be recognised. Defining a language that is robust enough to handle the ambiguities of sketch recognition is a difficult task. Overall, this approach has not been as successful as others in meeting the goals of sketch recognition. However, for domain specific solutions, this may be a suitable approach.

The accuracy of simultaneous optimisation approaches varies (see Table 2.3)We

believe the accuracy is limited because these approaches rely on the temporal ordering of primitives. They typically have a high computation time, as they need to maximise a joint probability for the entire diagram. Although Sezgin & Davis (2008) have attempted to allow interspersed drawing, these approaches have limited support for a free-sketch environment, as users need to follow a similar order of drawing to that represented in the dataset.

The sequential optimisation approaches have achieved a reasonable accuracy, and are generally domain independent. However, these approaches are usually computationally expensive as they carry out an optimisation process. In addition, given that these techniques typically limit the number of primitives allowed per shape, their support for a completely free sketch environment is questionable.

The sequential approaches that use hard clustering for the task of grouping have reported relatively low accuracy rates. It is possible that the accuracy could be improved by designing more discriminating feature sets. The computation time of these approaches is polynomial, although during the classification process, the calculation of pairwise features for each pair of primitives can take some time in practice. However, one advantage is that they avoid generating too many shape candidates in the grouping phase, which maintains low computation time for the recognition step. Given that these methods are trainable, they can be applied to multiple domains. They also support a free sketch environment.

The sequential PGM-based approaches are also domain independent and allow for a free sketching environment, like the hard-clustering methods. However, they have a high computation time as the maximisation of joint probability (in the inference process) is computationally expensive. The reported accuracy of these approaches are relatively low.

None of the mentioned grouping techniques satisfies all the required characteristics. We believe in the case of over-segmentation (producing large number of shape candidates), a more intelligent grouping algorithm that is capable of pruning the search space is needed. We also believe that a domain independent recogniser is needed to accurately reject the invalid shape candidates.

2.3 Shape Recognisers

One of the core components of a sketch recognition system is the shape recogniser responsible for identifying sketched shapes. Different shape representation and classification methods used in the literature will be discussed in this section.

Shape recognition techniques can be categorised in several ways. One way is based on the type of shapes they recognise, which is single stroke or multi-stroke shapes. Single-stroke recognisers are more suitable for the task of gesture recognition or shapes that

are drawn with a single stroke. Diagrams, on the other hand, can include more complex shapes requiring users to draw components using multiple strokes. In addition, allowing multi-stroke shapes is essential to supporting a free sketching environment. Therefore, we have focused on reviewing multi-stroke shape recognisers, although we have included seminal work on single stroke recognisers.

In this section, in order to capture the variety of approaches used for shape recognition, we have categorised existing recognisers into three categories based on the choice of classification: feature-based approaches, template matching, and grammar-based recognisers. There is also a group of sketch recognition systems that use Probabilistic Graphical Models (PGMs), which identify shapes by modelling the relationship between shapes and strokes. Although these methods perform stroke-level labelling, they do not have a separate shape recogniser component that could be covered in this section. These sketch recognition systems were reviewed in Section 2.2.2

2.3.1 Feature-based Methods

Feature-based approaches are based on calculating various ink features to train a classifier for recognition. These methods require a suitable set of features that can describe different aspects of strokes to be computed and used as input to a machine learning algorithm for classification. In early research, Rubine (1991) proposed a trainable single-stroke gesture recogniser that used 13 features to capture different aspects of a gesture. Training samples are used to calculate weights for each feature. For a given undefined gesture, the stroke's features with their corresponding weights are given to a linear classifier that generates a probability score for each class. The class with the highest probability score is selected as the classification result. Rubine's classifier was later used in many sketch recognition systems (Landay & Myers 1995, 1996, Damm et al. 2000, Chen et al. 2003, Plimmer & Apperley 2003, Pereira et al. 2004).

Other early work in this domain includes CALI (Fonseca et al. 2002), a multi-stroke recognition algorithm that finds geometric properties of the strokes and uses Fuzzy rules to classify geometric shapes. A trainable variant of CALI was tested with K-Nearest Neighbour, Naïve Bayes and a decision tree. The results show that the Naïve Bayesian Network was the easiest to implement and provided the highest recognition rate.

RATA.Gesture (Chang et al. 2012) is another single stroke recogniser that computes 114 ink features (Blagojevic et al. 2010) for each stroke. The features are chosen to measure different aspects of a stroke such as curvature, size, density, spatial and temporal relationships to other strokes. This is followed with a feature selection process to discard the misleading and duplicate information. A wide range of machine learning algorithms provided in Weka (Hall et al. 2009) were tested, and an ensemble of four of the best performing algorithms was chosen for classification.

PaleoSketch (Paulson & Hammond 2008) is capable of recognising 8 low-level single stroke primitives including line, polyline, circle, ellipse, arc, curve, spiral, and helix. PaleoSketch was first introduced as a geometric-based recognizer (Paulson & Hammond 2008) that heuristically checks rules and conditions for each primitive. Later on, PaleoSketch was extended in the research by Paulson et al. (2008) to combines 31 geometric features with the 13 features introduced in the work of Rubine (1991) to train a quadratic classifier that returns multiple ranked interpretations with normalised confidence values. A Multi-Layer Perception (MLP) version of PaleoSketch that produces normalised confidence values was also introduced in a paper by Paulson (2010).

Fahmy et al. (2018) propose a two step classification method for symbol recognition. In the classification pre-processing the input is classified as one of *closed shape, open shape*, or *incomplete shape* classes. A separate classifier is trained for open shapes and a separate one is trained for the closed shapes. Each input shape is represented with some geometric features, while SVM and K-NN were the choice of classifiers, with the SVM achieving better results. The choice of geometric features used in this research make this approach domain dependent. The proposed method can only recognise lines, triangles, rectangles, squares, circles, ellipses, diamonds, arrows, arcs, and zigzag lines, which needs to be further extended for other shapes.

The majority of the feature-based methods only support single stroke shapes or gestures. Although the neural network version of PaleoSketch (Paulson 2010) is capable of recognising multi-stroke shapes, it still cannot recognise complex shapes. Although most of these methods have performed well, yielding accuracies in a range of 90-98%, they cannot easily be extended for complex shapes (where a subset of a shape is also a valid shape).

2.3.2 Template Matching Methods

These techniques perform the classification by matching the given input to a set of training templates and choosing the best match as the classification result. The choice of shape representation and matching techniques varies in different approaches. We have categorised these methods based on the representation method.

$ Family Recognisers

The $ family recognisers were designed to be easy to implement and fast in terms of computation time. The first of these is the $1 recogniser (Wobbrock et al. 2007), a single stroke recogniser that performs classification in 4 steps. First, depending on the speed of drawing and the sampling rate of the hardware, gestures are resampled into N equidistantly spaced points. After the resampling process, the gesture is rotated so that the angle between the centroid of the gesture and the gesture's first point is 0°. Next,

the gesture is non-uniformly scaled to fit into a reference square. Lastly, the recognition is carried out by calculating the average Euclidean distance between the gesture's points with that of the template shape in the dataset. The template with the least distance is chosen as the result of recognition.

Protractor (Li 2010) builds on $1 by supporting both orientation-variant and orientation-sensitive shapes. For the orientation sensitive version, eight rotation variations of the gesture are considered for recognition. In the classification process, the distance is measured by calculating the inverse cosine distance between the input gesture vector and template gesture vectors. Since the rotation in the pre-processing step might not remove the orientation noise, the templates are also rotated by an extra amount when calculating the distances. The gesture template's label with the highest similarity (the inverse cosine) is then selected as the recognition result.

$N (Anthony & Wobbrock 2010) adds support of multi-stroke recognition to $1 algorithm. This algorithm generates all permutations of a multi-stroke gesture and stores them as a single stroke. Each permutation represents one possible combination of stroke order and direction. By connecting the endpoints of gesture strokes, a multi-stroke gesture is treated as a single stroke. Henceforth, the recogniser treats the multi-stroke gestures as single-stroke ones. The $N algorithm was extended to $N-Protractor (Anthony & Wobbrock 2012) that uses the same matching method as Protractor, resulting in more accurate recognition than the $N recogniser. Vatavu et al. (2012) introduced $P to overcome the problem of memory and computation cost in $N and $N-Protractor. The $P recogniser disregards the gesture timeline and treats gestures as an unordered list of points. The task of the recogniser is to match the points of a gesture to that of all templates by calculating the sum of Euclidean distances for all pairs of points. The template's label with the smallest distance is then selected as the classification result. Searching for the minimum matching distance is a computationally expensive task in these methods. $Q was introduced by Vatavu et al. (2018). It is slightly more accurate than $P, and is on average 46 times faster. $Q was designed on top of $P with some computation optimisation during the template distance calculation. This algorithm has only been tested on a dataset they collected.

ARG-Based Recognisers

Another approach to template matching is through representing shapes as an Attributed Relation Graph (ARG) describing the geometry and topology of a shape (Yin & Sun 2005, Lee et al. 2007). The recognition is carried out by matching the input shape's graph to that of each shape definition in the dataset. In the work of Lee et al. (2007) a set of attributes including the relative length of primitive to the total length of primitives constructing the shape, number of intersections between primitives, relative location

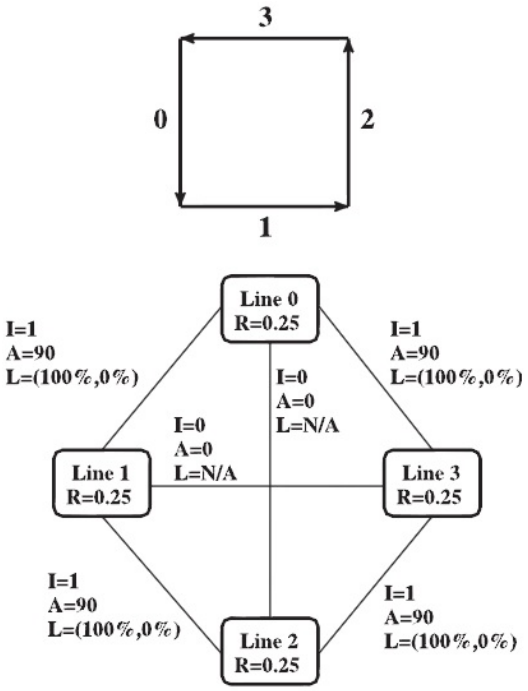


Figure 2.5: An ARG representation of a square (Lee et al. 2007)

of intersections and the angle of intersections are encoded into the ARG. Since graph matching is an NP-Complete problem, they propose five approximate matching techniques to find the best match between two ARGs. An example of an ARG representation of a square can be seen in Figure 2.5.

Costagliola, De Rosa & Fuccella (2014) present an ARG-based recognition system for partially drawn shapes. To estimate the similarity between partially drawn shapes and a template shape, they consider a number of possible subgraph isomorphisms. The mapping cost of each subgraph isomorphism is calculated and the minimum cost is selected as the matching distance between two shapes. Since the exact graph matching is computationally expensive, approximate graph matching procedure is used.

Although some techniques are proposed to speed up template matching for $-family (Reaver et al. 2011*b*, Taranta & LaViola 2015) and ARG-based recognisers (Lee et al. 2007), the recognition process still remains computationally expensive. The template matching recognisers can be used for complex shapes, however, these approaches are computationally expensive in the classification process. Recognisers with high computation time are not suitable for sketch recognition systems that generate several shape candidates. In addition, the ARG-based approaches require an expert to define the ARGs for each shape in the domain.

Appearance-Based Recognisers

Appearance-based approaches only consider the visual appearance of shapes, rather than the properties of each individual stroke or their relationships with each other. Although the choice of classification for these approaches can be both template matching and discriminative classifiers, for simplicity we have categorised them as template matching approaches.

Visual appearance, as a choice of shape representation, has been investigated in different ways (Shilman & Viola 2004, Shilman et al. 2004, Oltmans 2007, Deng et al. 2013). For example, Hse & Newton (2004) calculate Zernike moments (a form of rotation invariant image representation) as features for training three different classification algorithms: Support Vector Machine (SVM), Minimum Mean Distance (MMD) and Nearest Neighbour (NN), having SVM achieving the highest recognition rate. In the work of Kara & Stahovich (2005), shapes are represented as 48×48 down-sampled binary images and recognition is carried out in two steps. As a pre-recognition step, the input is incrementally rotated and compared to the shapes in the dataset to determine the best alignment angle. Next, as a recognition process, four algorithms calculate the spatial distance between the aligned input and the shapes in the dataset. The result of the four classifiers are then combined to produce a single recognition value describing the similarity of the input and shapes in the dataset.

Ouyang & Davis (2009*b*) present a shape recogniser that focuses on the visual properties of shapes while exploiting the temporal information of strokes. As a pre-processing step, strokes are first resampled to a constant spatial frequency and then each shape is normalised by translating its centre of mass to the origin, and scaling it horizontally and vertically. The on-line stroke sequences are then converted into a set of low resolution feature images by calculating four orientation features and one endpoint feature. The classification is performed by computing an image deformation model (IDM) (measuring the distance of the input to all the templates in the training set), resulting in a high recognition rate on three different datasets (ranging from 96.2% to 99.2%). They also use discriminative classifiers (i.e., SVM with linear and RBF kernel functions) resulting in a high accuracy. This work has been extended by Johnston & Alvarado (2013) to include velocity information of drawing for digital circuit domain, which improved the results. A similar shape representation was used by Liu & Zhou (2006) for online Japanese character recognition, but with a different pre-processing steps.

It has been shown that the choice of features and classifiers can affect the recognition accuracy in appearance-based recognisers (Tumen et al. 2010). The classification of the input can either be performed using template matching techniques or discriminative classifiers, which the latter has the advantage of lower computation time in classification. Overall, appearance-based approaches have shown to be robust against drawing

variations as compared to other recognisers.

2.3.3 Grammar-based Methods

These approaches exploit geometric relationships between parts of a shape to perform the classification. Some grammar-based approaches (Costagliola et al. 2003, 2005) allow shapes to be defined hierarchically by textually describing shapes in the form of a programming language. A popular multi-stroke recogniser that uses a shape description language is LADDER (Hammond & Davis 2006). The language consists of predefined shapes, constraints and some high level semantics for each domain. In the recognition process, if the rules and constraints of a shape definition meet for the given input, the recogniser returns the shape label as the classification result. LADDER has been incorporated into SketchREAD (Alvarado & Davis 2004), which uses a dynamically constructed Bayesian Network to determine how well the given input fits the shapes in the dataset. An example of a description for an open arrow with LADDER can be seen in Figure 2.6.

Grammar-based approaches are shown to be powerful for recognition of shapes by checking a number of rules and constraints. However, using these approaches slows down the sketch recognition system when the sketch contains large number of strokes (Cheema 2014). Moreover, these approaches require experts to describe diagram components, which makes them domain dependent.

2.3.4 Summary

In this section we reviewed existing shape recognisers; their shape representation and classification techniques. Feature-based methods are more suitable for single stroke recognition, as they capture a wide variety of different aspects of a stroke. The main computation time of these methods is the feature calculation, while the classification using discriminative classifiers can be computed quickly. On the other hand, the template matching systems support multi-stroke and complex shapes, although the computation time for the classification is high since they need to match the input with the templates. The appearance-based approaches that only consider the visual appearance of shapes have shown to have a high recognition rate while being insensitive to drawing variations.

2.4 Rejection

One of the challenges in sketch recognition is determining which set of strokes together form a shape candidate, which is referred as the grouping process. The majority of the work in the literature (except the clustering approaches, Section 2.2.2), perform

```
(define shape OpenArrow
  (description "An arrow with an open head")
  (components
    (Line shaft)
    (Line head1)
    (Line head2))
  (constraints
    (coincident shaft.p1 head1.p1)
    (coincident shaft.p1 head2.p1)
    (coincident head1.p1 head2.p1)
    (equal-length head1 head2)
    (acute-meet head1 shaft)
    (acute-meet shaft head2))
  (aliases
    (Point head shaft.p1)
    (Point tail shaft.p2))
  (editing
   ( (trigger (click_hold_drag shaft))
     (action
       (translate this)
       (set-cursor DRAG)
       (show-handle MOVE tail head)))
    ( (trigger (click_hold_drag head))
      (action
        (rubber-band this head tail)
        (show-handle MOVE head)
        (set-cursor DRAG)))
   ( (trigger (click_hold_drag tail))
     (action
       (rubber-band this tail head)
       (show-handle MOVE tail)
       (set-cursor DRAG))))
  (display (original-strokes)))
```

Figure 2.6: The description of an arrow in LADDER (Hammond & Davis 2006)

grouping by producing several shape candidates, among which the majority are invalid. The invalid shape candidates either contain strokes from multiple shapes or are incomplete forms of a shape. To reject such shape candidates, different methods are adopted. For example, in grammar or rule-based approaches, the shape candidates that do not conform to the defined language/rules are rejected (Alvarado & Davis 2004, 2006, Julca-Aguilar et al. 2017, Hammond & Paulson 2011). In the sequential optimisation approaches (Section 2.2.2) a combination of shape candidates that optimise a cost function are selected and the rest are rejected (Shilman et al. 2004, Shilman & Viola 2004, Feng et al. 2009). Defining rules and domain patterns requires an expert's input for each domain, and optimising a cost function can be a computationally expensive task.

The use of negative examples is a common method used for rejection (Section 2.2.1), where a classifier is trained with invalid shapes. These approaches produce a large number of shape candidates and train the recogniser with both the negative and positive examples. Since the number of negative examples is very large and there is a large variation within this negative class, these approaches cannot be used for accurate rejection of negative examples. Therefore, a final step of optimisation for selecting the final candidates is carried out. The classifier needs to learn about a very large class of negative examples, which can affect the accuracy of the classifier for the regular shapes. Due to this dataset imbalance (the very large negative examples class), either synthetic data should be generated for the positive classes (Bresler et al. 2016*a*) or the negative class needs to be separated into multiple sub-classes (Bresler et al. 2015*b*). The number of sub-classes depends on the domain and needs to be defined by experiments (for example Bresler et al. (2015*b*) used 30 and 20 classes for FC and FA datasets, respectively), which is another downside for these approaches by making them more domain dependent.

An alternative approach for rejecting negative examples is through novelty detection, where the goal is to identify objects that are not similar to the training set (Marsland 2003, Pimentel et al. 2014). To the best of our knowledge, three papers have used novelty detection in the context of sketch recognition (Arandjelović & Sezgin 2011, Tirkaz et al. 2012, Yesilbek & Sezgin 2017). As described in Section 2.2.1, Arandjelović & Sezgin (2011) use a one-class SVM classifier to reject invalid shape candidates that are generated using HMM, while Yesilbek & Sezgin (2017) explore the use of few examples for labelling sketch datasets.

Yesilbek & Sezgin (2017) have looked at ways to build recognisers with small number of examples, which include novelty detection techniques. They automatically extend a very small set of labelled examples (1-3 examples) with new examples extracted from unlabelled sketches. Since this work considers labelling the full sketch (instead of isolated shapes), it produces possible shape candidates for labelling, among which the majority

are invalid. To reject invalid shape candidates in a sketch, two steps are carried out. The first step is a conservative rejection that allows confidently discarding some of the negative examples. For this, a linear kernel SVM is trained with some positive and some negative examples of 1-3 randomly chosen sketches. The trained classifier is then used to reject those unlabelled shape candidates that are predicted as negative with a high confidence. In the second step, four methods are proposed to label some of the shape candidates to be used for training the shape classifier. Two of these approaches use the nearest neighbour classifier while one generates artificial instances instead of selecting the candidates. They also propose a context-based system that selects the shape candidates based on their appearance and their context. The context-based approach achieved the best results in the experiments.

Tirkaz et al. (2012) propose an auto-completion system while performing rejection when enough information is not provided. In this research the training dataset is extended by adding all the partial shapes that occur during drawing to be able to learn the visual appearance of partial shapes. Based on the feature representation of shapes, Constraint K-Means (CKMeans) clustering algorithm is used to enforce the separation of full shapes into different clusters. Based on the calculated posterior class probabilities for a shape candidate, the system either infers a class label or delays the classification decision until further strokes are added to the input. The experiments on Course of Action Diagrams (COAD) (Manual 1997) dataset show that when 100% accuracy for full symbol recognition is achieved, about 23% of symbols were incorrectly rejected. The results on NicIcon (Niels et al. 2008) shows 99.40% accuracy for full symbol recognition and 2.36% is the rejection rate. This should be noted this approach is only tested on isolated shapes, not on connected diagrams. In connected diagrams, apart from the valid and incomplete shapes, depending on the grouping strategy, some invalid shape candidates might also be hypothesised that need to be rejected.

Sketch recognition systems often produce many shape candidates among which, only a small portion is valid. In this section, we reviewed different methods used for rejecting invalid shape candidates. The most dominant work is training a SVM classifier with negative examples and using this to recognise invalid shape candidates, as a negative class. This method has not been efficient in rejecting the invalid shape candidates and have required further steps for selecting the valid shape candidates. We also reviewed work towards using novelty detection techniques for other applications in sketch recognition. The possibility of using novelty detection techniques for rejecting invalid shape candidates has not been well explored and seems to be a promising way forward.

Figure 2.7: Some examples of different arrow shafts in the flowchart diagram.

2.5 Connector Recognition

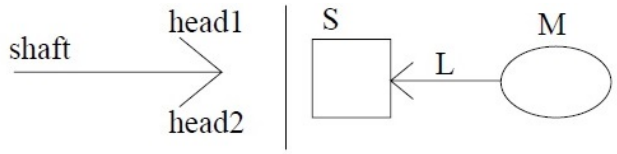
The connectors are inherently different from regular shapes by their appearance. Aside from un-directed connectors (without a head), most of the connectors in diagrams are composed of two main parts, the shaft and the head. The shaft can be drawn with any arbitrary shape, which make them difficult to recognise. For example, Figure 2.7 shows a few examples of different drawn arrow shafts in the flowchart dataset. Connector heads can also be drawn with different shapes. For example, in the work of Awal et al. (2011), the connector heads are drawn with filled circles, circles, triangles, and arrow heads. This uncertain shape of the shaft and heads makes the task of connector recognition difficult. Moreover, the connectors are spatially close to the shapes, which makes them difficult to differentiate from the regular shapes.

In this section we will cover the approaches designed specifically for connectors. Note that the clustering approaches in Section 2.2.2 perform the grouping of connectors without a separate connector grouping/recognition component. Similarly, the optimisation methods that use time-based models (Section 2.2.1) also perform grouping and recognition of connectors within the same model.

Some work treats the connectors in the same as way as regular shapes and do not have a separate connector recognition component (Bresler et al. 2013*b*, Awal et al. 2011), although these approaches have performed poorly on connectors. Shilman & Viola (2004) use the same feature representation as for regular shapes to represent synthetically generated arrows, and achieve a good recognition rate, however, the generated arrows only have a short straight line. They also point out that their system would not work with real arrows with different arrow shafts.

A series of work on connector recognition is specifically devoted to arrows using grammar and language (Costagliola et al. 2006, Brieler & Minas 2010). In these methods, a visual language is used to describe the the arrow and other shapes in a particular domain. For example, Figure 2.8 shows the description of an arrow in the family tree domain. These approaches require an expert to define the connectors and the rest of the shapes in a diagram.

In a similar manner, some rely on a geometric definition for connectors (arrows in particular). For example in the work of Kara & Stahovich (2007), arrows are required to



DEFINE ARROW
(**Subshapes**
L1,L2,L3: (Line *shaft head1 head2*))
(**Constraints**
C1: (coincident *shaft.p1 head1.p1*)
C2: (coincident *shaft.p1 head2.p2*)
C3: (equal-length *head1 head2*)
C4: (shorter *head1 shaft*)
C5: (acute-angle *head1 shaft*)
C6: (acute-angle *head2 shaft*)

DEFINE MOTHER-SON
(**Subshapes**
M,S,L: (Female *M*) (Male *S*) (Child-link *L*)
(**Constraints**
C1: (touches *L.head S*)
C2: (touches *L.tail M*))

Figure 2.8: The description of the shape "arrow" in family tree domain Alvarado & Davis (2004)

Figure 2.9: An example of a drawn arrow with the five key points matched (Kara & Stahovich 2007)

be drawn either with a single or two strokes, from tail to head. Five key points (shown in Figure 2.9) are identified based on the pen speed minima. A series of geometric tests, such as calculating the angle between points, or the length of line segments, are carried out to determine whether the input is an arrow. The work of Kara & Stahovich (2007) was extended by Stoffel et al. (2009) who used a different classification method. In this research, first the arrows are rotated to point to a specific direction, after scaling all the points, the point coordinates, angles and lengths of the segments between points are calculated and fed into a neural network to determine whether they belong to a shape or a connector. These approaches put a restriction on how the arrows should be drawn and would need further expansion to support other connector heads.

Hammond & Paulson (2011) found only four types of drawing for arrows in their testing dataset (a collection of 29 shapes or partial diagrams from different domains

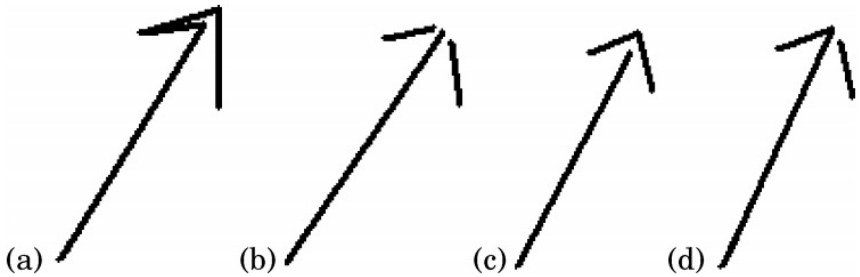


Figure 2.10: An example of a drawn arrow with the five key points (Hammond & Paulson 2011)

collected in isolation). The first case is when the whole arrow is drawn with a single stroke, and the second one is when the shaft is drawn with a single stroke and the head is drawn with two strokes (see Figure 2.10a and b). These two cases are recognised with the same approach they use for recognising regular shapes (the classifier's confidence must be above 75% for the connector to be accepted). For the other two cases, before the classification a pre-processing step is required. The third case is when the shaft is drawn with a single stroke and the head is drawn with a separate single stroke. For this case, they split the head at the corner and then recognise it in the same way as regular shapes. The fourth case is when the user draws the shaft and one half of the arrow head with a single stroke, and the second half of the arrow head with another stroke. This case is handled by separating the head part from the shaft and then performing the recognition. Although 96.3% of the arrows are recognised correctly, this work is designed for arrows that are drawn in a specific way and is not applicable for other types of connectors or other variations of arrow drawings.

Bresler et al. (2014) propose an arrow detection system that is carried out after finding all the regular shapes. The arrow detector in this system tries to find connectors between all the pairs of recognised shapes. The connector's shaft between shapes are created incrementally, by finding a stroke in the vicinity of a recognized shape and iteratively finding the strokes in the vicinity of the connector stroke and so on. This continues until the number of strokes reaches 5 and get discarded or the continuation of strokes reach another shape. For an identified arrow shaft, multiple head candidates in the vicinity of the shaft stroke's endpoints are hypothesised and the one with the highest score is selected. The score of a head is calculated by considering the confidence of the head's bounding box shape and distance of the head centroid from the corresponding endpoint of the shaft. This arrow detection approach is incorporated in their sketch recognition system and achieves 74.4% and 89.3% accuracy in correctly grouping and recognizing the arrows in the flowchart and finite automata datasets, respectively. It should be noted that for the domain of finite automata, the arrows entering the initial

state are considered as a regular shape, not a connector, since they all have a similar shape.

Bresler et al. (2015*a*) proposed an arrow head detection using relative stroke positioning. Similar to their previous work (Bresler et al. 2014), the detection of an arrow's shaft is done iteratively by simply adding strokes to a sequence such that the first stroke starts in a vicinity of the first shape and the last stroke ends in a vicinity of the second shape. Both endpoints of the shaft are considered to find the arrow head. For an arrow shaft, first a reference stroke (a sub-stroke of the shaft) and a reference point (end-point of the shaft) are identified. These points are used to express the relative positioning of the query strokes. The query strokes are all strokes in the vicinity of a given end-point of the shaft's endpoint, which are not a part of the shaft itself nor the two given shapes. The information about the relative positioning and each query stroke are given to a Bidirectional Long Short-Term Memory (BLSTM) classifier which returns the class label ('shape' or 'no-shape') and its confidence level. The sum of confidences of each arrow head candidate is calculated and the one with the highest value is selected as the arrow head. This approach has not been evaluated on connectors in isolation and has only been incorporated with their whole sketch recognition system, which carries out a final candidate selection using the optimisation technique. In the experiments, their system found 78.1% and 92.8% of the connectors in the flowchart and finite automata datasets, respectively.

In this section, we reviewed different approaches for connector grouping and recognition. We showed that the majority of the work is designed for a specific type of connector that should be drawn in a particular way. In the work of Bresler et al. (2014, 2015*a*) some possible connector candidates are generated and the best suitable ones are selected in the final optimisation process.

2.6 Summary

In this chapter we reviewed sketch recognition systems, focusing on grouping and shape recognition, rejection and connector recognition. One challenge is to determine which strokes together form a single shape. Different methods are adopted for this grouping, that either work simultaneously with a recogniser or perform the grouping independently. We reviewed the existing methods used for recognising a shape candidate. We also reviewed the different techniques used for rejecting invalid shape candidates. This area has received little attention thus far, particularly the use of novelty detection for rejection. Finally, we reviewed various work towards connector recognition and showed that the work in this area is very limited.

Chapter 3

System Design

This chapter describes the architecture of our sketch recognition system, followed by full details of the sketched diagram datasets used throughout this thesis for system development.

A sketch diagram can be composed of text, shapes and connectors. The separation of text from the rest of strokes is out of the scope of this PhD. Therefore, for our experiments throughout the development, we manually deleted the text strokes from the diagrams. For the final evaluations, we either removed the text for full comparative study or used the same divider other approaches used for a fair comparison.

The process of grouping and recognition of shapes in a diagram are tied together. Therefore, we decided to treat them as a simultaneous task. Later, this became more clear that rejection is going to be an essential part of the process. Despite the simultaneous nature of our work, we chose to design, implement and test each part separately and then combine them all together and test the whole system.

One important example of developing a part of the system in isolation that will be described shortly (in Chapter 4) is the grouper. Since the grouper relies on having a recogniser with recognition and rejection capability, we built a mock recogniser that gives 100% accuracy to be able to check the grouper's performance.

The design of each part was initially informed by the literature review. We identified areas that could be improved upon and devised and implemented methods to address these areas accordingly. We made each part of the system as good as we can by an iterative process of design, implementation, analysis, and evaluation. Each part of the system was evaluated with the development datasets (which will be described in Section 3.4) in isolation. The analysis of the results could lead to further design, development and evaluation.

Once all the parts of the system were finalised and had been evaluated in isolation, we combined them together to form the whole sketch recognition system. The whole system was then evaluated on fresh datasets for further evaluation and comparison to

other approaches in the literature. In the following, we will provide details of the system design.

3.1 System Design

In this section we illustrate the architecture of our sketch recognition system. The sketched diagrams are usually comprised of three semantically distinct strokes, which are text, shape and connector. In our sketch recognition pipeline, we assume a text divider such as that designed by Blagojevic et al. (2011) is used to separate the text strokes from the rest. Therefore, the text stroke separation, grouping and recognition is out of the scope of this PhD. Our system is comprised of two phases, the use and training phases, which is described in this section.

3.2 Training Phase

The training process shown in Figure 3.1 takes a set of labelled diagrams as input and produces the shape recogniser and the connector recogniser. The grouper is a deterministic algorithm without any trainable part; therefore, it is not included in this section. The shape recogniser is composed of two main components: the shape rejector and the shape classifier. The shape rejector is a component that only learns about valid shapes and rejects any other shape, whereas the classifier identifies which class the shape belongs to.

The connectors are also shapes. However, their appearance can vary markedly between examples, and they have requirements that are not true of general shapes, such as the need to connect shapes. We therefore treat them separately. The connector recogniser only learns about the common part of connectors, which is the head. For connectors, we only need a rejector, and not a classifier, since the class is already known. Training a classifier or a rejector requires a training set, which is obtained through a feature extractor component.

3.3 Use Phase

As shown in Figure 3.2, the use phase is comprised of two main steps: grouping and recognition of shapes, and connector recognition. The input to the system is a list of strokes drawn in a diagram. The output of the system will be a list of recognised shapes, connectors and a list of unrecognised strokes.

The shape grouper and recogniser work simultaneously to find and recognise shapes in the given diagram. The grouper is a deterministic algorithm that hypothesises shape

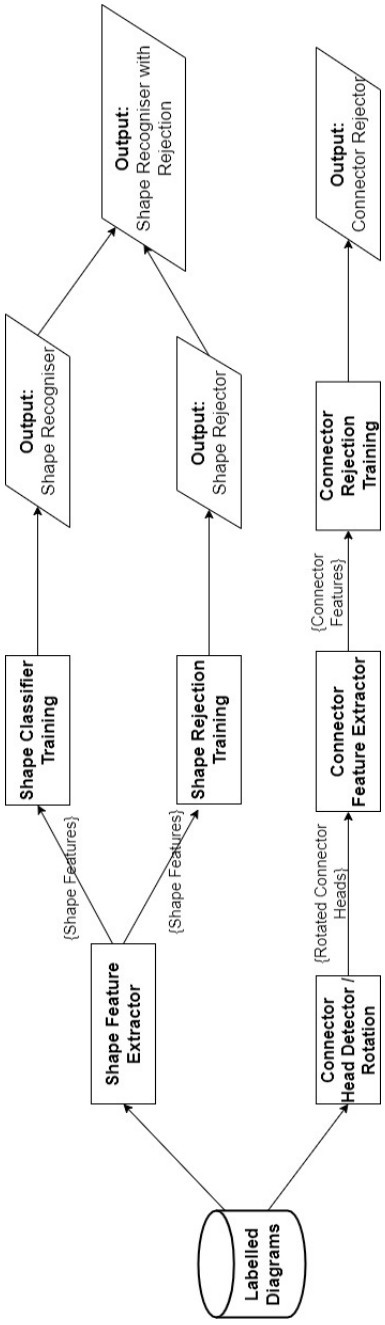


Figure 3.1: Our sketch recognition scheme (training phase)

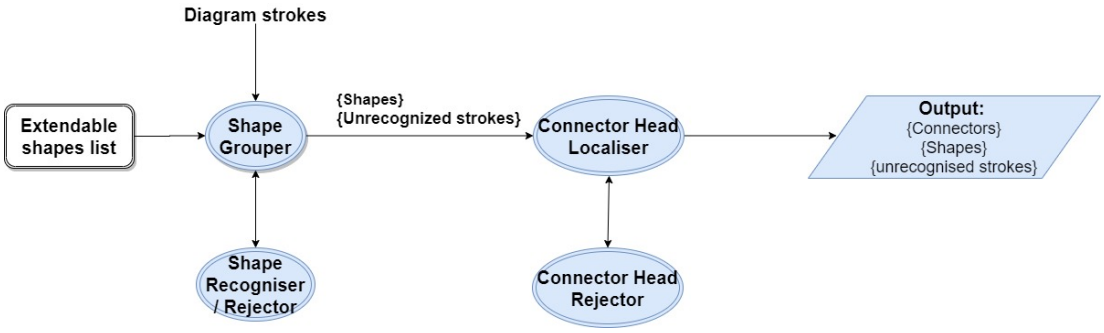


Figure 3.2: Our sketch recognition scheme (use phase)

candidates based on some heuristics. The design of such a grouping algorithm is one of our contributions in this PhD.

The shape recogniser has an embedded rejector that determines if a shape candidate is valid. The rejector uses novelty detection methods for rejection. The output of the shape grouping and recognition process is a list of recognised shapes and unrecognised strokes that are given to the connector recognition system.

The connector recognition process is to find the connectors among unrecognised strokes as elements connecting the shapes. The connector localiser hypothesises possible connector head candidates around recognised shapes and the connector rejector determines the validity of the candidates.

Diagrams often include shapes that are made of more than one shape. These shapes can be extended to other shapes in that domain. Having knowledge about which shapes are extendable is helpful information that can reduce the computation time of the grouping (this is discussed further in Chapter 4). We give this information to the system explicitly by defining a list of extendable shapes for that domain.

3.4 Datasets

We gathered different datasets, including publicly available and some from other research groups. We divided these datasets into development and evaluation datasets. The development datasets were used for training and analysis of each component during the development process; their details are provided in this section. It should be noted that the temporal order of drawing for all datasets is provided. These datasets are as the following:

- FC (Awal et al. 2011).
- FA (Bresler et al. 2014).
- Flowchart diagrams (Stevens et al. 2013).

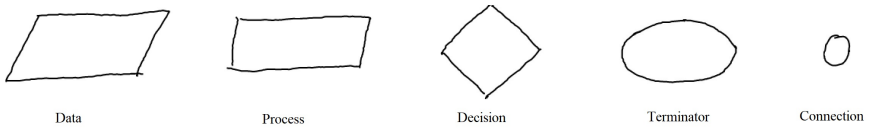


Figure 3.3: Examples of the shapes in the FC dataset (Awal et al. 2011)

- Class diagrams (Stevens et al. 2013).
- Digital circuit diagrams (Stevens et al. 2013).

Fresh datasets are used for evaluation of the whole system (referred to as the evaluation datasets); their details are provided in Chapter 7.

All the experiments for this thesis are carried out on a Surface Pro 4 (Core i7-6650U, 16GB RAM). All the code is written in C# using Visual Studio 2017.

3.4.1 FC Dataset

This dataset includes 419 sketches of flowcharts collected from 38 participants using digital pen enabled devices. This dataset is publicly available [1] [2] and is the largest dataset we have used. Participants were shown a template diagram and were asked to copy the diagram. In total there are more than 9000 shapes from 6 classes (text, arrow, data, connection, terminator, process, decision). The ground truth for the dataset is provided as well. The dataset is split into a training set (248 sketches) and test set (171 sketches). Figures 3.3 and 3.4 show an example of each class and an example of a sketch in this dataset, respectively.

After checking the test set, we realised two of the shapes are mislabelled; in one sketch a connection is mislabelled as a decision and in another sketch, an arrow is labelled as a process. The details of this dataset after changing the label for these two shapes can be seen in Table 3.1. It should be noted that the details of the dataset in the paper introducing this dataset (Awal et al. 2011) is different from their website, as the published paper has lower number of diagrams. We assume that they have expanded this dataset since the paper (Awal et al. 2011) was published.

This dataset includes text strokes as well as the shape and connector strokes. In the pipeline of our sketch recognition, before the grouping and recognition of shapes, a divider separates the text strokes from the rest. For our experiments, we have assumed a perfect divider that can accurately separate the text strokes by excluding all the text strokes from the diagrams.

[1] http://ivc.univ-nantes.fr/en/databases/Flowchart/
[2] We are using FC dataset v1.7

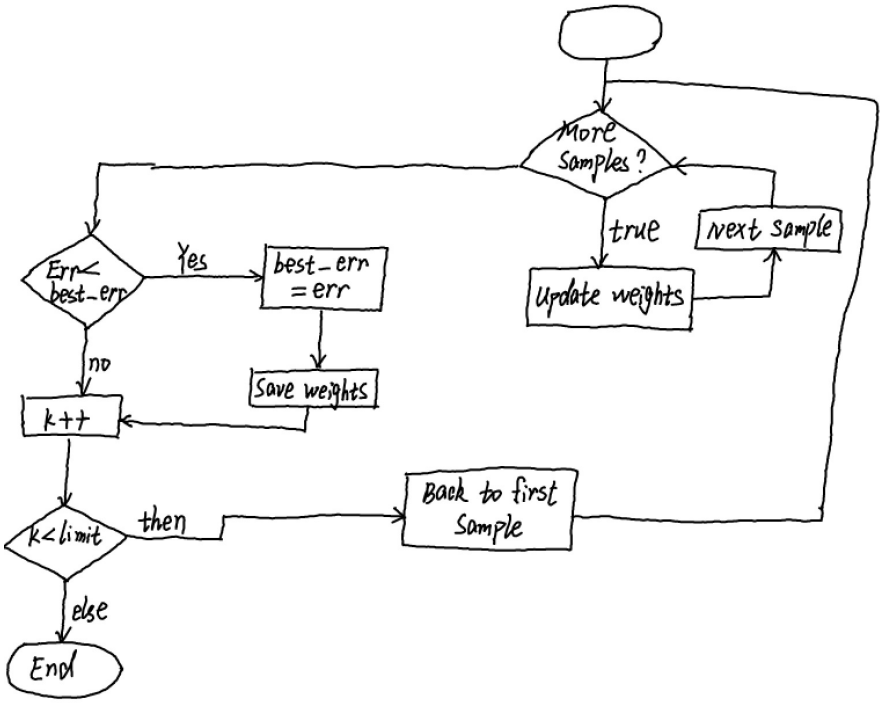


Figure 3.4: An example of a drawn sketch in the FC dataset (Awal et al. 2011)

3.4.2 FA Dataset

Another publicly available dataset is the finite automata (FA) dataset (Bresler et al. 2014)[3]. This dataset contains samples of 12 diagram patterns drawn by 25 participants, which results in 300 diagrams. The dataset is collected on a Tablet PC. The dataset is divided into training, validation and test sets. The validation set is usually used for training neural networks to keep track of how well the algorithm is doing as it learns. For our experiments, since no validation process is required during training, we combined the training and validation set to be closer to the 20/80 rule (where 80% of the data is used for training and the other 20% is used for testing). This resulted in 216 diagrams for training and 84 diagrams for testing.

This dataset is composed of two classes for regular shapes (state and final state), arrow and text. Table 3.2 shows the details of this dataset and Figures 3.5 and 3.6 show an example of a drawn sketch and an example of each class in this dataset, respectively. This dataset contains a shape hierarchy because *state* can be extendable to *final state*, which can be a challenge for the grouping and recognition process.

Similar to the FC dataset, this dataset also includes text strokes. For our experiments, we manually exclude the text strokes as we assume a perfect divider is used.

[3]We are using FA dataset v1.1

	Training set		**Test set**	
	#Shapes	**#Strokes**	**#Shapes**	**#Strokes**
Terminator	299	458	203	446
Data	416	1382	294	924
Decision	310	1055	211	695
Process	598	1828	407	1130
Connection	177	190	125	136
Arrow	1829	3882	1260	2736
Text	1911	14560	1291	9629
Total	5540	23355	3791	15696

Table 3.1: The details of the FC dataset

	Training set		**Test set**	
	#Shapes	**#Strokes**	**#Shapes**	**#Strokes**
State	720	733	284	287
Final State	342	688	129	260
Text	2833	5448	1114	2077
Arrow	2043	3980	796	1501
Total	5938	10849	2323	4125

Table 3.2: The details of the FA dataset

3.4.3 Flowchart Dataset

This second dataset of flowcharts (Stevens et al. 2013) were drawn by 20 participants on a Tablet PC. This dataset is simpler and much smaller than the FC dataset. We picked this dataset for our development due to its simplicity, which made it easier to trace and track each process of the development. Each diagram is a sketch of a flowchart diagram, comprised of three different regular shapes (start/end, process and decision) and arrows as connectors. This dataset does not include any text. Figures 3.7 and 3.8 show an example of this diagram and an example of each class, respectively. The details of this dataset can be see in Table 3.3.

	#Shapes	#Strokes
Start_end	39	47
Process	108	198
Decision	62	122
Connectors	249	512
Total Shapes	209	367
Total	458	879

Table 3.3: The details of the flowchart dataset

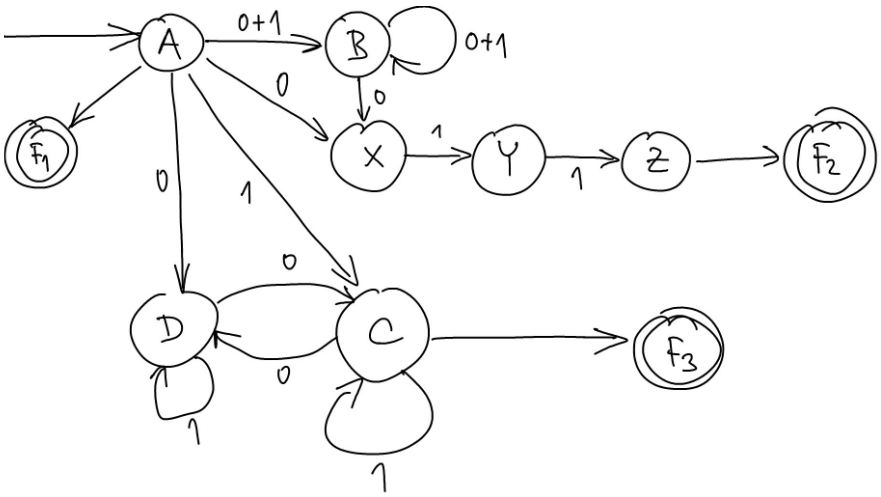


Figure 3.5: An example of a drawn sketch in the FA dataset

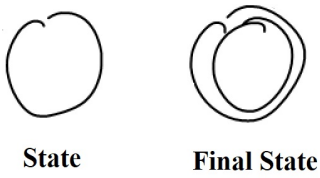


Figure 3.6: An example of the shapes in the FA dataset

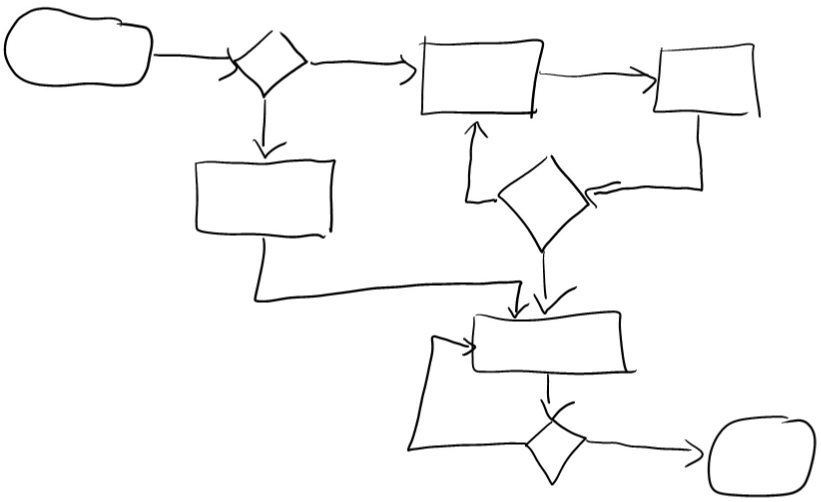


Figure 3.7: An example of a drawn sketch in the flowchart dataset

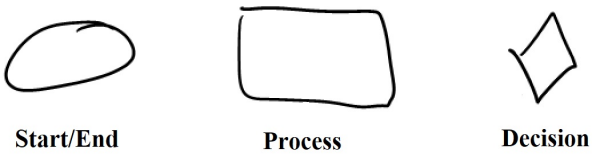


Figure 3.8: An example of the shapes in the flowchart dataset

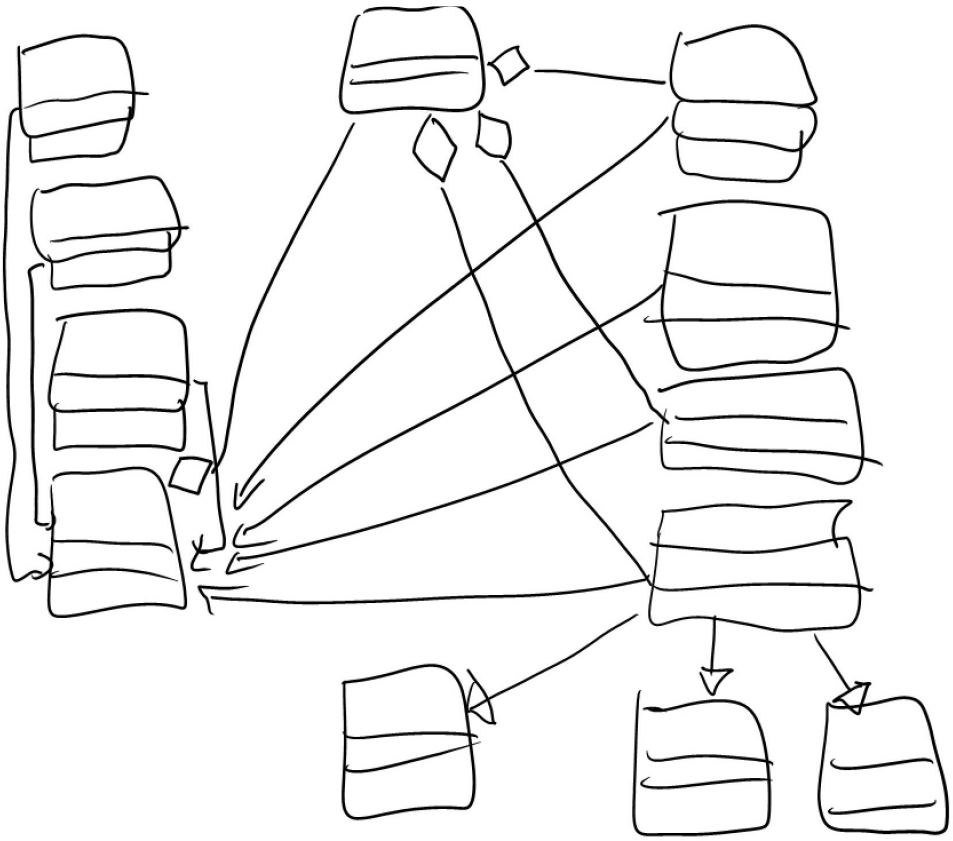


Figure 3.9: An example of a drawn sketch in the class diagram dataset

	#Shapes	#Strokes
Class	174	751
Connector	190	477
Total	364	1228

Table 3.4: The details of the class diagram dataset

3.4.4 Class Diagram Dataset

This dataset was collected by 20 participants each drawing a class diagram on a tablet PC (Stevens et al. 2013). This dataset is different from other datasets as there is only a single regular shape (the class) with arrows connecting them. Figure 3.9 shows an example of a class diagram drawn in this dataset. The details of the class diagram dataset can be see in Table 3.4. The ground truth for this dataset was already provided for the class shape and the connectors.

3.4.5 Digital Circuit Diagram Dataset

The digital circuit dataset comprises 20 sketches drawn by 20 participants on a Tablet PC (Stevens et al. 2013). The dataset includes 8 distinct shapes, which are AND, NAND, OR, NOR, XOR, XNOR, Start and NOT. One of the challenges in this domain is that some of the shapes from different classes have a high similarity to each other. For example, Figure 3.10 shows an example of each class in this diagram, where OR and AND classes are visually similar. In addition, this domain contains complex shapes (i.e. shapes that are built from other shapes). Therefore, there is a hierarchy of shapes in this domain, which is shown in Figure 3.11. For example, in Figure 3.11 it can be seen that OR gate is extendable to both NOR and XOR gates. The details of this dataset can be seen in Table 3.5. Figure 3.12 shows an example of a drawn digital circuit diagram.

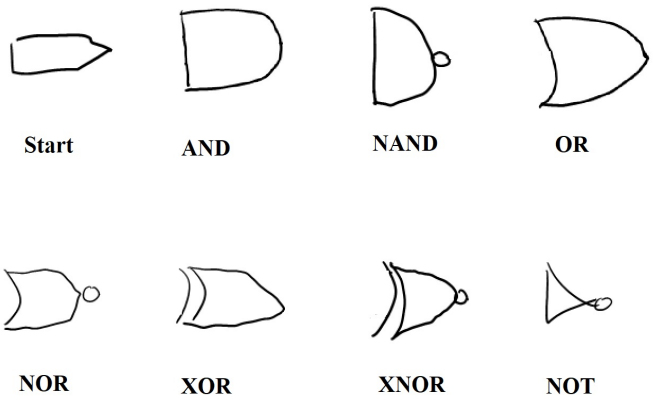


Figure 3.10: Examples of the shapes in the digital circuit dataset

	#Shapes	#Strokes
AND	21	48
NAND	21	66
OR	19	43
NOT	20	58
XOR	21	75
XNOR	18	79
NOR	22	72
Start	77	223
Connectors	207	444
Total	426	1108

Table 3.5: The details of the digital circuit diagram

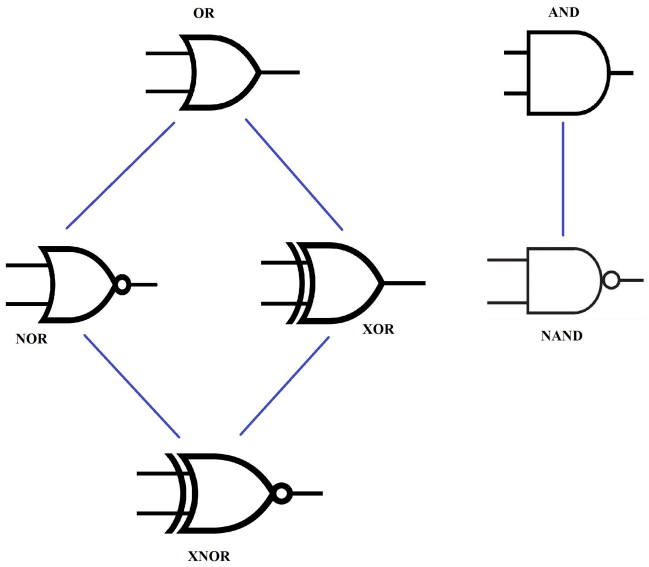


Figure 3.11: A visual representation of the shape hierarchy in the digital circuit domain

3.5 Summary

In this chapter we described the system design and the required components of our sketch recognition system. We also provided a description of the datasets we used during the development.

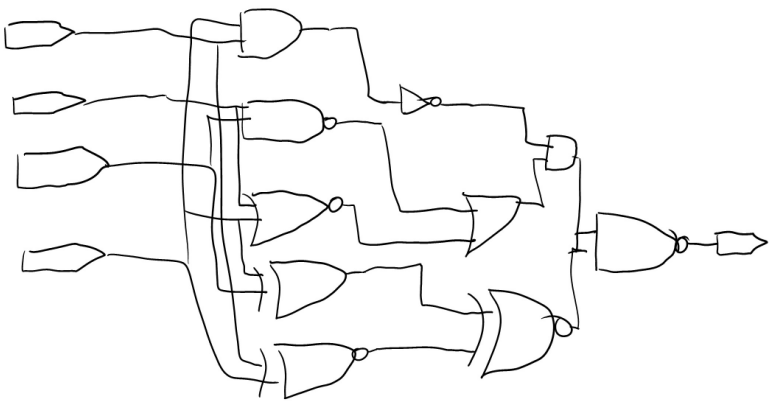


Figure 3.12: An example of a drawn sketch in the digital circuit dataset

Chapter 4

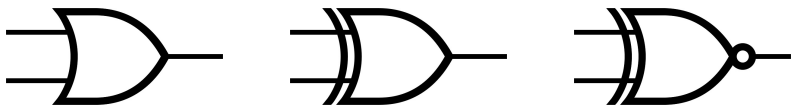
Shape Grouper

For a sketch recognition system, it is crucial to group strokes that belong to individual shapes, so that those shapes can then be recognised. Particularly in free-sketch environments, grouping of strokes is necessary to deal with multi-stroke shapes and interspersion of strokes. This chapter describes our approach towards grouping. In our system, the grouping process occurs after separating text strokes. It uses a search algorithm that uses spatial heuristics to form shape candidates from neighbouring strokes. The grouper works simultaneously with the recogniser (which will be described in detail in Chapter 5) to check the viability of the shape candidates. In this chapter we describe our grouping algorithm followed by experiments showing the performance of the grouping algorithm under different circumstances.

4.1 Neighbourhood Search-based Grouping Algorithm

Our grouping algorithm takes a bottom-up approach. The algorithm picks a single drawn stroke and incrementally explores its spatial neighbours to hypothesise different shape candidates. The algorithm starts with a small scale search to pick up shapes with a small number of strokes and increases the scale to find the shapes with a larger number of strokes. Once a shape candidate is hypothesised, it is given to a recogniser. The recogniser should classify the candidate shape if it is a valid shape and reject it otherwise, returning it to the search space. The spatial proximity of two strokes is measured by calculating the minimum Euclidean distance between all point pairs of the two strokes.

The input to the algorithm (I) is a list of strokes in temporal order of drawing and a list of extendable shapes in the domain (E, which is given to the system by the user). E contains the list of shapes in the domain that are further extendable to other shapes (e.g. OR is extendable to XOR and XNOR gates, and XOR is extendable to XNOR in the digital circuit domain, hence, OR and XOR are included in E; see Figure



(a) OR gate (b) XOR gate (c) XNOR gate

Figure 4.1: An example of the OR, XOR and XNOR gates that shows the XOR gate is extendable to the XNOR gate.

4.1). The output is a list of recognised shapes (S) and unrecognised strokes (U). The algorithm starts with the head of I as the primary stroke (P) and incrementally builds a candidate shape list (C) by exploring the neighbours of P. The candidate shape list is built by first exploring the immediate neighbours of P, and then exploring the neighbours of neighbours, and so on. The candidate shape list is expanded up to Max_{s} size, a threshold that determines the maximum number of elements in C. The algorithm starts with a small value for Max_{s} to pick up shapes with a small number of strokes and increases it when there are leftover strokes. The initial value of Max_s is set to 5 based on experiments.

Every time a new element is added, the subsets of C are checked to see if a new shape can be recognised. Expanding C up to size Max_{s} and not finding a shape in the subsets of C implies that starting with that P and exploring its Max_{s} neighbours could not result in recognising any shape, hence, the algorithm picks the next available item in I as a new P. The same process of building a candidate shape list is repeated for the new P. If a new shape is recognised in the subsets of C, based on the information about extendable shapes (E), the algorithm decides if the shape should remain in the search space or it should be moved to a list of recognised shapes (S). If the recognised shape is extendable, we keep the shape in the search space to see if it is a sub-shape of a bigger shape; otherwise, the shape is moved to S to reduce the search space.

The search process continues with selecting new primary strokes and building candidate shape lists. Once all the items in I have been used as P, the algorithm scales up the search by increasing Max_{s} to check larger candidate shape lists. This process continues until Max_{s} either reaches an Upper Bound (UB) threshold or is equal to the size of I, which ensures that all possible combinations of items in I are checked if required. Another stopping condition for the algorithm is that I is empty, meaning that all the drawn shapes are unextendable and have been moved to S. The pseudocode of the grouping algorithm can be seen in Algorithm 1, and Figure 4.2 shows the flowchart of the grouping process.

Figure 4.4 shows an example of a sketch with two shapes connected with a line; and the initial values of I, S, P and C. The numbers show the order of drawing of the strokes for these shapes. In this example it is assumed that the recogniser can classify strokes

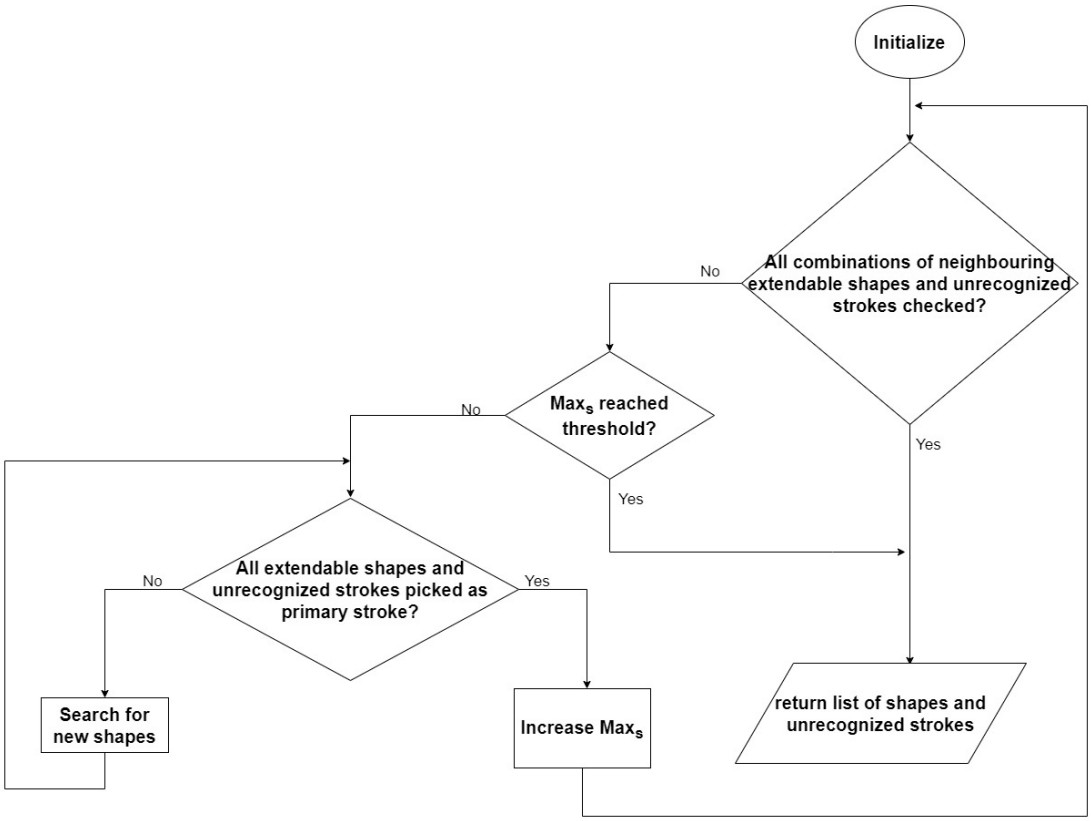


Figure 4.2: Initial Grouper Flowchart

2,3 as an OR gate, strokes 1,2,3 as an XOR gate, strokes 2,3,4 as a NOR gate, strokes 1,2,3,4 as an XNOR gate and strokes 6,7 as an AND gate; all other combinations are assumed to be rejected by the recogniser. Therefore, the list of extendable shapes are {AND, OR, XOR, NOR}. Figure 4.3 shows a visual representation of the extendable shapes in the digital circuit domain, where edges show which shapes are extendable to which (e.g. on the right it shows that AND is extendable to NAND).

4.1.1 Walk Through of the Algorithm

Figure 4.5 shows the first three steps of the algorithm when stroke 1 is selected as P and strokes 2,3 are added to C as the neighbouring strokes. Every time a new stroke is added to C all subsets are checked in descending order of size to see if a shape can be recognised. In step 3, when the subsets of C are being checked, strokes 1,2,3 are recognised as XOR gate. Since an XOR gate is an extendable shape, it is kept in the candidate list C. The strokes of the XOR gate are merged and treated as a single element from here on.

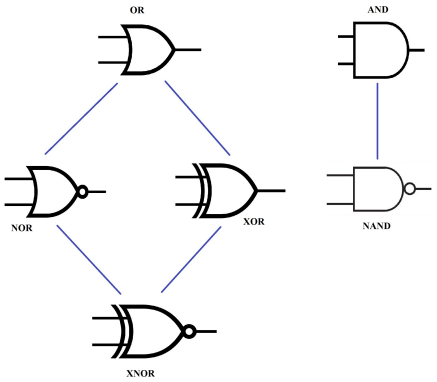
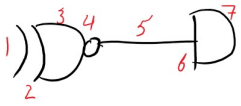


Figure 4.3: A visual representation of the extendable shapes in the digital circuit domain



I = {1,2,3,4,5,6,7}
S = {}
P = null
C = {}

Figure 4.4: A Sample of Sketched shapes

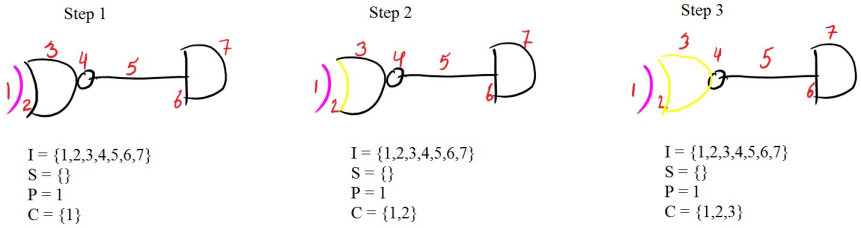


Figure 4.5: Steps 1 to 3 of the algorithm

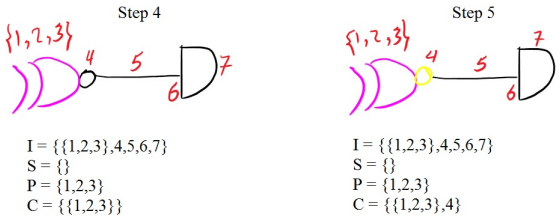


Figure 4.6: Steps 4 and 5 of the algorithm

In step 4 shown in Figure 4.6, since stroke 1 (the current P) is part of element {1,2,3}, P and C are updated to reflect the recognised shape. In step 5, stroke 4 is added to C as the neighbour of {1,2,3}. When checking the subsets of C, an XNOR

gate with strokes 1,2,3,4 is recognised. XNOR gates are not extendable shapes, therefore we have successfully grouped and recognised a complete shape. The recognised shape strokes are moved to S; these strokes are no longer part of the search space.

Next, in Step 6 (shown in Figure 4.7), stroke 5 is picked as the primary stroke. In step 7, stroke 6 is added to C as the neighbour of P and since no shape can be recognised in the subsets of C, in step 8, stroke 7 is added to C as the neighbour of stroke 6. The AND gate with strokes 6, 7 is recognised in the subsets of C and since it is an extendable shape, it is kept in the search space.

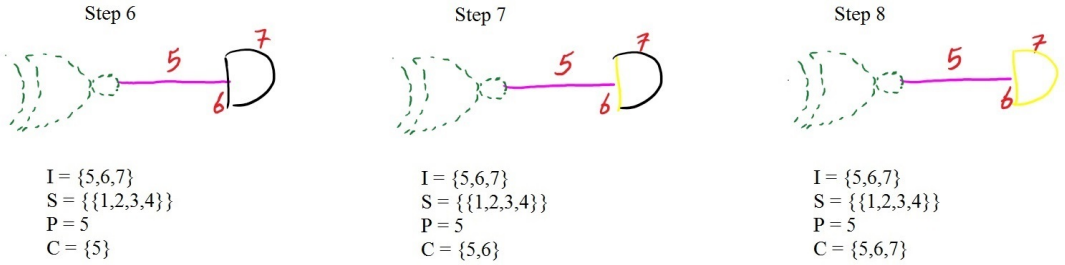


Figure 4.7: Steps 6 to 8 of the algorithm

In step 9 (shown in Figure 4.8), the I and C lists are updated with the recognised shape. Next, in step 10 (Figure 4.8), element {6,7} is picked as the primary element and in step 11, stroke 5 is added to C as a neighbour. At this point, all the elements in I have been picked as the primary element and all the combinations of items in I have been checked; hence, the algorithm stops here and adds the {6,7} to S. The grouper's output (Figure 4.9), which is a list of unrecognised strokes (I) and recognised shapes (S) (i.e. the XNOR and the AND gates) is given to the connector recogniser.

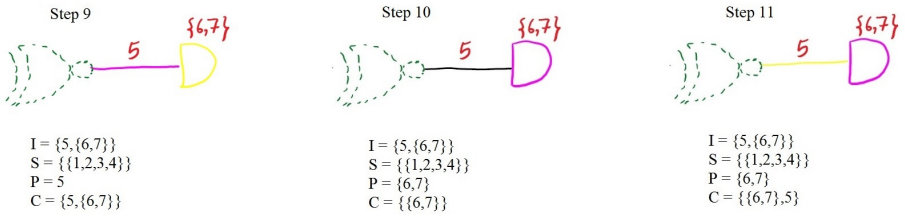


Figure 4.8: Steps 9 to 11 of the algorithm

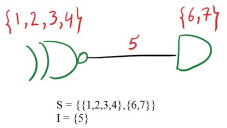


Figure 4.9: Grouper's output

Algorithm 1 Neighbourhood Search-based Grouping Algorithm(NS_Grouper)

Input: I , E {I: List of strokes in temporal order of drawing, E: List of extendable shapes in the domain}

$Max_{\mathrm{s}} = 5$ ▷ {Maximum number of strokes in a shape},

Max_{dist} ▷ {Maximum distance threshold between points},

$UB = 14$ ▷ {UB : Upper Bound value for Max_{s}}

Output: U , S {S: List of shapes, U: List of unrecognized strokes}

$S = \varnothing$, $C = \varnothing$ ▷ {C : Candidate shape list}
while (($\mathrm{Max_s} \leq |I|$) && ($\mathrm{Max_s} < \mathrm{UB}$)) **do**
    $J = I$
    loop over elements of J
        $C = J[0]$
        $J = J \setminus C$
        $foundUnextendable = false$ ▷ {$foundUnextendable$: flag for unextendable shapes}
        while (($|C| < Max_{\mathrm{s}}$) && (elements of C have neighbours in I) && (!$foundUnextendable$)) **do**
            N = neighbours of C in I
            $n = N[0]$
            $N = N \setminus n$
            $C = \{C\} \bigcup n$
            while (all subsets of C are not checked) **do**
                get next subset $\{c\}$ of C containing n in descending order of size
                if ($\{c\}$ is a shape) **then**
                    if ($\{c\}$ is extendable) **then**
                        $s = \{c\}$ as a single element
                        $I = I \setminus \{c\} \bigcup s$
                        $J = J \setminus \{c\} \bigcup s$
                        $C = C \setminus \{c\} \bigcup s$
                    else
                        $I = I \setminus \{c\}$
                        $S = \{S\} \bigcup \{c\}$
                        $foundUnextendable = true$
                    end if
                end if
            end while
        end while
    end loop
    if (no new shape is found) **then**
        $Max_{\mathrm{s}} = Max_{\mathrm{s}} + 1$
    end if
end while
$R = \{$recognized shapes in $I\}$
$U = I \setminus R$
$S = S \bigcup R$
Return S , U

Computation optimisations

In this algorithm, in order to avoid re-computing the proximity of strokes, we use an adjacency bit matrix(A) that is initialised as a pre-processing step. The size of the matrix is $n \times n$, where n is the number of strokes. This matrix stores the information about the proximity of strokes; each pair of strokes is represented either by 0 or 1. If two strokes are neighbours (based on the proximity of points of the two strokes), the corresponding cell in the matrix is set to 1 otherwise it is 0. The diagonal entries of the matrix represent whether the item is a recognised shape. If a diagonal entry is set to 1, it implies that it is a recognised shape. Since the matrix is symmetric we only need the upper triangle of the matrix. In the grouping process, when a new shape is recognised, we update the matrix by performing a bitwise OR operation on the corresponding rows and columns to merge them together. In the updated A matrix, a stroke is neighbour to a shape if it is neighbour to at least one of the strokes constructing that shape. When a new shape is recognised, in addition to merging corresponding rows and columns, we update the diagonal value to indicate that this shape is recognised. The same idea applies to single stroke shapes. If a shape is drawn with a single stroke, as no bitwise operation is needed, only the corresponding diagonal value in the matrix is updated. Figure 4.10a shows the initial value of the A matrix for the previous example, and Figure 4.10b shows the A matrix when the XOR gate is recognised. Figures 4.10c and 4.10d show the A matrix when the XNOR and AND gates are recognised, respectively.

In this algorithm, when a new item is added to the candidate shape list(C), in order to speed up the process, we avoid checking the combinations that have already been checked and only check the combinations that include the new item. As an example, in step 2 in Figure 4.5, C contains strokes 1 and 2. In step 3, when stroke 3 is added to C, the algorithm only checks the subsets of C that contain stroke 3: {3}, {1,3}, {2,3}, {1,2,3}. We also use memoization to avoid checking entire candidate shape lists that have been checked before. As an example, the candidate shape list in step 11 is the same as the candidate shape list in step 9. Therefore, in step 11 we do not check the subsets of C. In addition, we also store all the inputs given to the recogniser with the result that the recogniser gives back for that input. Next time, if the recogniser is being called with the same input, in order to reduce the recogniser calls, we get the results of that input from memory.

The aim of the mentioned optimisation techniques (using memoization, avoid checking previously examined C and using memory to store the recogniser calls) is to reduce the number of times the grouper needs to consult with the recogniser. Table 4.1 shows the number of valid shapes as well as the number of generated shape candidates with and without these optimisation techniques. As can be seen in Table 4.1, using such

	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0
2	1	0	1	0	0	0	0
3	1	1	0	1	0	0	0
4	0	0	1	0	1	0	0
5	0	0	0	1	0	1	0
6	0	0	0	0	1	0	1
7	0	0	0	0	0	1	0

(a)

	1,2,3	4	5	6	7
1,2,3	1	1	0	0	0
4	1	0	1	0	0
5	0	1	0	1	0
6	0	0	1	0	1
7	0	0	0	1	0

(b)

	1,2,3,4	5	6	7
1,2,3,4	1	1	0	0
5	1	0	1	0
6	0	1	0	1
7	0	0	1	0

(c)

	1,2,3,4	5	6,7
1,2,3,4	1	1	0
5	1	0	1
6,7	0	1	1

(d)

Figure 4.10: Adjacency matrix (A) in different stages of the grouping process when new shapes are recognised

	#Valid	#Invalid	#Invalid (optimised)
Flowchart	210	1259	1236
Class	171	6707	5909
Digital	247	17745	15445
FC	1225	40693	34747
FA	671	65911	54997

Table 4.1: The details of generated shape candidates using the grouper and a mock recognizer with out without optimisation techniques

optimisation techniques reduce the number of generated shape candidates.

4.2 Experiments

In this section we evaluate the performance of the proposed grouping algorithm on the five development datasets. The aim of these experiments is to measure the grouper's performance in isolation, without the recogniser affecting the results. To achieve this, we use a mock recogniser that works perfectly, i.e. can recognise and reject shape candidates with 100% accuracy based on the ground truth. We will perform the evaluation and comparison of our full system with a real recogniser in the Evaluation chapter (Chapter 7). We should also note that the connectors are also being rejected by the

mock recogniser, as this part of the system deals with shapes only. We deal with the connectors after the regular shapes are recognised, which will be described in Chapter 6.

There are two main parameters that can affect the grouper's performance in terms of computation time and accuracy. One is the Upper Bound (UB) and the other is the information about extendable shapes (which we refer to as the shape hierarchy in the experiments). The UB is a threshold for Max_s that stops the algorithm after some iterations. A value for this parameter that is too high can lead to a higher run time, whereas selecting a value that is too low would reduce the accuracy of the grouper. The information on extendable shapes helps the grouper to reduce the search space by removing the unextendable shapes once they are recognised, which leads to a lower computation time.

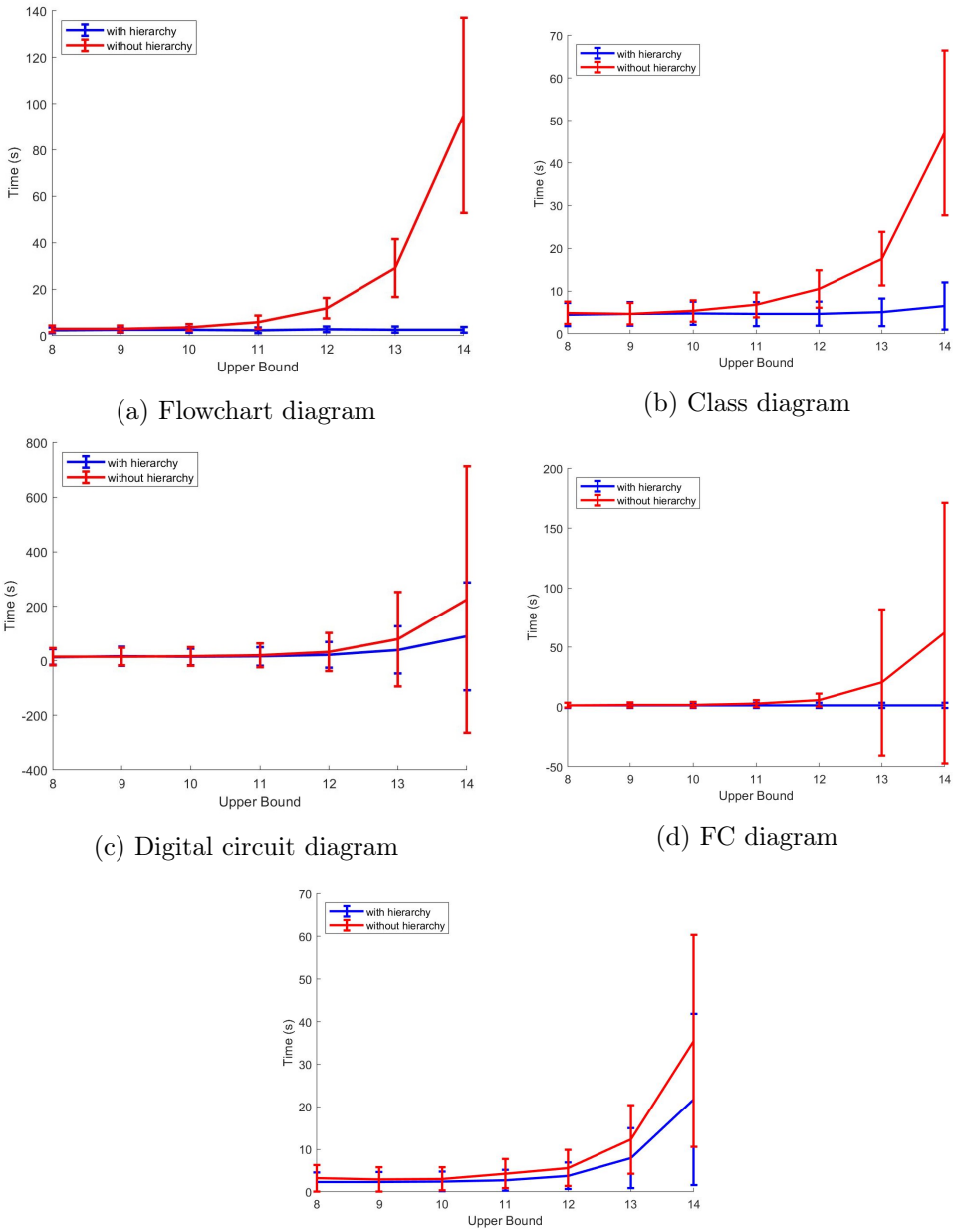
We measured the computation time from the moment the diagram is given to the grouper until it returns the output. We ran each experiment 10 times and reported the average of computation time. The distance threshold between the points of two strokes is set to 600 himetric units [1]. We chose the threshold value empirically.

In order to evaluate the performance of the grouper, we used different Upper Bound (UB) values to see its effect on the computation time and accuracy. We also checked the grouper's performance for two cases, when an extendable shape list is provided (with hierarchy) and when this information is not provided (without hierarchy). In the case of "without hierarchy" all shapes are assumed to be extendable and remain in the search space once recognised.

Figures 4.11a to 4.11e show how different values of UB affect the grouper's computation time, with and without shape hierarchy information. As can be seen in Figures 4.11a, 4.11b and 4.11d, the computation time of the grouper is almost constant for flowchart, class diagram and FC when the shape hierarchy (the extendable shape list) information is provided. However, as can be seen in Figures 4.11c and 4.11e, the computation time of the grouper for digital circuit diagram and FA increases when a higher value of the UB is selected. This is due to the fact that about 40% of the drawn shapes in digital circuit diagram and 52% of the drawn shapes in FA dataset are extendable and remain in the search space when they are recognised.

Table 4.2 shows the results of paired t-tests with and without shape hierarchy for different UB values. The results show that when higher values of UB are selected (11 or higher for flowchart, class and FA, 12 or higher for FC and digital circuit) and information about extendable shapes is provided, the computation time is significantly lower than the case that information about extendable shapes is not provided.

[1] Himetric units are a measure of length, where one himetric is equal to 10 μm



(a) Flowchart diagram

(b) Class diagram

(c) Digital circuit diagram

(d) FC diagram

(e) FA diagram

Figure 4.11: The average and standard deviation computation time (seconds) of the grouper with different upper bound values for the development datsets.

		8	9	10	11	12	13	14
Flowchart	t-value	0	0	0	1	1	1	1
Diagram	p-value	0.63	0.33	0.24	<0.05	<0.05	<0.05	<0.05
Class	t-value	0	0	0	1	1	1	1
Diagram	p-value	0.7	0.39	0.25	<0.05	<0.05	<0.05	<0.05
Digital	t-value	0	0	0	0	1	1	1
Circuits	p-value	0.7	0.31	0.38	0.08	<0.05	<0.05	<0.05
FC	t-value	0	0	0	0	1	1	1
	p-value	0.74	0.4	0.31	0.11	<0.05	<0.05	<0.05
FA	t-value	0	0	0	1	1	1	1
	p-value	0.63	0.16	0.11	<0.05	0.06	<0.05	<0.05

Table 4.2: t-test results (with confidence level of 0.05) of the grouper's computation time with and without extendable shape list for different upper bound values ranging from 8 to 14

Figure 4.12 shows how the accuracy is affected by different UB values. We measure the accuracy by dividing the number of correctly recognised shapes over the all shapes in that sketch. The reason that the results in Figure 4.12 are very good (almost 100%) is that a perfect mock recogniser is used for recognition and rejection to demonstrate the grouper's performance only. In Chapter 7 we will report the results with a real recogniser. As can be seen in Figure 4.12, there is a trade-off between the accuracy and computation time of the grouper. If a higher value of the UB is selected, the computation time increases and so does the accuracy because the algorithm checks for more combinations of strokes to find shapes. This is a sensitive parameter that needs to be chosen carefully. It should be noted that the information about extendable shapes only affects the computation time and does not affect the accuracy. This is because removing recognised unextendable shapes from the search space helps the grouper reach the stopping condition faster, but does not change the recognition results.

The results show that for the domains that only include unextendable shapes (flowchart, FC and class diagram), the grouper has a constant computation time when this information is provided to the system. In the case of the domains with extendable shapes (e.g. digital circuits or FA), the information about extendable shapes can reduce the computation time. For such domains, the UB becomes a sensitive parameter that controls the trade-off between the computation time and the accuracy. A higher value of UB can lead to a higher accuracy but at the cost of higher computation time. In all cases, if the value of UB is not too low, the grouper can find and group all the shapes with a mock recogniser. For example, for flowchart, class diagram, digital circuit, FC and FA the UB value of 8, 11, 9, 11 and 9 respectively, allows the grouper to find

and group all the shapes in the diagram. Based on the aforementioned results of the development datasets, we choose an UB value of 11 for the evaluation datasets, which is the value that we will be using for new datasets.

Setting the UB threshold for a new dataset is not simple. If the value is too low, some shapes might be missed and if is selected too high, the computation time would increase. There is always a trade-off between the accuracy and computation time for selecting the value of UB. Since on the development datasets the value 11 was sufficient to find and recognise all the shapes, we will use the same value for new datasets as well. However, it would not guarantee to find all the shapes for a new dataset. Therefore, for a new dataset, the same experiments should be carried out to find a reliable UB value.

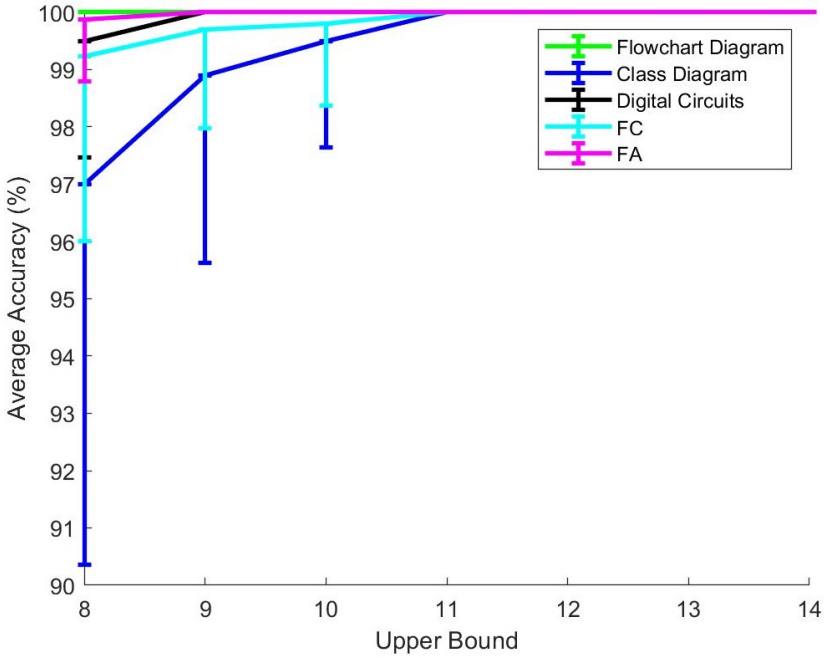


Figure 4.12: The average and standard deviation of accuracy for different upper bound values for flowchart, class diagram, digital circuits, FC and FA datasets. The accuracy is calculated by dividing the number of correctly grouped and recognised shapes over the all shapes in the dataset.

4.3 Summary

In this chapter we described our grouping algorithm, which is responsible for putting the strokes of a shape together. The grouper is a neighbourhood search-based algorithm that uses spatial information to hypothesise multiple shape candidates. We used memoization to minimise the number of times the grouper needs to consult the recogniser. The shape candidates are given to a recogniser capable of rejecting invalid ones and

accepting the valid ones. In the experiments we used a mock recogniser that can perfectly reject and recognise the shape candidates. The results of the experiments show that if the parameters of the grouper are selected carefully, the grouper can accurately group all the shapes in a diagram. We also illustrated that the information about extendable shapes can reduce the computation time of the algorithm. In the next chapter we will describe the recogniser we chose for recognition and how we equipped that with rejection capability.

Chapter 5

Shape Recognition and Rejection

In our sketch recognition system, the grouper hypothesises several shape candidates, among which the majority are invalid; i.e. either incomplete shapes or a combination of different shapes. The grouper is designed to work together with a recogniser, to identify valid shapes, and reject invalid shapes. Accurate rejection of invalid shape candidates is an essential component of our sketch recognition system. Existing recognisers e.g. (Chang et al. 2012, Lee et al. 2007, Alvarado & Davis 2004, Anthony & Wobbrock 2012, Ouyang & Davis 2009*b*) do not have the capability to accurately reject invalid shape candidates. In this chapter, we provide details of the shape recogniser we used, describe how we added rejection capability to this recogniser, and present experiments showing the performance of our rejection methods.

5.1 Shape Recognition

Our review of existing recognisers, in Section 2.3, found that appearance-based techniques were better in handling drawing variations, than other shape recognition approaches, while achieving high accuracy in recognition. In addition, these recognisers support multi-stroke shapes, which is essential to providing a free-sketch environment. One such recogniser is proposed by Ouyang & Davis (2009*b*). We chose to use this recogniser in our system for three main reasons: it is domain independent, it achieved high accuracy on the tested datasets, and it supports recognition of multi-stroke shapes.

The Ouyang & Davis (2009*b*) recogniser uses directional features to represent the visual appearance of the shapes. It is relatively simple to implement, and has the benefit of being able to visualise shape candidate features. The feature representation of an input shape is done in multiple steps: normalisation, feature representation, smoothing and downsampling. In this section, we describe the recognizer in detail and present experiments testing it's performance with other feature sets.

(a) Feature representation of a drawn shape and the corresponding feature images of different reference angles

(b) Feature representation after smoothing

Figure 5.1: Feature representation of a drawn shape before and after smoothing

5.1.1 Normalisation

A pre-processing normalisation step is carried out to make the recogniser invariant to sampling rate, scale and translation. In pen-enabled devices, strokes are sampled at a constant temporal frequency, which results in a higher density of points at corners or any part of the stroke where the pen speed is low. To make feature extraction independent of drawing speed, strokes are resampled to a constant spatial frequency. The paper (Ouyang & Davis 2009*b*) does not mention the sampling frequency used. We used the method introduced by Vatavu et al. (2012), which equalises the distance between stroke points. After the resampling process, the centre of the shape's mass is translated to the origin and then scaled horizontally and vertically.

5.1.2 Feature Representation

In this approach, each sequence of strokes (making a shape candidate) are converted into five low resolution feature images; four for the stroke orientations and one for stroke endpoints. The four orientation features correspond to the four reference angles 0, 45, 90 and 135 which measures how horizontal, vertical and diagonal the stroke is. The feature values are calculated by measuring how different the stroke points are from the reference vector, varying linearly between 0 (being different by more than 45°) and 1 (being exactly the same). The endpoint feature identifies strokes' endpoints. The feature values of the endpoint image are set to 1 at strokes' endpoints and 0 elsewhere. Each of these sequences of features for each angle are then converted into 24×24 feature grids. These grids can be considered as feature images in which the intensity of a pixel is determined by the maximum feature value of the sample points that fall within its cell. In Figure 5.1a we show the five feature representation of a drawn shape (the shape is S, which is drawn with a single stroke, shown on the left).

5.1.3 Smoothing and Downsampling

In this stage each feature image is smoothed and downsampled to increase the tolerance to local shifts and distortions. First a Gaussian smoothing function is applied to each image to spread the feature values to the neighbouring pixels. In the paper it is not mentioned what parameters are used for the Gaussian smoothing function. In our experiments, a kernel size of 3 with $\sigma = 1$ yielded the best accuracy on the tested datasets. Each smoothed feature image is then downsampled by the factor of 2 using a MAX filter, where each pixel in the downsized image is the maximum of the four corresponding pixels in the original. The downsampling process results in five 12×12 feature images, which are turned into a feature vector with 720 dimensions by rasterisation. Figure 5.1b shows the five feature representation images of the shape in Figure 5.1a after the smoothing process.

5.1.4 Classification

Ouyang and Davis also propose a novel Image Deformation Model (IDM) for classification. IDM measures the distance of the input shape to all the templates in the training set. It allows a shift in a 3×3 window for each point in the input image to form the best match to the template image to make it more robust to local shifts and distortions. The IDM distance between the shape candidate I_1 and the template shape I_2 is calculated as the following:

$$D^2 = \sum_{x,y} \min_{d_\mathrm{x}, d_\mathrm{y}} ||I_1(x + d_x, y + d_y) - I_2(x + y)||^2 \tag{5.1}$$

Where I_i represents the $3 \times 3 \times 5$ feature values in I_i centred at x, y, and d_x and d_y represent the pixel shifts.

Since matching the input with all the templates in the training set is an expensive task, a coarse candidate pruning method is used to speed up the process of classification. The idea is to be able to find the K-Nearest Neighbours (K-NN) of the input using Euclidean distance and then perform the IDM matching with the selected K-NNs. To find the K-NNs efficiently, an agglomerative hierarchical clustering algorithm is applied to each class to form a tree structure having the largest cluster at the top and progressively smaller sub-clusters below. Each cluster is then represented by the cluster centre and a radius which is determined by the farthest instance to the cluster centre. During the inference process, the algorithm starts with the top level of the hierarchy, keeping track of the best K matching scores and discarding clusters that cannot improve these scores. The IDM matching is then carried out on these K-NNs having the best match as the result of the classification. It should also be noted that the Euclidean distances are calculated on the first 128 principal components to improve the speed.

	SVM RBF	SVM Linear	IDM	Hausdorff
Theirs	95.0%	92.3%	95.2%	93.0%
Our implementation	97.9%	99.0%	95.6%	96.6%

Table 5.1: The comparison of our implementation result with (Ouyang & Davis 2009*b*) on the HHReco dataset

In their experiments, they have used other classification methods such as L_2 distance between the raw intensity images, L_2 distance between the five feature images, a modified Hausdorff distance (Kara & Stahovich 2005) and SVM classifier with Linear or Radial Basis Function (RBF) kernels. In their experiments, SVM RBF and IDM achieved the best results.

We evaluated our implementation of Ouyang & Davis (2009*b*) recogniser on HHReco (Hse & Newton 2004) (the only publicly available dataset we could access) and compared our results with the reported results in the paper (see Table 5.1). We performed this comparison to verify our implementation of Ouyang & Davis (2009*b*) recogniser. In our implementation, the optimised version of IDM that uses the pruning technique did not work well. We assume the reported results for IDM are on the non-optimised version of the IDM. In the paper, neither the implantation of the SVM nor the parameters of the kernel functions (either for linear or RBF kernel) are reported. In our experiments, we used the LibSVMsharp implementation of the SVM (*LibSVMsharp* 2019) (a .NET Wrapper for LibSVM (Chang & Lin 2011)), and achieved the best results with a linear kernel with $C = 3$ and for an RBF kernel with $C = 3$ and $\gamma = 0.5$. Our results with Hausdorff distance are slightly better than the reported results in the paper. We assume this is because we might have used different parameters in different steps such as the Gaussian smoothing function.

5.1.5 Experiments on Development Datasets with Different Feature Representations

We evaluated the recogniser's performance on the development datasets (except for the class diagram which only has one shape class) to examine the performance of the recogniser for different domains. As can be seen in Table 5.2 (the first row), the recogniser correctly classifies 98.4% to 100% of the shapes for FC, FA and flowchart datasets. However, the results for the digital circuit is lower compared to the rest (90.4%). This is because the appearance of the different shapes in the digital circuit domain are very similar. We also modified the feature representation in different ways for possible accuracy improvement, which will be discussed shortly.

Experiments with Different Feature Representations

We also carried out experiments with different feature representations. We carried out these experiments on the development datasets as well as the HHReco dataset that was used earlier. The different feature combinations we experimented with are listed below, and will be discussed in more details in the following:

- Adding speed feature image
- Removing the endpoint feature image
- Merging all the feature images
- Adding stroke level features

Johnston & Alvarado (2013) added a sixth feature image, which captures the speed of drawing. In their experiments, it shows that for the domain of digital circuit diagrams, the accuracy of classification increased from 90.58% to 92.02%. However, in our experiments, adding this feature image did not improve the classification accuracy (see the second row in Table 5.2). It should also be noted that for the FC dataset, speed information is not provided, hence, we could not report the results for this dataset.

The four feature images corresponding to the four reference angles provide a visual representation of the drawn shape. However, the endpoint feature image only provides the information about the endpoints of the drawn shape. This information can vary for different instances of the same shape. For example, a square that is drawn with four strokes has different endpoint information than the one that is drawn with a single stroke. The location of pen down and pen up can also vary for the instances of the same shape, resulting in different endpoint feature images. Therefore, we carried out another experiment to exclude this endpoint feature image to evaluate its affect on the classification result. As can be seen in the third row of Table 5.2, the exclusion of endpoint feature image did not improve the classification accuracy.

In another experiment we merged all the feature images (to reduce the feature dimensions to see if we achieve better results) by capturing the maximum value of the five feature images for each pixel, resulting in a 12×12 feature image. Since some information is being lost during this merging, we also captured the average of each feature image and the number of non-zero pixels. This resulted in a feature vector of length 154 (as opposed to the original 720 feature vector). As can be seen in the forth row of Table 5.2, this feature representation also did not improve the classification accuracy.

For all the aforementioned experiments, we achieved the best results using an SVM with a linear kernel function with $C = 3$. However, for the merged feature image

	FC	FA	Flowchart	Digital Circuit	HHReco
Original	98.4%	100%	99.5%	90.4%	99.0%
Speed	-	100%	99.0%	90.0%	98.9%
Without endpoint	97.7%	100%	99.0%	90.4%	98.5%
Merged	95.8%	100%	99.0%	86.8%	98.4%
Ink features	99.2%	100%	100%	92.7%	99.1%

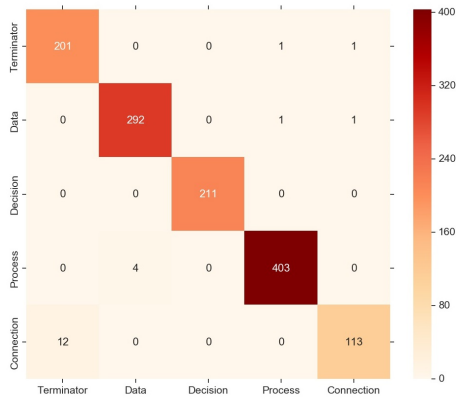
Table 5.2: Comparison of recogniser accuracy with different feature modifications on development datasets and HHReco dataset

representation, an SVM with an RBF kernel function with $C = 3$ and $\gamma = 0.5$ achieved the best results. We identified these parameter values by performing a grid search.

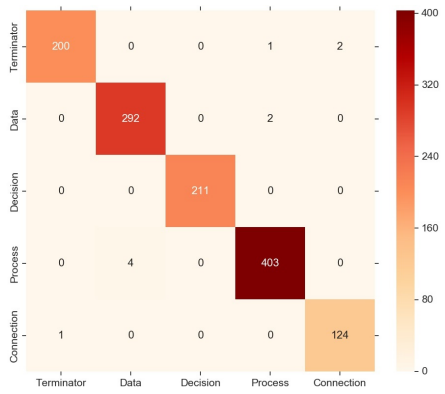
In addition to the aforementioned modifications to the feature images, we added three features in a stroke level that increased the classification accuracy (see the last row in Table 5.2 for the accuracy results). These features that are chosen by trial and error are as the following:

- **Group ink density (Ouyang & Davis 2009*a*):** The total length of the strokes in the group divided by the diagonal length.
- **Perimeter efficiency (Leung & Chen 2002):** Strokes' convex hull area divided by strokes' convex hull perimeter.
- **Thinness Ratio (Fonseca et al. 2002):** Perimeter of strokes' convex hull divided by area of strokes' convex hull

The confusion matrix is a common method to check how well a classifier performs for each class of the test set (Marsland 2015). Figures 5.2 to 5.5 show the confusion matrix of the recogniser with the original feature set, compared to the case that ink features are added for the FC, flowchart, digital circuit and HHReco datasets. Since the FA dataset has 100% accuracy in all cases, we did not include its confusion matrix. Figure 5.2 shows that for the FC dataset, adding stroke level features reduces the mis-classification of the *connection* class. Except for the one extra mis-classification for the terminator class, the accuracy of the recogniser for the rest of the shapes remains the same for this dataset. Figure 5.3 shows that adding the stroke level features, corrects the one miss-classification that occurs in the flowchart dataset. For the digital circuit diagram, the AND and Start classes are improved while the XNOR and NOR classes are slightly degraded (see Figure 5.4). For the HHReco, except the one extra mis-classification for the pentagon and triangle classes, the accuracy of the recogniser either remained the same or improves for the rest of the shapes (shown in Figure 5.5).

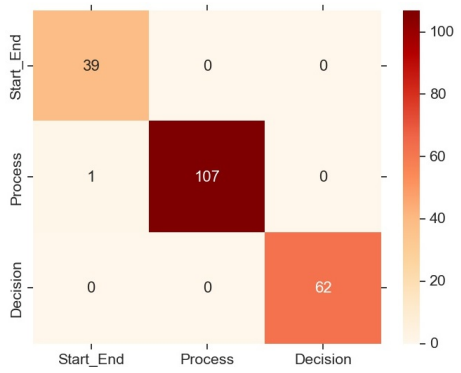


(a) Confusion matrix of FC dataset with original feature set

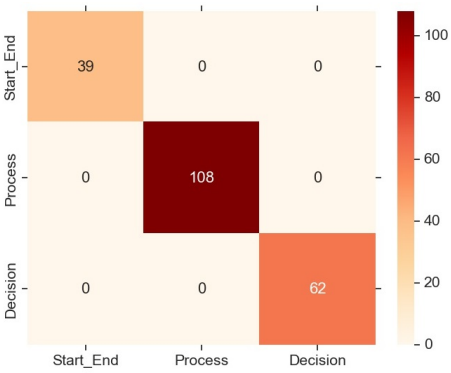


(b) Confusion matrix of FC dataset with added ink features

Figure 5.2: The confusion matrix of the recogniser for FC dataset



(a) Confusion matrix of flowchart dataset with original feature set

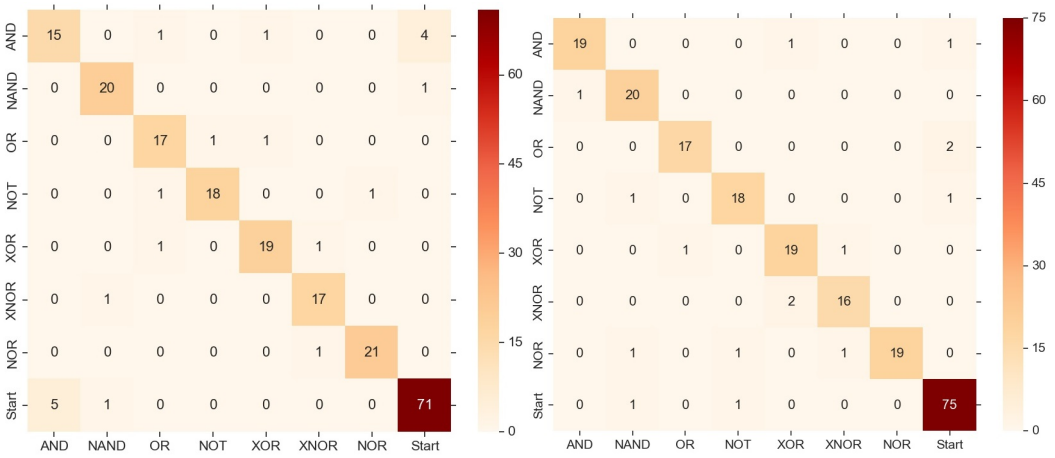


(b) Confusion matrix of flowchart dataset with added ink features

Figure 5.3: The confusion matrix of the recogniser for flowchart dataset

We also measured the computation time of the recogniser. The time that takes to calculate the direction features for a shape candidate on average is less than a millisecond for all datasets. The classification time depends on the training algorithm, its parameters, the feature set used and the complexity of the training data (e.g. number of classes). Table 5.3 shows the classification time for different datasets when the stroke level features are added. For these experiments we used SVM classifier with linear kernel function and $C = 3$. The results show that the classification is not a computationally expensive task.

Recogniser's Characteristics We believe that some of the important characteristics of a recogniser, apart from accuracy and low computation time, should also be invariant to scale, translation, number of strokes, order of drawing and speed of drawing. The described recogniser is invariant to number of strokes and order of drawing as it does not



(a) Confusion matrix of digital circuit dataset with original feature set

(b) Confusion matrix of digital circuit dataset with added ink features

Figure 5.4: The confusion matrix of the recogniser for digital circuit dataset

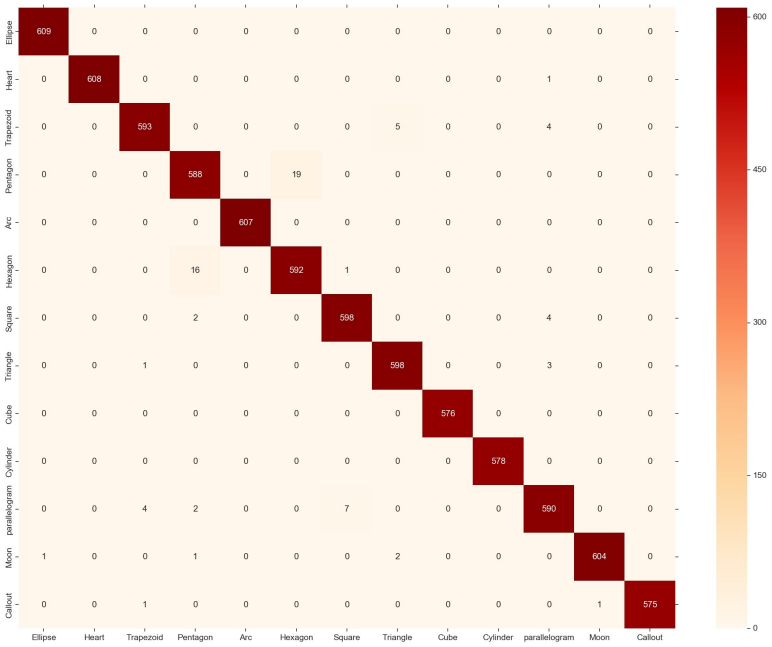
	FC	**FA**	**Flowchart**	**Digital Circuit**	**HHReco**
Time	30 ±4	12 ±2	12 ±2	24 ±3.5	123 ±10

Table 5.3: The average and standard deviation of shape classification time in milliseconds on different datasets

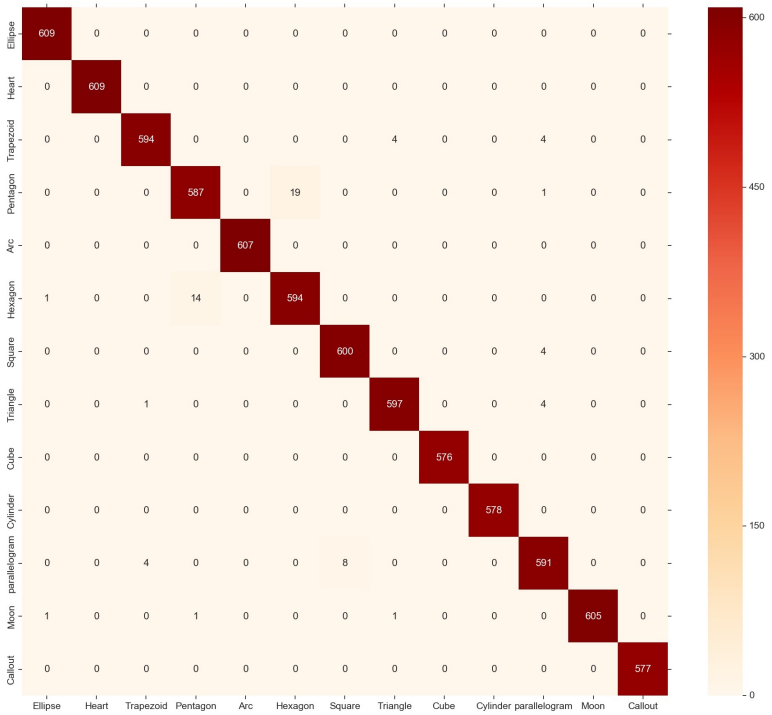
take these information into consideration for feature representation. The recogniser is also invariant to speed of drawing because of the resampling pre-processing. In addition, the recogniser is invariant to scale and translation as the centre of the shape's mass is translated to the origin and then scaled horizontally and vertically in the pre-processing step.

A controversial characteristic for a recogniser is rotational invariance. The described recogniser is inherently rotation variant. This could be an advantage for some domains, such as flowcharts, where some shapes are the rotated form of other one. For example, Figure 5.6 shows a process box is a 45° rotation of the decision shape in flowchart diagram (Stevens et al. 2013) and should be recognised as two different shapes. On the other hand, for some domains such as digital circuits, where the same shapes can be drawn in different rotations, it could be a disadvantage. Figure 5.7 shows an example of two drawn NAND gates in a digital circuit diagram (Stevens et al. 2013) that are drawn in different orientations, but should be recognised as the same shape.

Ouyang and Davis proposed to make the recogniser rotation invariant by rotating the input into 32 evenly spaced orientations from 0 to 360 degrees. As described before, each class is represented as a tree after hierarchical clustering is applied to them. In the process of matching, the rotations of the input is compared against the top 64 clusters



(a) Confusion matrix of HHReco dataset with original feature set



(b) Confusion matrix of HHReco dataset with added ink features

Figure 5.5: The confusion matrix of the recogniser for HHReco dataset

Figure 5.6: An example of a drawn decision and process shapes in a flowchart diagram (Stevens et al. 2013)

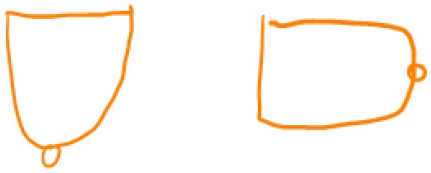


Figure 5.7: An example of two NAND gates drawn in a digital circuit diagram (Stevens et al. 2013)

of each class. This method of rotation invariant can be used only when a template matching technique is being used, which in our experiments were less accurate and more computationally expensive than a discriminative classifier such as SVM. Another possible solution to make the recogniser rotation invariant is to use Hu moments (Hu 1962).

5.2 Rejection

The grouping algorithm described in Chapter 4 requires a recogniser capable of rejecting invalid shape candidates. Invalid shapes are those that either contain strokes from multiple shapes or are incomplete. The majority of existing recognisers take a set of strokes as input and return their best guess to classify the input; typically they are not able to reject an invalid shape. We consider these invalid shapes as outliers that need to be rejected. In machine learning, outlier detection refers to the task of identifying objects that are not similar to the training set. In the following we will describe the approaches we investigated for the task of outlier detection.

In this section, we first describe the method we use for visualising the data. This is followed by describing the proximity-based method we use for rejection. This includes the description of how clusters are formed, and how the validity of a shape candidate is being examined for each cluster. Next, we describe a pre-processing outlier detection method for identifying outliers in a training set. The proximity-based rejection method has a limitation of rejecting incomplete shapes; we propose two solutions for dealing with such shape candidates. In the experiments, we evaluate the performance of the proposed

proximity-based rejection method in isolation and compare the results with two other rejection techniques, using the classifier's confidence, and using a one-class SVM for rejection. Finally, we combine the grouper (introduced in Chapter 4) with the recogniser capable of rejection and evaluate the system's performance on the development datasets.

Visualisation At this stage all the training set instances are represented by a 720 feature vector and all the operations in this chapter will be carried out in 720 dimensions, unless otherwise is stated. However, in order to be able to visualise the distances of the training set instances to each other, we use Multi-Dimensional Scaling (MDS) (Torgerson 1952) to reduce the dimensions to 2 for plotting. Given a distance matrix and the number of desired dimensions, N (i.e. 2 for our visualisation purpose), the MDS algorithm moves the instances around in N dimensional space and checks how well the distances between instances can be reproduced in the new configuration. Two data points that are close together in high-dimensional space will also be close together in the low-dimensional space.

In the remainder of this chapter, for each step of rejection we provide a visualisation. MDS is used where the distances of instances need to be visualised. Using MDS might lead to loss of accuracy but it serves the purpose of visualising the distances. The choice of dataset for this purpose is the flowchart diagram described in Chapter 3. This dataset is relatively small and serves the purpose of illustration. The plot of the dataset in the 2D space after applying MDS to the 720D feature vector can be seen in Figure 5.8. As can be seen in Figure 5.8 there is an instance of the *process* class that is very close to an instance of *Stat/End* class. Figure 5.9 shows these two drawn shapes in the dataset.

5.2.1 Proximity-Based Rejection

The simplest approach for the task of outlier detection would be to use proximity-based approaches, where the distance from the input to all the instances of the training set is calculated, and the input is rejected if it is too far from the training set based on some threshold. However, this approach scales poorly with training set size since the proximity of the input needs to be checked with the entire training set. Instead we use a clustering method where a cluster centre represents a group of similar shapes.

In order to group similar instances of the training set into a single cluster, we use the ground truth to put all the instances of the same class into a single cluster. We then consider the average of all the instances of each cluster as the cluster centre. Figure 5.10 shows the images of the cluster centres of the flowchart diagram.

After dividing the training set into clusters, we need some criteria to assess how well a candidate shape fits with each cluster; or in other words, how similar or dissimilar the shape is to each cluster. Here we use various distance metrics for measuring dissimilarity

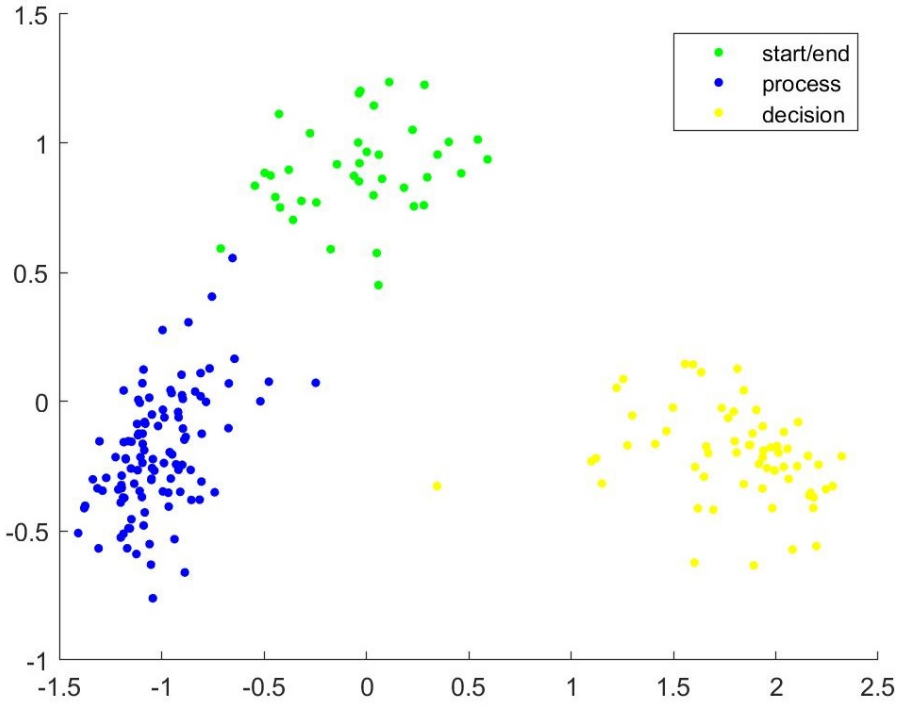


Figure 5.8: The plot of flowchart dataset after mapping the data into 2D space using MDS

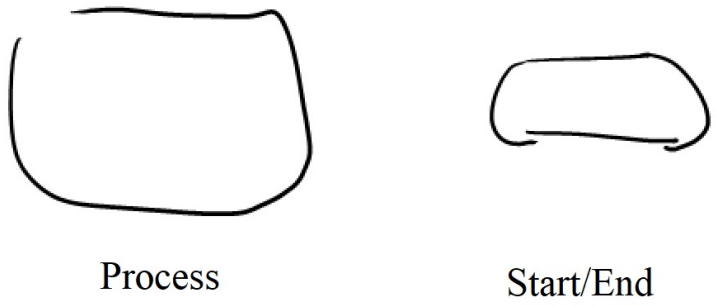


Figure 5.9: The drawn *Process* and *Start/End* shapes that their 2D plot are close to each other in Figure 5.8

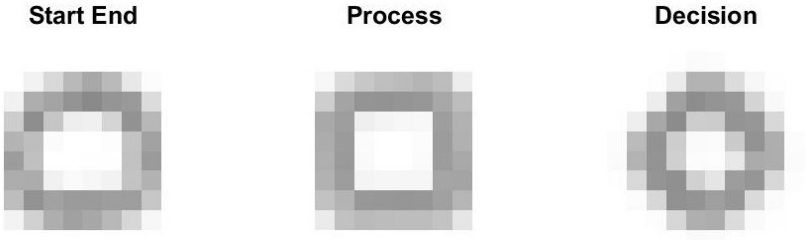


Figure 5.10: The cluster centres of the flowchart diagram

Name	**Distance**
Euclidean	$\sqrt{\sum_{i=1}^{n}(X_i - Y_i)^2}$
Hamming	$\sum_{i=1}^{n} \delta(X_i, Y_i)/n \; ; \; \delta(X_i, Y_i) = \begin{cases} 0 & X_i = Y_i \\ 1 & X_i \neq Y_i \end{cases}$
Bray Curtis	$\sum_{i=1}^{n} \lvert X_i - Y_i \rvert / \sum_{i=1}^{n} \lvert X_i + Y_i \rvert$
Dice	$\frac{\sum_{i=1}^{n} \lvert X_i - Y_i \rvert}{\sum_{i=1}^{n} X_i^2 + \sum_{i=1}^{n} Y_i^2}$
Hellinger	$\frac{1}{\sqrt{2}}\sqrt{\sum_{i=1}^{n}(\sqrt{X_i} - \sqrt{Y_i})^2}$
Kulczynski	$\sum_{i=1}^{n} \lvert X_i - Y_i \rvert / \sum_{i=1}^{n} min\{X_i, Y_i\}$
Manhattan	$\frac{\sum_{i=1}^{n} \lvert X_i - Y_i \rvert}{n}$

Table 5.4: Different distance metrics used for measuring the dissimilarity of two vectors X and Y, both of size n

and some metrics to measure the similarities. We reject a shape candidate if is distant from all clusters and/or not similar enough to any of the clusters. In the following, we will describe how we measure the dissimilarity and similarity of the input shape candidate to each of the clusters and how they are being used for rejection.

Forming Hyper-spheres Around Clusters

Each cluster is represented by the cluster centre, which is calculated by averaging the values of all the cluster members. We form a hyper-sphere around each cluster centre in the 720 dimension and accept the input shape candidate if it falls within any of the cluster spheres, and reject otherwise. A sphere is effectively a statement that the distribution is the same in all directions (a spherical distribution). We make the sphere around each cluster centre by setting the radius to be the distance to the farthest cluster member to the cluster centre. We measure the distance using various distance metrics between the two 720 dimension feature vectors. The list of distance metrics that we used, which are available in Accord.NET framework (*Accord.NET* 2019) can be seen in Table 5.4.

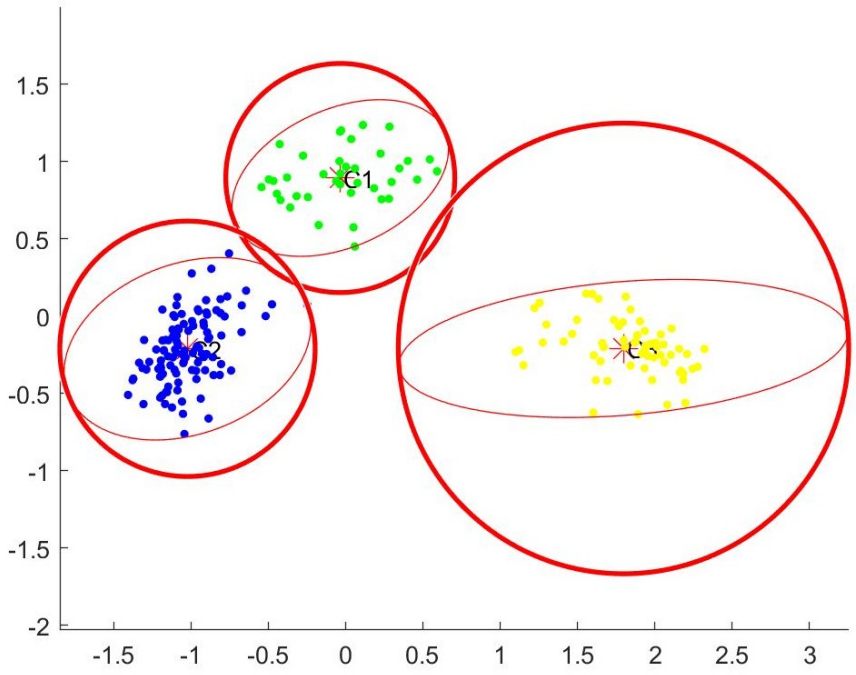


Figure 5.11: The fitted hyper-sphere and ellipse around each cluster for a visual comparison reference

Forming Ellipses Around Clusters

Fitting a hyper-sphere around each cluster can result in allocating some empty space to a cluster (not occupied by any shapes within the cluster). This led us to experiment with fitting an ellipse around each cluster (instead of a hyper-sphere) to avoid extra space being allocated to a cluster. An ellipse is a statement that the distribution varies in different directions. See Figure 5.11 for a visual comparison between fitting a hyper-sphere and an ellipse around each cluster.

An ellipse can be represented in different ways, one is the Foci/String way. Figure 5.12 shows an ellipse with the foci points (F_1 and F_2), the centre (C), major axis (a) and minor axis (b). For any given point on the ellipse boundary (P_i), the sum of distances of P_i to F_1 and F_2 would be the length of string (S) ($||F_1 - p_i|| + ||F_2 - p_i|| = S$). Fitting an ellipse around a set of data points requires knowing the values of F_1, F_2 and S. It should be noted that different values for S gives different ellipses with different sizes, but we are looking for the smallest one that can enclose all the data points.

To find the foci points and the length of the ellipse around each cluster, we perform the following steps:

1. Find the cluster centre (C).

2. Find the farthest data point to the centre (major semi-axis), and consider it as

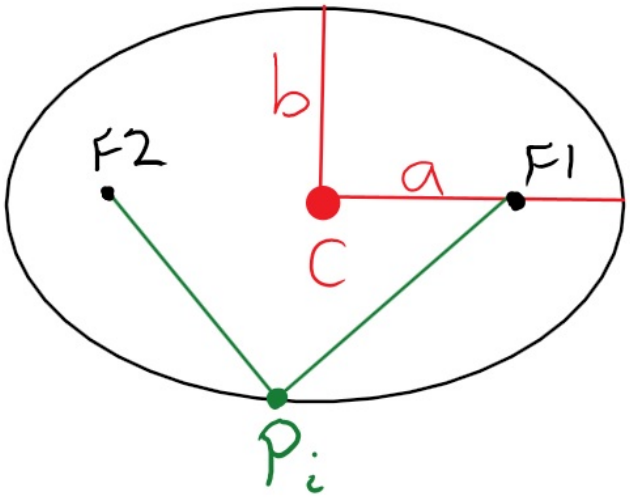


Figure 5.12: An ellipse with the foci points ($F1$ and $F2$), centre (C), major axis (a) and the minor axis(b)

F_1.

3. Find the location of the other end of the semi-major axis and consider it as F_2.

4. For each data point in the cluster (P_c) calculate the sum of absolute distances to F_1 and F_2, and take the maximum as the value of S for the ellipse (see Equation 5.2).

$$S_c = \max_{P_c \in cluster} \{||F_1 - P_c|| + ||F_2 - P_c||\} \tag{5.2}$$

To find the location of the other ending of the major semi-axis, considering that the ellipse might be rotated, we do the following:

1. Move the centre to the origin (0,0).
2. Move the farthest point (F_1) accordingly ($F_1^{'}$).
3. Reflect $F_1^{'}$ ($F_1^{''} = -F_1^{'}$).
4. Move the centre and the farthest point (F_1) back to their locations.
5. Move $F_1^{''}$ accordingly ($F_1^{'''}$). This point is the other end of the major axis.

Rejection using Ellipse After finding the F_1, F_2, S and fitting the ellipse around each cluster, we can check if a given shape candidate fits within any of the ellipses. When a new input feature vector (I) is given, to check if it fits within a cluster's ellipse, we calculate its sum of distances to the cluster's F_1 and F_2 and check if it is less than

the S for that cluster (S_c) ($||I - F_1|| + ||I - F_2|| \leqslant S_c$). The input gets rejected if it does not fall within the ellipse of any of the clusters. If it fits within more than one cluster, the recogniser then determines the class of the shape. It should be noted that all the aforementioned operations are performed in 720 dimensions. The results of this method compared to other rejection methods will be reported in Section 5.3.3.

Similarity Metrics for Rejection

The aforementioned distance metrics measure the dissimilarity of the input to the cluster centres. The dissimilarity between two objects represents the degree to which the two objects are different. On the other hand, similarity metrics measure how the two objects are alike. We use two well-known similarity metrics; Pearson's correlation and the Cosine similarity to measure how similar an input is to a cluster centre.

Pearson's correlation is a similarity metric that measures the linear relationship between two feature vectors (Tan et al. 2018). The correlation value is between -1 and 1, where correlation 1 refers to a perfect linear relationship between the two objects. The correlation of two feature vectors x and y of size n is defined by the following equation:

$$\text{corr(X,Y)} = \frac{\text{covariance(x,y)}}{\text{standard\_deviation(x)} * \text{standard\_deviation(y)}}$$

$$\text{covariance(x,y)} = \frac{1}{n-1}\sum_{k=1}^{n}(x_k - \bar{x})(y_k - \bar{y})$$

$$\text{standard\_deviation(x)} = \sqrt{\frac{1}{n-1}\sum_{k=1}^{n}(x_k - \bar{x})^2}$$

$$\text{standard\_deviation(y)} = \sqrt{\frac{1}{n-1}\sum_{k=1}^{n}(y_k - \bar{y})^2}$$

$$\bar{x} = \frac{1}{n-1}\sum_{k=1}^{n}x_k$$

$$\bar{y} = \frac{1}{n-1}\sum_{k=1}^{n}y_k$$

(5.3)

The cosine metric is a similarity measure that calculates the cosine of the angle between two vectors. Therefore, if the cosine similarity is 1, the angle between the two vectors is 0°(i.e. the two vectors are the same), and if the similarity is 0, the angle

between the two vectors is 90°(i.e. the two vectors are not similar) (Tan et al. 2018). The cosine of two feature vectors X and Y is calculated using the following equation:

$$\cos(\mathrm{X,Y}) = \frac{\langle X, Y \rangle}{||X||\ ||Y||} \tag{5.4}$$

where the $\langle X, Y \rangle$ indicates the inner product of the two vectors and $||X||$ is the length of vector X.

Rejection For rejection using similarity, for each cluster we choose a threshold ($T_{similarity}$) determined by the least similar instance of the cluster to its centre. For a given input, we calculate its similarity (either using Equation (5.3) or (5.4)) with all the cluster centres. If the input is similar enough to one of the cluster centres (less than $T_{similarity}$ for that cluster), the shape gets accepted. If the rejection method is the combination of multiple metrics, the shape candidate gets rejected if any of the metrics reject it.

5.2.2 Outlier Detection within Clusters

Due to human errors we often see mislabelled shapes in a dataset, or sometimes badly drawn shapes (see an example of a badly drawn decision shape in Figure 5.13). Such instances would lead to a larger ellipse and lower similarity threshold for that cluster, resulting in lower rejection accuracy. We treat these instances as outliers and remove them as a pre-processing step. This process is carried out before fitting an ellipse and finding the similarity threshold for each cluster.

After forming the clusters, we check the similarity score of each cluster member with its cluster centre. If the similarity of an instance with its cluster centre is less than a threshold[1] (T_{OD}), we remove that instance from that cluster. Figure 5.14 shows the plot of the similarity of each cluster member to its centre. As can be seen in this figure, the decision instance that is badly drawn (shown in Figure 5.13), has a lower similarity to the centre as compare to the rest of the instances in that cluster. Figure 5.15 shows the ellipse around data before and after outlier detection preprocessing.

5.2.3 Incomplete Shapes

In our experiments (in Section 5.3) we observed that some of the false positives (shapes that were supposed to get rejected but did not) are the incomplete shapes. After analysing these incomplete shapes, we realised that most of them are the shapes that have one missing side. We also realised that some of the invalid candidates or some of the arrow shafts are similar to an incomplete shape in that domain (see Figure 5.16 for

[1] We empirically used a value of 0.65 for T_{OD}

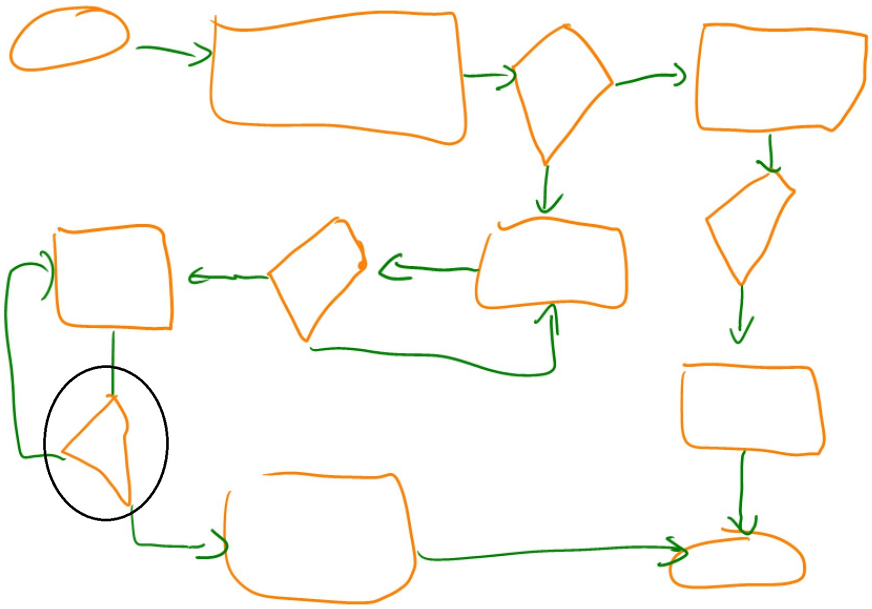


Figure 5.13: An example of badly drawn decision symbol in a diagram (the one with a circle around it) in the flowchart dataset

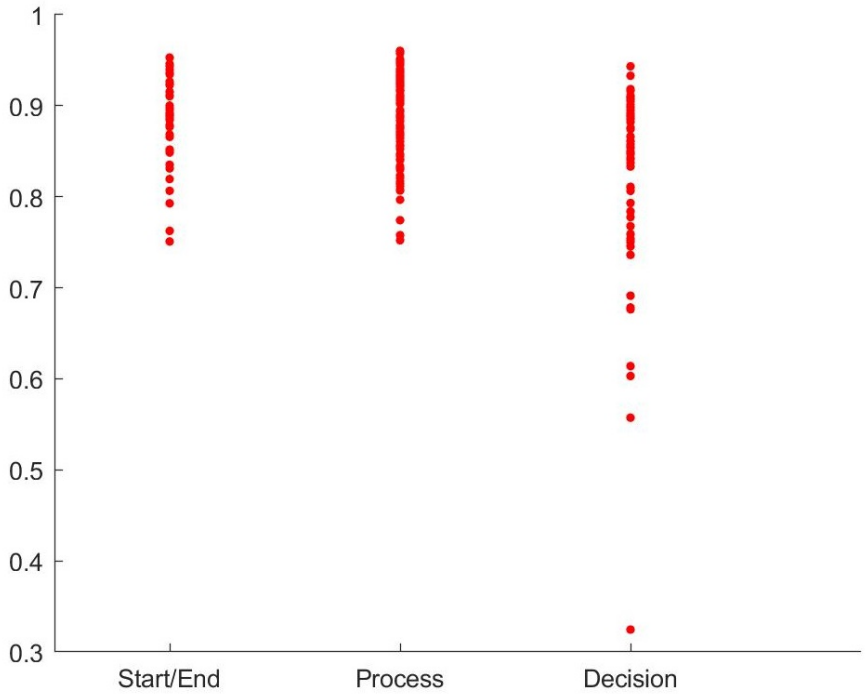
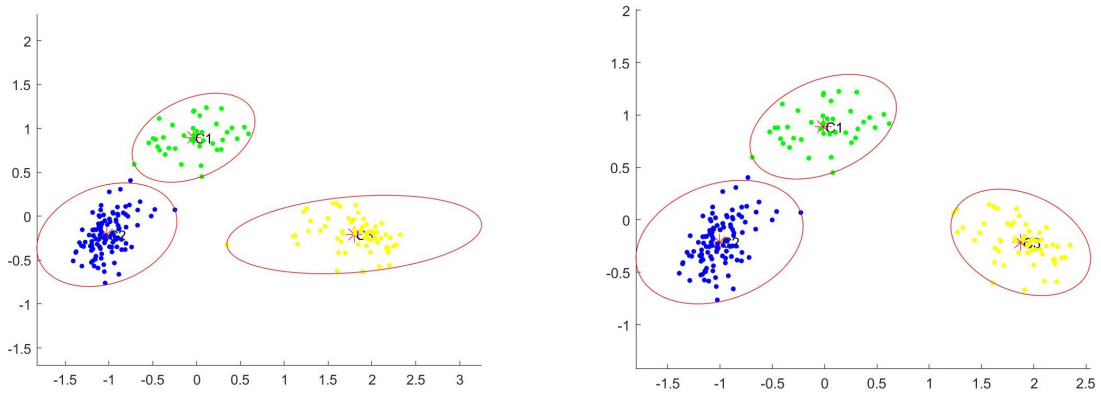


Figure 5.14: The plot of similarity of each cluster member to its cluster centre



(a) (b)

Figure 5.15: The visualisation of the flowchart dataset before and after outlier detection step with the fitted ellipses. (a): The fitted ellipse around each cluster, (b): The fitted ellipse around each cluster after preprocessing outlier detection.

an example of an arrow shaft that looks like an incomplete process (a rectangle) in the flowchart diagram).

We considered various options to deal with this issue. One option is to perform a final check before accepting a shape candidate using heuristics. For example a naïve approach would be to only accept closed shapes. However, this would restrict the approach to particular domains. Another approach would be to do a local search around the accepted shape candidate by adding the neighbouring strokes and checking if the similarity to the cluster centre would improve. We observed that sometimes adding an arrow head to the shape would increase its similarity to the cluster centre (for example see Figure 5.17 when the arrow head is added to the square, the similarity score increases). Hence, this approach did not work very well either. In the following we will describe the two approaches we used to deal with the incomplete shapes.

Training A Classifier with Incomplete Shapes

Since the rejection system has limitations in rejecting the incomplete shapes (the ones with a missing side), we decided to train a classifier to learn about such instances. The classifier is therefore trained with the incomplete shapes as well as the complete ones. This is inspired by the work of negative examples described in Section 2.2.1, in which a classifier is trained with a very large number of negative examples. However, we only train the classifier to learn about the shapes that the proximity-based rejection method is not able to reject.

The process of training the classifier with the incomplete shapes can be seen in Figure 5.18. The grouper and a mock recogniser work simultaneously together to group and recognise the shapes in diagrams of the training set. All the hypothesised shape

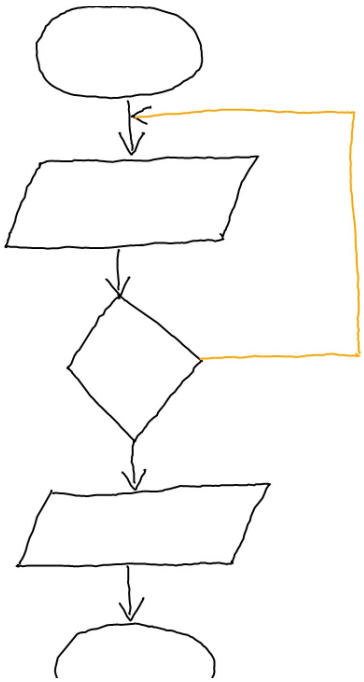


Figure 5.16: An example of an arrow shaft (coloured with orange) that looks like an incomplete process (rectangle) in the FC dataset

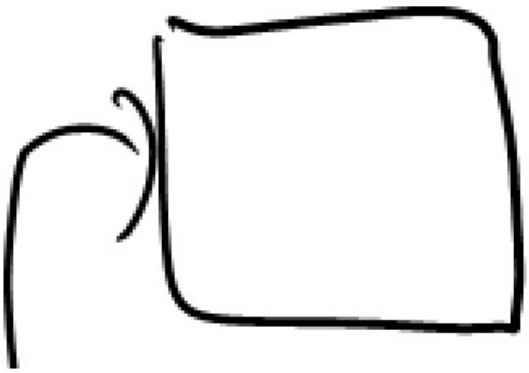


Figure 5.17: An example of an arrow head attached to the shape that improves the similarity score from the flowchart dataset

candidates during grouping and recognition are captured. These shape candidates are then labelled automatically. The labels can be "incomplete-shape name" (if it has missing strokes of a complete shape), "invalid" (if it consists of strokes from multiple shapes or is a connector) or the actual "shape name" if it is valid. All the hypothesised shape candidates are then given to the rejector and the incomplete shapes that are not being rejected are captured. These incomplete shapes with the complete ones are then used to train the classifier.

In our experiments, we found that training the SVM with the merged feature images works better than training with the original 720 dimension features (see Section 5.3 for the results). As described in Section 5.1, we get the merged feature vector by capturing the max value of the five feature images for each pixel, resulting in a 12×12 feature image, which is a vector of size 144 if stored row by row. We also capture the average of each feature image and the number of non-zero pixels. This results in a feature vector of size 154.

The merged feature vectors of the complete shapes and the incomplete shapes are used to train an SVM classifier. This classifier acts as the second tier of rejection in the use phase, i.e., if a shape candidate is not rejected by the rejector, this classifier determines whether it is a complete shape or not. If a shape candidate is classified as any of the incomplete shapes, it gets rejected. To determine the parameters for the SVM, we performed a grid search. The best results were achieved by the RBF kernel, using $C = 3$ and $\gamma = 0.5$.

It should be noted that in some domains (in particular the ones with extendable shapes), an incomplete form of a shape might represent a complete form of another shape in that domain. For example, as shown in Figure 5.19 in the domain of a digital circuit diagram, an incomplete NAND gate (without the circle at the end of the gate) represents a complete AND gate. Therefore, for such domains we train the classifier only with the incomplete shapes that cannot represent a complete shape. This information is explicitly given to the system.

Using Image Masking for Incomplete Shapes

Another approach we used in our system to deal with the incomplete shapes is an image-based approach to first observe which part of the incomplete shape is missing, and then find a match for the missing piece/s among the available neighbouring strokes. To determine which part of the shape is missing we mask the image of the input shape (the incomplete shape) with the cluster centre that it has best fitted in. To find a match for the missing piece/s we use the same masking technique. For this approach we need to work with the images of the input and the cluster centre. If the shape is similar or close to more than one cluster, we pick the most similar or the closest one. We assume

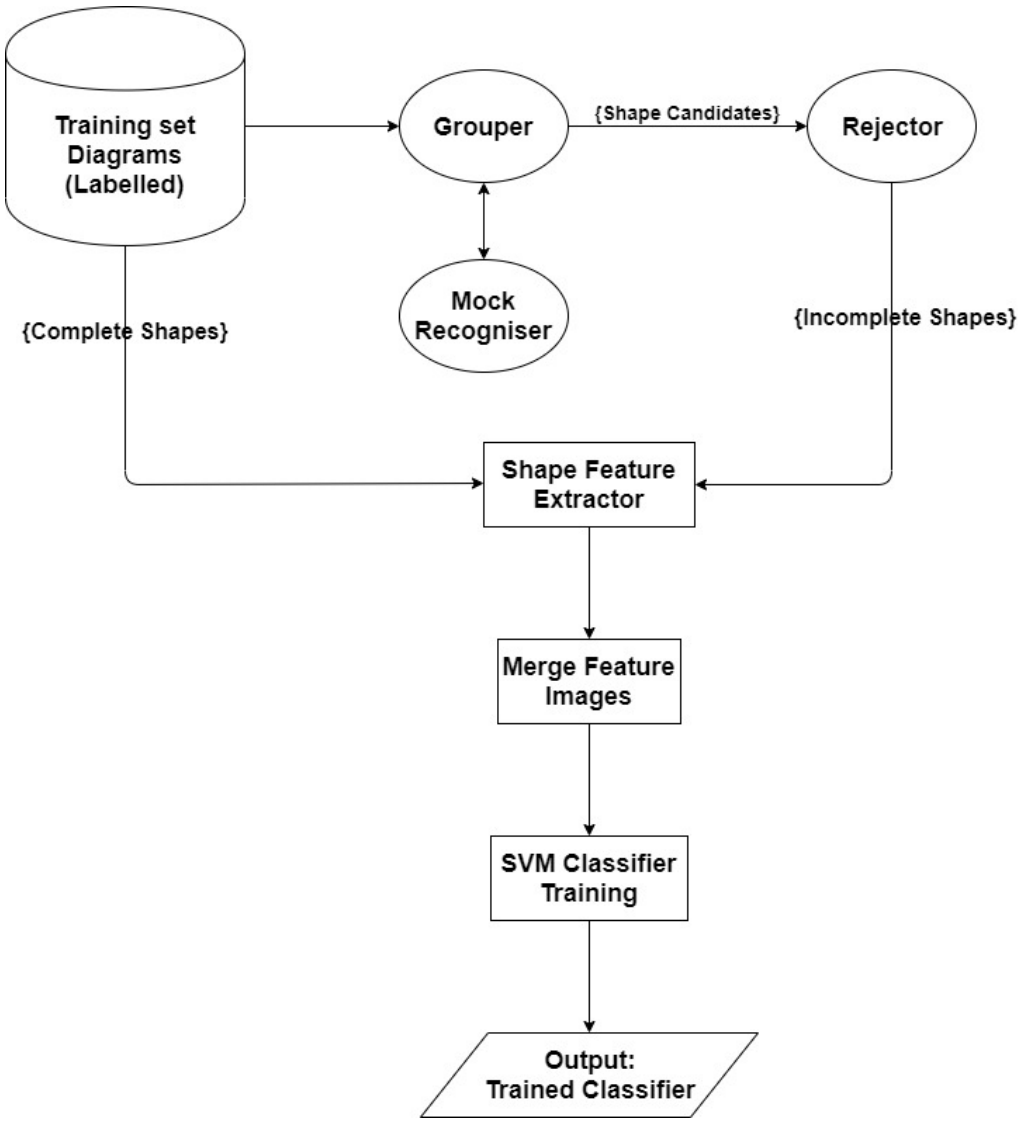


Figure 5.18: The process of the training classifier with the incomplete shapes for rejection

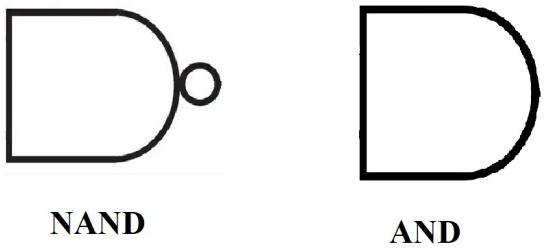


Figure 5.19: An example of AND and NAND gates in the domain of a digital circuit diagram

the incomplete shape is closest to the cluster representing the shape if it was complete. In the following we describe this approach step-by-step:

1. Convert the shape candidate and the cluster centre to images. In Figure 5.1a we displayed each of the five feature images for a shape. To have a single image representing the shape candidate (or the cluster centre), we merge these five feature images by taking the maximum value for each pixel.

2. Masking ($\oplus$) the cluster centre (C) with the input image (I), which results in the masking output (MO) (see Figure 5.20 for a visual example of this step). In the process of masking, we also check if the pixel values of the cluster centre are above a threshold (T_m)[2]; this is because the cluster members might have some variations in drawing and therefore, the cluster centre (which is the average of all the members) may have values that are not really representative of the cluster shape:

$$MO_{(x,y)} = \begin{cases} C_(x,y) & I_{(x,y)} == 0, C_{(x,y)} > T_m \\ 0 & otherwise \end{cases} \tag{5.5}$$

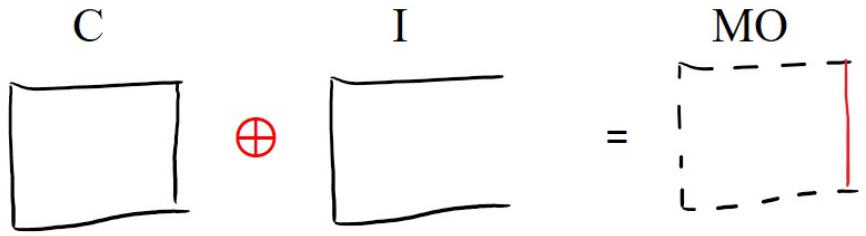


Figure 5.20: An visual example of masking. The red line in the MO shows the missing part of the input and its location with respect to the drawn parts (the dashed lines).

3. Add an immediate neighbouring stroke (n) to the shape (I) to form the new shape candidate (New_S) (see Figure 5.21):

$$New\_S = S \bigcup n \tag{5.6}$$

[2]We used a value of 0.2 for T_m

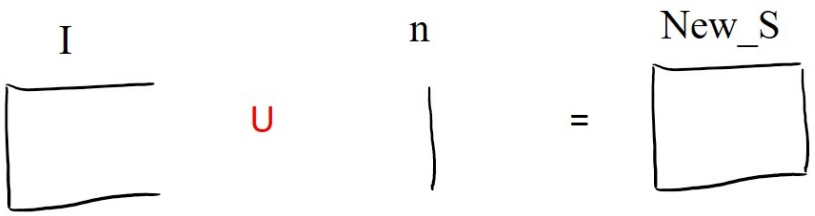


Figure 5.21: Adding n to the shape to form New_S

4. Masking the New_S with the MO to see how closely they match to find the Region of Interest (ROI) (note that the MO has 0 in all the pixels except for the missing parts of the shape). The ROI is the missing part of the shape. See Figure 5.22 for a visual example of this step:

$$ROI_{(x,y)} = \begin{cases} MO_{(x,y)} & MO_{(x,y)} > 0 \\ 0 & otherwise \end{cases} \tag{5.7}$$

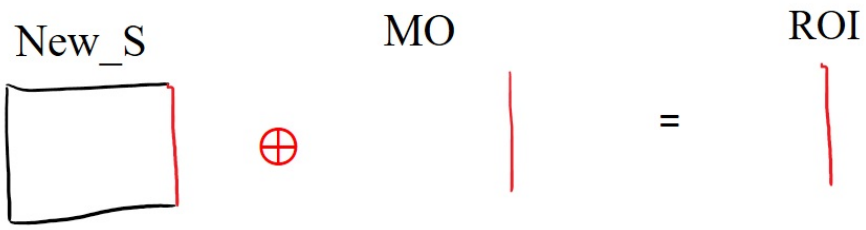


Figure 5.22: Getting the ROI

5. Checking if the similarity (Pearson's correlation) of the added stroke (ROI) with the masking output (MO) is above a threshold (T_{s1})[3]:

$$Similarity(ROI, MO) > T_{s1} \tag{5.8}$$

6. If the similarity from the previous step is greater than the threshold, it means that the added stroke covers at least some of the missing part of the shape. However, we still need to make sure that the added stroke does not have other parts. For example, in the experiments we saw that sometimes a part of an arrow head can cover the missing part of the shape, but that was not the whole drawn stroke. Hence, we need to check if the similarity (Pearson's correlation) of the added

[3]We empirically selected a value of 0.6 for T_{s1}

stroke (n) with the ROI is above a threshold (T_{s2})[4].

$$Similarity(n, ROI) > T_{s2} \tag{5.9}$$

Since the missing part might be drawn with multiple strokes, we iterate through steps 3 to 6 to find the matching strokes. The pseudo code for this process can be seen in Algorithm 2.

Algorithm 2 The algorithm for finding the missing strokes of an incomplete shape

Input:

{S},{C},{U} ▷ {S: shape candidate, C: List of cluster centres, U: List of unrecognized strokes}

Output: *{S}* ▷ {(S: The updates shape candidate)}

1: $improved = true$
2: **while** (improved) **do**
3: $improved = false$
4: for all neighbours $\{n\}$ of S:
5: $New\_S = S \bigcup n$
6: $ROI = New\_S \oplus MO$ (based on equation 5.5)
7: **if** (similarity$(MO, ROI) < T_{s1}$ and similarity$(n, ROI) < T_{s2}$) **then**
8: $S = New\_S$
9: $improved = true$
10: **end if**
11: **end while**
12: return S

5.3 Experiments

In this section we first evaluate the performance of the proposed proximity-based rejection method in isolation. We compare the results with two other rejection methods, i.e., one-class classification and the use of classifier's confidence for rejection which will be described in the following. Next, we incorporate different rejection methods with the grouper and evaluate the performance of the system, thus far, with these rejection methods. Finally, we discuss the performance of the system and provide some analysis.

As mentioned in Section 3.4, the FC and FA datasets are divided into the training and test sets. For the following experiments for FC and FA datasets, we use the specified training set to train the system. However, this is not the same for the flowchart, class, and digital circuit datasets. Therefore, for these datasets, we perform a 5-fold cross validation.

[4]We used a value of 0.5 for T_{s2}

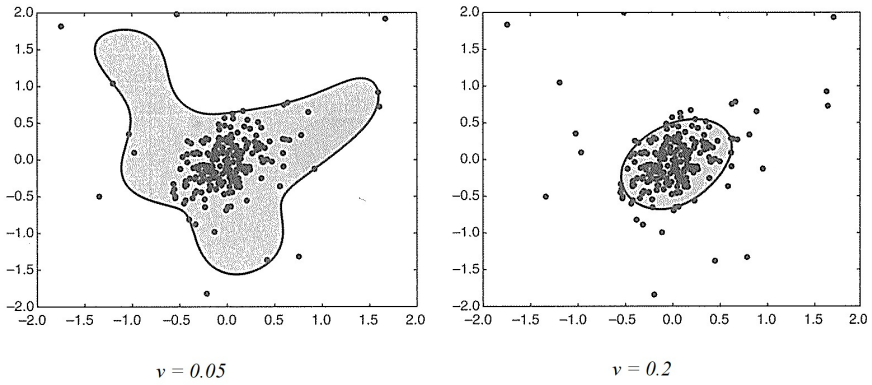


Figure 5.23: Decision boundaries of one-class SVM for different v values. Picture from (Tan et al. 2018)

5.3.1 One-Class SVM for Rejection

Traditional classification algorithms focus on the classification of two or more classes. The multi-class classification problems are usually decomposed into multiple two-class classification problems. The two-class classification problem is then considered as the basic classification task (Pimentel et al. 2014) in which the classifier learns to distinguish between the two classes. On the other hand, in a one-class classification problem, one class (i.e. the normal class) needs to be distinguished from all other possibilities.

In a One-class classification problem a decision boundary that encloses all the normal classes is learned. The SVM is a popular technique for forming decision boundaries that separates data into different classes. SVM-based one-class classification (known as one-class SVM) is a popular approach for the one-class classification problem. A conventional SVM classifier separates two classes by transforming all the data to a high-dimensional space, then separating the two classes using a linear hyperplane. In a one-class classification problem, since we only have one class, the algorithm tries to separate the instances in high-dimensional space from the origin. In the original space, this corresponds to finding a small region that encloses all the training instances. If a given input falls into this region, it is considered as a normal class, otherwise is identified as an outlier (Géron 2019). Apart from the usual kernel parameters for the SVM, a margin hyperparameter (v) needs to be tuned. The v parameter represents the upper bound on the fraction of training instances that can be mistakenly considered as anomalies while learning the hyperplane (Tan et al. 2018). Figure 5.23 shows the decision boundary of one-class SVM for different values of v. The class of the accepted shapes are determined by the trained recogniser.

In order to use one-class SVM for rejection in our system, for each class in the training set, we train a one-class SVM. Therefore, for a training set with n classes, we end up with n SVM models. When a new shape candidates is given to this rejector, if

all the SVM models reject the input, it gets rejected. Otherwise, if one or more of the SVM models accept the input, it gets accepted.

5.3.2 Classifier's Confidence for Rejection

The SVM classifier used for the shape classification can output the distance from the input to the closest point on the decision boundary, and be used as a confidence score. This score cannot directly be used to estimate the class probabilities. However, there are techniques for converting scores into class probability. Different implementations of SVM use different techniques for estimating the probability. In our experiments, we use the LibSVM Sharp (*LibSVMsharp* 2019) which is a .Net Wrapper for the LibSVM. In LibSVM, after training the classifier, it calibrates probabilities using Logistic Regression on the SVMs scores, fitted by an additional five-fold cross-validation on the training data. The details of how LibSVM is extended to provide probability estimates are described by Chang & Lin (2011) and Wu et al. (2004).

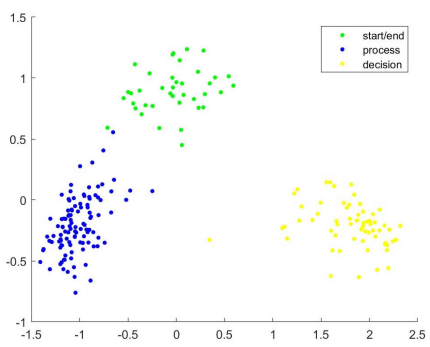
Having the probability estimate of each class, we need to set a threshold for rejecting shape candidates. We choose a threshold for each class separately. After the training set is used to train the classifier, we use them to determine the threshold for each class as well. Each shape in a class is given to the classifier, and the lowest predicted probability for the class is captured as the threshold for that class. We obtain the thresholds after the pre-processing outlier detection to make sure the outliers do not affect the determination of the thresholds.

5.3.3 Rejection Evaluation in Isolation

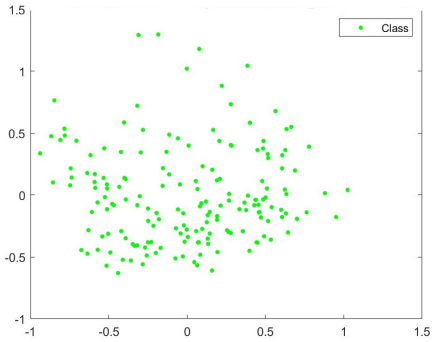
We evaluate the rejection system's performance on the five development datasets. In order to have a visual insight of these datasets, the plot of these datasets after mapping their data into 2D space using MDS (without the preprocessing outlier detection) is shown in Figure 5.24.

In order to be able to evaluate the rejector's performance in isolation, besides the valid shapes, we need some invalid shape candidates to check how well they get rejected while the valid ones are being accepted. Similar to the process described in Section 5.2.3, we generate these shape candidates by incorporating a mock recogniser with the grouper and capturing all the hypothesised shape candidates. This process assures that all the valid shape candidates are hypothesised and also there would be a large number of invalid shape candidates to evaluate the rejection methods with. Table 5.5 shows the details of generated shape candidates for the development datasets.

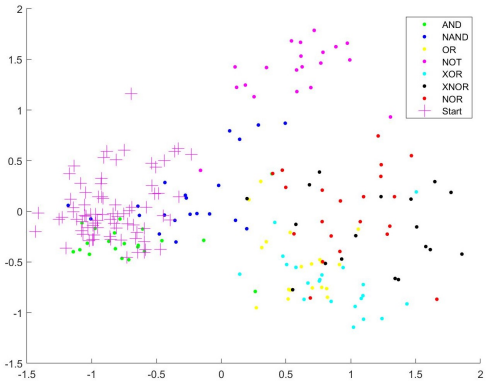
For the domains with extendable shapes (i.e. digital circuit and FA datasets), the sub-shapes of a shape might represent another shape in that domain. For example, as shown in Figure 5.25, an XNOR gate has sub-shapes of XOR, NOR and OR gates. If



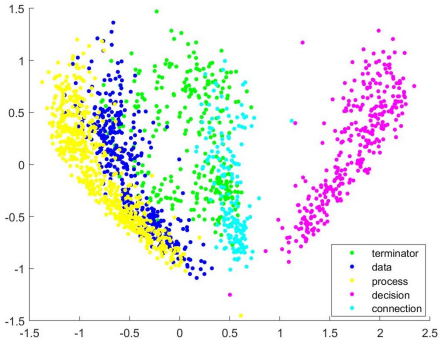
(a) Flowchart diagram



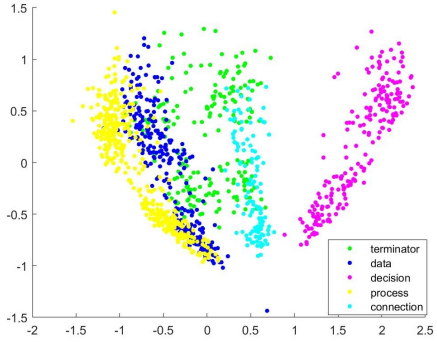
(b) Class diagram



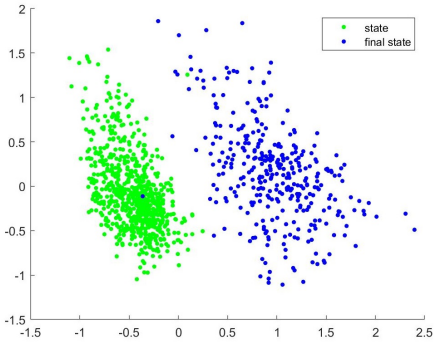
(c) Digital Circuit diagram



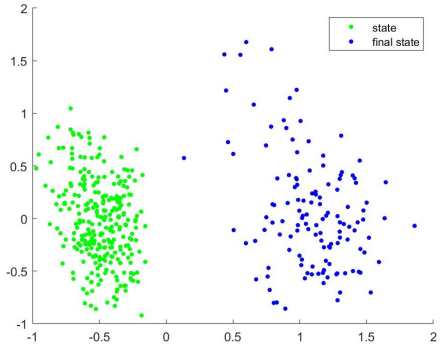
(d) FC Training set



(e) FC Testing set



(f) FA Training set



(g) FA Testing set

Figure 5.24: The plot of the development datasets' shapes after mapping the data into 2D space using MDS. All these plots are before the pre-processing outlier detection step.

	Valid	Invalid	Incomplete
Flowchart	210	874	362
Class	171	1724	4185
Digital	247	14390	1055
FC	1225	26346	8401
FA	671	54982	15

Table 5.5: The details of generated shape candidates using the grouper and a mock recogniser

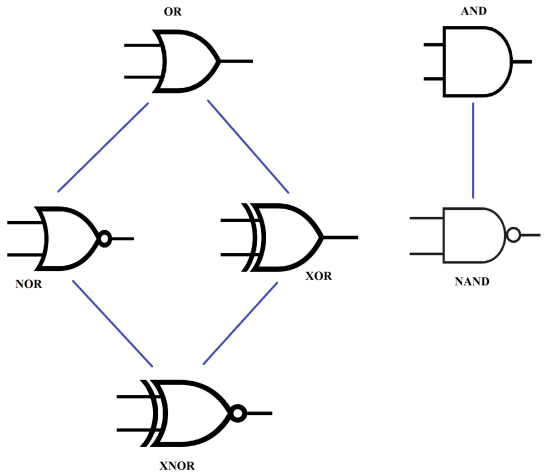


Figure 5.25: A visual representation of the extendable shapes in the digital circuit domain

any of these valid sub-shapes are labelled as "incomplete-XNOR" and get accepted by the rejector (since they represent a valid shape), it counts as a false positive which would affect the evaluation results. In addition, if we train a classifier with the incomplete shapes, having two shapes from the same class being labelled differently (e.g. one as OR and one as "incomplete-XNOR") would affect the classifier's performance. Therefore, we relabel such datasets to provide label for the sub-shapes that represent a valid shape. While generating the shape candidates and labelling them, if a candidate can be considered as the incomplete of multiple shapes, we label it as an incomplete of the shape that has no further sub-shapes. This would also provide a consistency in the labels for training a classifier with incomplete shapes. For example, a drawn stroke shown in Figure 5.26 can be labelled as "incomplete-XNOR", "incomplete-XOR", "incomplete-NOR;" or "incomplete-OR"; however, we label it as "incomplete-OR" since the OR gate has no further valid shape.

We evaluate the performance of a rejection system using True Positive Rate (TPR), True Negative Rate (TNR) and Area Under Curve (AUC). The TPR (Equation 5.10) measures how well the positive examples are being accepted. The TNR (Equation 5.11)

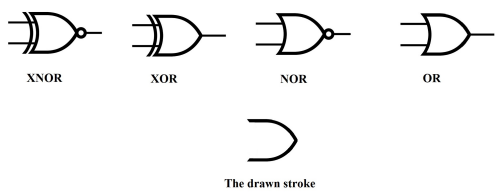


Figure 5.26: A representation of XNOR gate with its possible sub-shapes (XOR, NOR and OR gates) and an example of a drawn stroke.

measures how well the negative candidates (the invalid and incompletes) are being rejected. In the following equations True Positive (TP) refers to the valid shapes that are correctly accepted, False Positive (FP) refers to the invalid shape candidates that are incorrectly accepted, True Negative (TN) refers to the invalid shape candidates that are correctly rejected and the False Negative (FN) refers to the valid shapes that are incorrectly rejected.

$$TPR = \frac{\#TP}{\#TP + \#FN} \tag{5.10}$$

$$TNR = \frac{\#TN}{\#TN + \#FP} \tag{5.11}$$

It is common to compare different learning algorithms using the Receiver Operating Characteristic (ROC) curve. The ROC curve is a plot of TPR (Equation 5.10) on the y axis and against false positive rate (1 - TNR) on the x axis. A perfect rejection method would be at (0,100) where has achieved 100% true positive and 0% false positive, while the worst rejection that gets everything wrong would be at (100,0); so the closer to the top-left-hand corner the result of a rejection method, the better the classifier has performed. Figure 5.27 shows an example of a ROC curve, where the diagonal line represents a system that behaves at a chance level and the other two lines above that are better (the farther above the diagonal line, the better the performance of the system) (Marsland 2015). The Area Under Curve (AUC) is an integrated quantitative representation of ROC curve, which is commonly used to measure the performance of outlier detection systems (Ding et al. 2014). The AUC can summarise the ROC curve as the higher the AUC value, the better the system is performing.

In our experiments, we count a shape candidate as a true positive if is a valid shape that is accepted by the rejector. For the evaluations in this chapter, we do not consider the shape label, which allows us to measure the rejector's accuracy without having the classifier affect the results. In the following we first report the results of different proximity-based rejection methods with the original feature space (720D) on the development datasets. Next, we perform the same experiments with the merged feature representation. Finally, we train a classifier with the incomplete shapes and

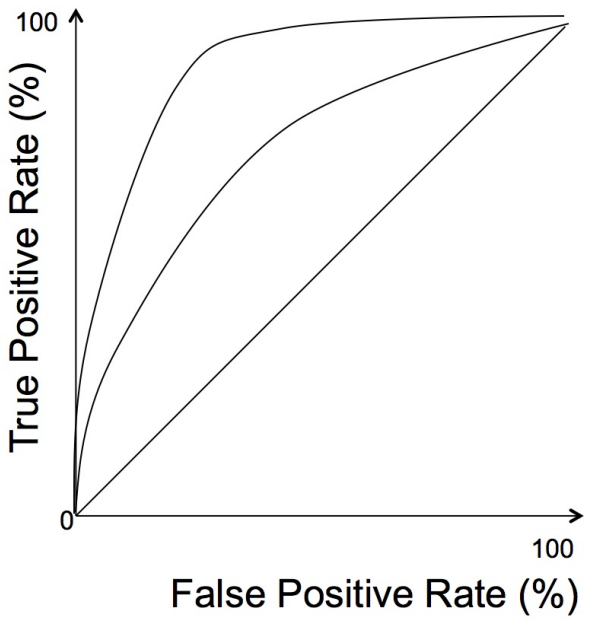


Figure 5.27: An example of an ROC curve (Marsland 2015)

carry out the same experiments.

Results on the Original Space

We evaluated the proximity-based rejection method (fitting the hyper-sphere) with different distance metrics, the similarity metrics and the method of fitting an ellipse around each cluster. We also combined each of the distance metrics with a similarity metric and measured the performance. For such cases that a combination of two metrics are used, a shape candidate gets rejected if at least one metric (either the similarity or the distance) rejects the candidate. The distance metrics, similarities and their combinations results in 26 different rejection methods. The full results table of the 26 methods can be seen in Appendix A.

The results showed that no single approach outperforms others in terms of TPR, TNR and AUC. Some metrics achieve a higher TPR than others while having a lower TNR, which makes it difficult to decide which approach performs better. Therefore, we use AUC metric when comparing the results of different approaches. A summary of the best performing methods based on the average of their AUC on different datasets can be seen in Table 5.6.

Fitting a linear mixed effects model shows that there is a significant difference in the 26 methods of rejection ($p < 0.05$). In order to obtain pairwise mean comparison, we fit Analysis of Variance (ANOVA) model and perform TukeyHSD test. The results show

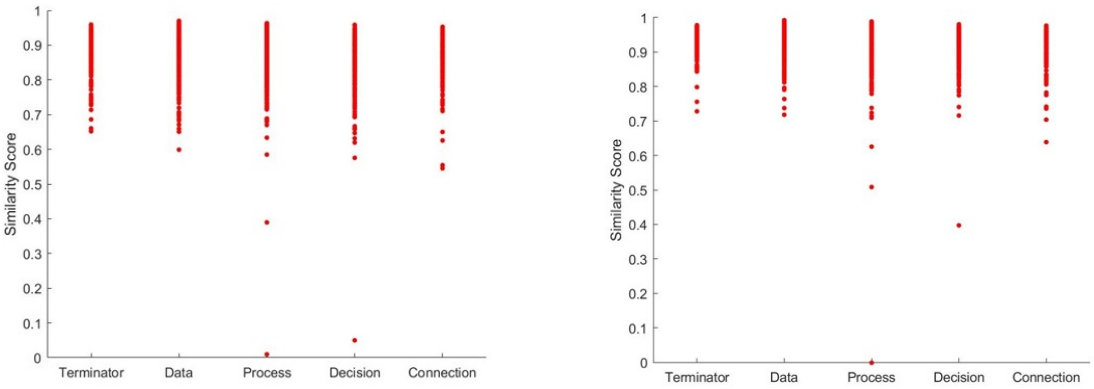
		FC	FA	Flowchart	Class	Digital
Dice + Correlation	TPR	99.5%	100.0%	95.2%	98.8%	91.5%
	TNR	89.4%	95.3%	95.0%	87.6%	94.0%
	AUC	94.5%	97.6%	**95.1%**	**93.2%**	92.8%
Dice + Cosine	TPR	99.6%	100.0%	95.7%	99.4%	97.2%
	TNR	87.0%	93.4%	93.9%	86.3%	91.1%
	AUC	93.3%	96.7%	94.8%	92.9%	**94.1%**
Hellinger	TPR	99.5%	100.0%	95.2%	99.4%	94.7%
	TNR	89.1%	96.8%	93.9%	84.7%	92.5%
	AUC	94.3%	**98.4%**	94.5%	92.1%	93.6%
Hellinger + Correlation	TPR	99.5%	100.0%	95.2%	98.8%	91.5%
	TNR	90.2%	96.9%	94.2%	86.0%	93.2%
	AUC	**94.8%**	**98.4%**	94.7%	92.9%	92.4%
Hellinger + Cosine	TPR	99.5%	100.0%	95.2%	98.8%	94.3%
	TNR	89.5%	96.8%	93.9%	84.7%	92.5%
	AUC	94.5%	**98.4%**	94.5%	91.7%	93.4%

Table 5.6: The summary of the best results of proximity-based rejection methods on the original feature space.

that the Kulczynski method is significantly worse ($p < 0.05$) than all other methods except the Dice, Manhattan and Euclidean. The Dice method is also significantly different from all others except the Euclidean and Manhattan. The rest of methods are not statistically different from each other.

The results show that any distance metric that is combined with a similarity metric has resulted in a better AUC. The idea of combining two metrics is to achieve a better TNR, since we reject a shape candidate if any of the two metrics reject it. We also carried out an experiment by combining the correlation and cosine similarity metrics, which the results were exactly the same as the correlation metric. This implies that combining a distance metric with both of the correlation and cosine metrics would achieve the same results as the combination of the distance metric with the correlation metric. We also fit a linear mixed effects model on the result of the combined methods only (combination of a distance with a similarity metric) and the results show that there is no significant difference among combined methods.

The results show that for the FC and FA datasets, the best results are achieved through the combination of Hellinger distance and correlation similarity metric (having TPR 99.5%, 100% and TNR 90.2%, 96.9% and AUC 94.8%, 98.4% for the FC and FA, respectively). For the Flowchart and Class diagram datasets, the Dice + Correlation achieves the best results, while for the digital circuit dataset the Dice + Cosine achieves the best results (having TPR 95.2%, 98.8%, 97.2% and TNR 95.0%, 87.6%, 91.1% and AUC 95.1%, 93.2%, 94.1% for the flowchart, class and digital circuits, respectively). The results in Table 5.6 seem promising (having AUC in the range 93.2% to 98.4%),



(a) Original space

(b) Merged images

Figure 5.28: Comparison of the similarity score of each instance to its cluster centre for the original space 5.28a and the merged one 5.28b.

however, we will carry out the same experiments on the merged features for possible improvements on the results.

Results on the Merged Features

We carried out the same experiments of using different metrics for rejection on the merged feature representation. As mentioned before, we get the merged features by capturing the max value of the five feature images for each pixel, as well as the average of each feature image and the number of non-zero pixels, which results in a feature vector of size 154. It should be noted that when merged features are used, the threshold for the pre-processing outlier detection method is changed to 0.7 (from 0.65) as the similarity of cluster instances to the centre are higher (see Figure 5.28 for a comparison of the similarity metrics of each instance to its cluster centre for the FC dataset).

In our experiments, Hellinger + Correlation achieves the best results for FC and FA datasets, and Dice + Correlation achieves the best results for the flowchart, class and digital circuit diagrams. The summary of these two methods can be seen in Table 5.7. The full table of the combined methods on the merged features can be seen in the Appendix B.

Fitting linear mixed effects model on the combined methods shows that there is a significant difference in the method of rejection when merged features are used ($p < 0.05$). The TukeyHSD test results show that the Hamming + Cosine is significantly worse than seven methods of Hamming + Correlation, Dice + Correlation, Hellinger + Cosine, Hellinger + Correlation, Kulczynski + Correlation, Ellipse + Cosine, Ellipse + Correlation. The TukeyHSD test shows that there is no significant difference between other methods. In addition, we fit a linear mixed effects model on the combined methods

Dataset	Method	TPR	TNR	AUC
FC	Dice + Correlation	99.5%	89.4%	94.5%
	Hellinger + Correlation	99.5%	90.2%	**94.8%**
FA	Dice + Correlation	100.0%	95.3%	97.6%
	Hellinger + Correlation	100.0%	96.9%	**98.4%**
Flowchart	Dice + Correlation	95.2%	95.0%	**95.1%**
	Hellinger + Correlation	95.2%	94.2%	94.7%
Class	Dice + Correlation	98.8%	87.6%	**93.2%**
	Hellinger + Correlation	98.8%	85.0%	91.9%
Digital	Dice + Correlation	91.5%	94.0%	**92.8%**
	Hellinger + Correlation	91.5%	93.2%	92.4%

Table 5.7: The results of the two methods Dice + Correlation and Hellinger + Correlation on the merged features

of the original space and the merged ones. The results show that there is no significant difference among all these approaches when all considered together. Therefore, between the merged and the original features, the latter is preferred to avoid the extra computation of merging the features.

Results for Incomplete Rejection

In Section 5.2.3, we proposed two approaches to deal with the incomplete shape candidates that have a missing side and are being accepted by the proximity-based rejector. One approach is to train a classifier with the incomplete shapes to discriminate them from the complete ones, and the other approach is to use image masking to find the missing part of an incomplete shape. Using the image masking can be done in the process of grouping and recognition and cannot be used for evaluation of rejector in isolation (however, we will evaluate this approach later in this chapter when grouper and recogniser are incorporated). Therefore, we report our results for the case that a classifier is trained with the incomplete shapes.

Each shape candidate is first checked with the proximity-based rejector, and if gets accepted by that, then we use the trained classifier to check if the shape is complete. If the classifier classifies the shape candidate as an incomplete shape, the shape candidate is rejected. In our experiments, training a classifier with the incomplete shapes performed better on the merged features than the original space. We believe this is because of the high dimension of the original space (720D), the classifier fails to discriminate between the incomplete and complete shapes. The parameter of the SVM classifier for RBF kernel are $C = 3$ and $\gamma = 0.5$, which are determined by a grid search. We used the rejection methods that achieved the best results on the merged features, (i.e., Hellinger + correlation and Dice + Correlation), and incorporated the classifier

		Merged Features			Merged and Trained with Incompletes		
	Method	**TPR**	**TNR**	**AUC**	**TPR**	**TNR**	**AUC**
FC	1	99.7%	86.8%	93.2%	99.5%	96.8%	98.1%
	2	99.7%	88.2%	93.9%	99.5%	96.7%	98.1%
FA	1	99.8%	88.3%	94.0%	99.5%	98.5%	99.1%
	2	100.0%	90.0%	95.0%	99.7%	98.5%	99.1%
Flowchart	1	98.6%	97.1%	97.8%	98.6%	99.4%	99.0%
	2	98.1%	96.1%	97.1%	97.6%	99.5%	98.5%
Class	1	99.4%	91.3%	95.3%	91.2%	98.8%	95.0%
	2	98.8%	91.0%	94.9%	90.6%	98.7%	94.7%
Digital	1	98.0%	91.9%	94.9%	91.0%	93.0%	92.2%
	2	96.8%	91.0%	93.9%	90.3%	94.0%	92.1%

Table 5.8: The proximity-based rejection output using Dice + Correlation (method 1) and Hellinger + Correlation (method 2) on merged feature images, and for the case that a classifier is trained with incomplete shapes.

with incomplete rejection capability.

As can be seen in Table 5.8, having a classifier to reject incomplete shapes has improved the TNR for all datasets, although the TPR has decreased. The decrease in TPR for FC, FA and flowchart dataset is negligible, which with the increase in TNR has eventually resulted in a better AUC. For the class diagram dataset, there is a large decrease in TPR (about 8%) and a large increase in TNR (about 7%), which has resulted in a slightly lower AUC. For the digital circuit dataset, although the TPR has a large decrease (about 6%), the TNR has a slight increase, which has resulted in a lower AUC. The results also show that when incomplete shape rejection is used, the Dice + Correlation method achieves comparable or better results than the Hellinger + Correlation method (see Table 5.8).

We fit a linear mixed effects model on the combined methods on the original features, merged ones and the method of Dice + Correlation on the merged features that is also using a classifier for rejection. The results show that there is a significant difference in the chosen methods ($p < 0.05$). We fit ANOVA model and perform TukeyHSD test for pairwise mean comparison. The results show that the Hamming + Cosine on the merged features is worse than the Dice + Correlation on merged that uses a classifier for incomplete rejection. Removing the Hamming + Cosine results and refitting linear mixed effects model shows that there is no significant difference among the rest of methods.

		FC	FA	Flowchart	Class	Digital
One-class SVM	TPR	91.4%	96.1%	82.4%	87.1%	79.4%
	TNR	67.3%	91.6%	98.9%	96.7%	94.4%
	AUC	79.4%	93.8%	90.7%	91.9%	86.9%
Dice + Correlation	TPR	99.5%	100.0%	95.2%	98.8%	91.5%
	TNR	89.4%	95.3%	95.0%	87.6%	94.0%
	AUC	94.5%	97.6%	95.1%	93.2%	92.8%

Table 5.9: The results of rejection using one-class SVM compared to that of Dice + Correlation (in the original space)

	FC	**FA**	**Flowchart**	**Class**	**Digital**
TPR	97.1%	99.2%	85.2%	-	57.5%
TNR	91.0%	96.7%	88.3%	-	96.4%
AUC	94.0%	98.0%	86.8%	-	76.9%

Table 5.10: The results of rejection using classifier's confidence level.

Results of the One-Class SVM rejection

We evaluated the use of one-class SVM for rejection. This requires training a one-class SVM for each class in the domain. For a shape candidate, if all the classifiers reject the candidate, it gets rejected. In our experiments, we achieved the best results on the original space (720 dimensions) with Linear function with $Nu = 0.01$, $C = 3$. As can be seen in Table 5.9, one-class SVM achieves a lower TPR and AUC for all domains compared to the proximity-based rejection method (for example compared to the Dice + Correlation in the original space). The TukeyHSD test results show that the one-class SVM rejection method is significantly worse than other methods (i.e., the combined methods of original space, merged ones and using incomplete rejection).

Results of the Rejection with Classifier's Confidence

In the next experiments, we used the classifier's confidence for rejection. It should also be noted that this approach cannot be used for domains with a single class (such as class diagram), as the classifier needs to have multiple classes to learn about. The results of this method can be seen in Table 5.10. The results are obtained on the original features (720 dimensions) with Linear kernel and $C = 3$. The TukeyHSD test results show that this method is significantly worse than other approaches.

Summary

In this section, we evaluated various rejection methods (with different metrics) in isolation on the original features and the merged feature representation. The results showed that no single approach has outperforms all other approaches in all evaluation metrics

		FC	FA	Flowchart	Class	Digital
Hellinger + C + O	TPR	99.5%	100.0%	95.2%	98.8%	91.5%
	TNR	90.2%	96.9%	94.2%	85.0%	93.2%
	AUC	94.8%	98.4%	94.7%	91.9%	92.4%
Dice + C + M	TPR	99.7%	99.8%	98.6%	99.4%	98.0%
	TNR	86.8%	88.3%	97.1%	91.3%	91.9%
	AUC	93.2%	94.0%	97.8%	95.3%	94.9%
Dice + C + incomplete rejection + M	TPR	99.5%	99.5%	98.6%	91.2%	91.0%
	TNR	96.8%	98.5%	99.4%	98.8%	93.0%
	AUC	98.1%	99.1%	99.0%	95.0%	92.2%

Table 5.11: The summary of selected rejection methods for evaluation of grouping and recognition. Correlation (C), Original space (O), Merged space (M)

(TPR, TNR and AUC) on all datasets. The TukeyHSD test results also show that there is no significant difference in the majority of the rejection methods. In the next section, we pick some of the rejection methods and incorporate them in the whole grouping and recognition system.

5.3.4 Evaluation of the Grouping and Recognition Using Different Rejectors

In this section we evaluate the grouping and recognition with different rejection methods on the development datasets. From the methods on the original features and the merged ones, we picked the ones that achieved the best average AUC on the five datasets. We also picked the method of using classifier for rejecting incompletes, as it achieved best overall AUC. The summary of these methods with their evaluation results in isolation can be seen in Table 5.11. The one-class SVM and the classifier's confidence for rejection methods performed significantly worse results, therefore, we do not include them in the experiments of this section.

One of the techniques we used to deal with the incomplete shapes is to use image masking during grouping and recognition to find the missing part of an incomplete shape candidate (described in Section 5.2.3). This method looks for the neighbouring strokes of the accepted shape to find the missing part of the incomplete shape; hence, this method could not be evaluated in isolation. In this section, we evaluate each method by excluding this functionality to see its affect on the final results. Therefore, we report the results for six different cases, which are listed in the following:

- **Method 1:** Hellinger + Correlation on the original space, with image masking process.
- **Method 2:** Hellinger + Correlation on the original space without the masking

	1	2	3	4	5	6
Terminator	94.6%	88.7%	89.7%	83.3%	92.6%	87.7%
Data	94.9%	87.4%	93.5%	85.7%	90.8%	89.8%
Decision	76.3%	74.4%	70.1%	61.6%	73.9%	75.4%
Process	97.8%	89.7%	96.6%	87.2%	95.1%	92.6%
Connection	97.6%	97.6%	98.4%	98.4%	98.4%	98.4%
Overall	92.9%	87.2%	90.4%	83.0%	90.4%	88.8%

Table 5.12: The evaluation of grouping and recognition with the six different rejection methods on the FC dataset

process.

- **Method 3:** Dice + Correlation on the merged features, with image masking process.
- **Method 4:** Dice + Correlation on the merged features without the masking process.
- **Method 5:** Dice + Correlation with a classifier trained with incomplete rejection, and with image masking process.
- **Method 6:** Dice + Correlation with a classifier trained with incomplete rejection, and without the masking process.

Tables 5.12 - 5.16 show the evaluation result of the grouping and recognition of the above methods on the development datasets. We calculate the accuracy of each class by dividing the number of correctly grouped and accepted shapes by the total number of shapes of that class. Similarly, the overall accuracy shows the portion of correctly grouped and accepted shapes in that dataset. Based on the overall accuracy, the results show that the Hellinger + Correlation with the original features (method 1) has outperformed the other approaches for the FC and FA datasets. For the class, flowchart and digital circuit datasets, the Dice + Correlation with the merged features has outperformed other approaches.

The results also show that using image masking for finding missing part of the shape always improves the results, even for the case that a classifier with incomplete shapes is trained (see method 5 and 6 in Tables 5.12 - 5.16). Using a classifier for rejecting incomplete shapes always improves the results compared to the case that it is not being used. This can be seen by comparison of methods 4 and 6 in Tables 5.12 - 5.16, in which method 6 uses the same feature and metrics as 4, but with addition of incomplete shape rejection.

We carried out t-test for the three different methods (method 1,3 and 5), and the results showed that there is no significant difference among the three different methods

	1	2	3	4	5	6
State	98.2%	98.2%	94.7%	93.7%	95.8%	94.7%
Final State	97.7%	96.1%	93.0%	92.2%	96.1%	95.3%
Overall	98.1%	97.6%	94.2%	93.2%	95.9%	94.9%

Table 5.13: The evaluation of grouping and recognition with the six different rejection methods on the FA dataset

	1	2	3	4	5	6
Start End	92.1%	92.1%	89.5%	89.5%	89.5%	89.5%
Process	98.1%	65.7%	96.3%	92.6%	96.3%	92.6%
Decision	80.6%	53.2%	93.5%	67.7%	91.9%	88.7%
Overall	91.8%	66.8%	94.2%	84.6%	93.8%	90.9%

Table 5.14: The evaluation of grouping and recognition with the six different rejection methods on the flowchart dataset

	1	2	3	4	5	6
Class	80.5%	2.9%	83.3%	8.0%	82.2%	79.3%

Table 5.15: The evaluation of grouping and recognition with the six different rejection methods on the class diagram dataset

	1	2	3	4	5	6
Start	87.3%	87.3%	84.8%	82.3%	83.5%	82.5%
NOT	55.0%	65.0%	70.0%	75.0%	65.0%	65.0%
XOR	23.8%	28.6%	42.9%	9.5%	33.3%	33.3%
NOR	68.2%	45.9%	50.0%	45.5%	45.5%	45.0%
AND	57.1%	47.6%	52.4%	42.9%	47.6%	47.6%
OR	55.0%	20.0%	35.0%	20.0%	30.0%	25.0%
XNOR	33.3%	27.8%	44.4%	38.9%	44.4%	44.4%
NAND	61.9%	52.4%	76.2%	61.9%	76.2%	76.2%
Overall	64.0%	58.1%	64.4%	56.3%	61.3%	60.8%

Table 5.16: The evaluation of grouping and recognition with the six different rejection methods on the digital circuit dataset

for flowchart, class and digital circuit datasets. However, for the FC dataset, method 1 is statistically significantly better than method 5 ($p < 0.05$), and for FA dataset, method 1 is statistically significantly better than method 3 ($p < 0.05$).

Overall, since using incomplete shape rejection method did not achieve the best results on any of the development datasets, we will not be using this approach in the evaluation chapter. In addition, since using the merged features (with Dice + Correlation metrics, i.e. method 3) has achieved significantly lower results than the original features (with Hellinger + Correlation metrics, i.e. method 1) on FA dataset, we will not use this approach in the evaluation chapter. Therefore, we will be using the Hellinger + Correlation metrics on the original features in the evaluation chapter.

Discussion

Although using a classifier for rejecting incomplete shapes usually achieves better results in isolation (due to higher TNR), it does not achieve the best results when combined with the grouper. One reason is that other approaches deal with the incomplete shapes during grouping and recognition using image masking, which could not be used for evaluation of the rejector in isolation. In addition, we observed that when incomplete shape rejection is being used, some false positives are occurring, which would not happen in other cases.

Figure 5.29 shows an example of a drawn arrow and a data (the parallelogram) in the FC dataset. The grouping algorithm starts with stroke 1 in the shape candidate list (C) and continues expanding that by adding the neighbouring strokes (i.e. strokes 2,3 and 4 one at a time). Everytime a stroke is added to the C, its subsets are checked for a shape, which no shape can be found. When stroke 5 is added to the C and the subsets are checked, an incomplete shape of {3,4,5} gets accepted by the rejector when a classifier for the incomplete shape rejection is not used. Next, stroke 6 is found as a missing part of the shape using the image masking technique and the whole shape gets grouped and accepted correctly. On the other hand, when a classifier for rejecting incomplete shapes is being used, the shape candidate {3,4,5} gets rejected by the classifier as an incomplete shape. When stroke 6 is added, the subsets of C are checked. The shape candidate {2,4,5,6} is hypothesised before the actual shape (i.e. strokes {3,4,5,6}) and incorrectly is being accepted by the rejector. The shape candidate {2,4,5,6} gets accepted because the arrow head (i.e. stroke 2) covers some part of the missing stroke (i.e. stroke 3) due to the blurring process of feature calculation. The blurring of the strokes is carried out on the direction feature images using Gaussian smoothing function (as described in Section 5.1). We also carried out an experiment by reducing the σ of the Gaussian smoothing function to reduce the blurring, but this did not resolve this issue. This issue highlights a limitation of this feature representation. We believe such cases can

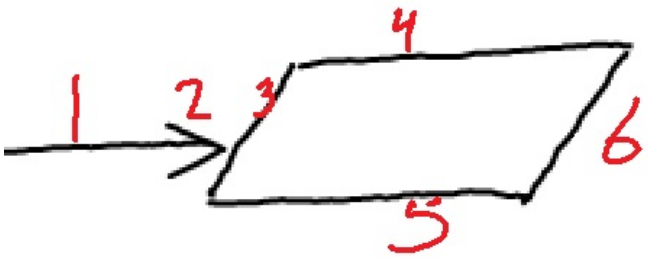


Figure 5.29: An example of a drawn arrow and process in the FC dataset

	Accuracy	**Missing Stroke**	**Extra Stroke**	**1 Error**	**Rejector TPR**
FC	92.9%	95.0%	95.0%	97.1%	99.5%
FA	98.1%	98.8%	98.8%	99.5%	100%
Flowchart	91.8%	92.8%	92.8%	93.8%	95.2%
Class	80.5%	83.9%	88.5%	92.0%	98.8%
Digital	64.0%	73.0%	73.0%	82.0%	91.5%

Table 5.17: The result of different measurement metrics for accuracy on the development datasets

be solved in the future by a backtracking strategy that will be discussed in Chapter 8.

The reported accuracies in Tables 5.12 - 5.16 are based on a strict method of perfect grouping and acceptance of the shapes. This means that if an accepted shape has even one missing or extra stroke, it is counted as a false positive. We use three other metrics to measure the performance of our grouping and recognition system. *ExtraStroke* is the percentage of the shapes that are correctly grouped and accepted as well as the shapes that erroneously have one extra stroke. *MissingStroke* is the percentage of the correctly accepted shapes and the ones that have one missing stroke. *1Error* shows the percentage of the correctly accepted shapes and the ones that have either a missing stroke or an extra stroke (not both at the same time). In Table 5.17 we report the result of the accuracies of the strict perfect grouping and recognition, the *ExtraStroke*, *MissingStroke*, *1Error* and the TPR of that approach. The reported results in Table 5.17 are with the Hellinger + Correlation metrics on the original features (method 1), as this is our choice of evaluation for Chapter 7.

The results show that if we allow one error (either missing or extra stroke), 92.0% to 99.5% of the shapes in FC, FA, flowchart and class diagrams are being found, which are close their rejector's TPR. This implies that the majority of the shapes are being correctly located; however, a missing or extra stroke affects the perfect grouping and recognition results. We realised some of the missing strokes are touch up strokes that the image masking process cannot pick them as a missing part (see Figure 5.30 for

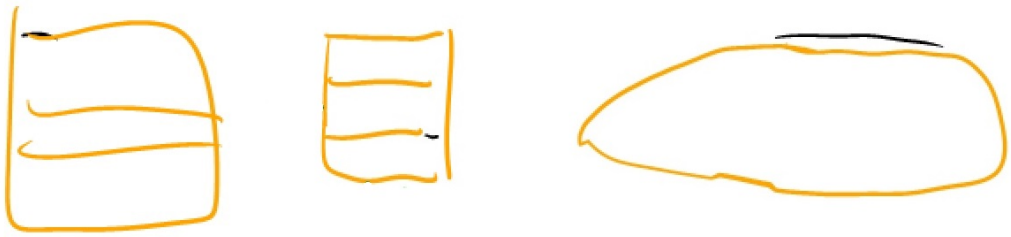


Figure 5.30: Some examples of touch-up strokes (shown in black) that are not recognised as part of the shape. The strokes in orange are the ones that are grouped together and are accepted as a single shape by the rejector.

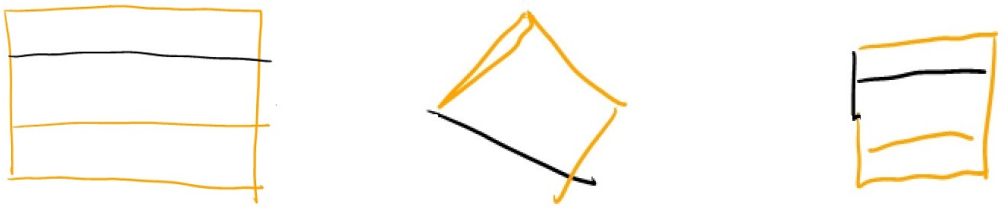


Figure 5.31: Some examples of missing strokes (shown in black) that are not picked in the image masking process. The strokes in orange are the ones that are grouped together and are accepted as a single shape by the rejector.

some examples of this case in different diagrams). There are also cases that the image masking does not find all the missing strokes (see Figure 5.31 for examples of this case). We also realised that in many cases part of a connector is attached to a shape and all strokes together get accepted (see Figure 5.32 for some examples of this case).

Allowing one error for the digital circuit dataset, increases the accuracy by 18%, however, it is still relatively low compared with the rejector's TPR. After observing the results, we realised that there are some cases with more than one error. See Figure 5.33 for some examples of more than one error. For these cases, the shape is being correctly located, however, some small strokes are attached to the shape, which does not make it different enough to be rejected.

5.4 Summary

In this chapter we first described the details of the recogniser that we used, followed by some modifications to the feature representation that lead to better results. Next, we introduced a novel proximity-based rejection method that performs rejection using novelty detection methods. This is done by forming a cluster around each class and using different distance and similarity metrics to measure the proximity of an input to a cluster centre. We also introduced two different techniques for dealing with the

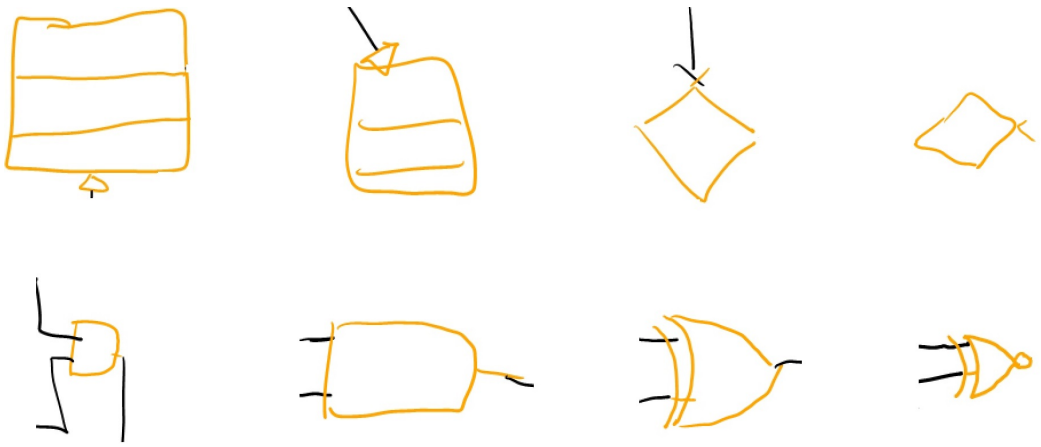


Figure 5.32: Some examples of an extra stroke that is attached to the shape. The strokes in orange are the ones that are grouped together and are accepted as a single shape by the rejector.

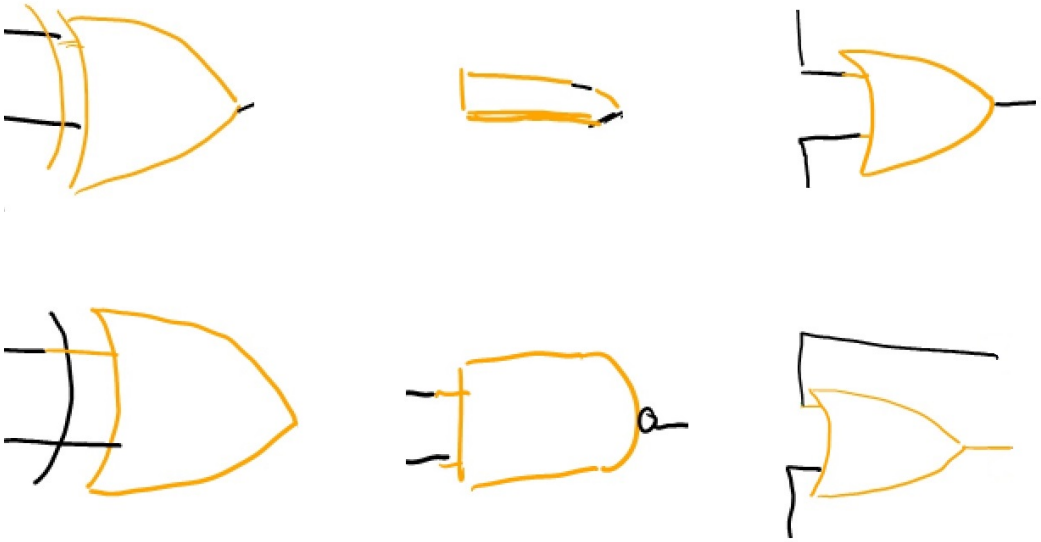


Figure 5.33: Some examples of having more than one error.

incomplete shapes. In the experiments, we evaluated each rejection method in isolation with different feature representation using different similarity and distance metrics. We also compared the results with two other techniques for rejection, which are using classifier's confidence and using one-class SVM. The rejector's evaluation results show a high accuracy in rejecting invalid shape candidates and accepting the valid ones. Finally, we incorporated some of the best performing rejection methods of different feature representations with the grouper followed by a discussion on the performance of the system. The results showed that the Hellinger + Correlation metric on the original feature space that uses image masking performs better.

Chapter 6

Connector Localisation and Grouping

In our sketch recognition system, the connector recognition is carried out after the shape recognition. So the input is a set of recognised shapes, and the leftover strokes, which are connectors and errors. In this step, the connectors need to be detected and their strokes grouped together. In this chapter we describe our approach towards localising connectors and grouping their strokes together. The connector recognition is not a trivial task since the appearance of the connector shaft can vary markedly. Regardless of how shafts are drawn, all the directed connectors (the type of connector that has a head at one end only) have one part in common, the connector head. Figure 6.1 shows an example of three drawn arrows with different shafts, but similar heads (the heads are highlighted with a red box around them).

Our approach towards connector recognition is to learn about the connector heads, not the whole connector since the common part is the head. In this chapter we will describe how we learn about the connector heads, followed by a description of a grouping algorithm that groups the shaft strokes. We have used flowcharts (Stevens et al. 2013) as a choice of dataset for illustration of each step of connector grouping and recognition due to its simplicity. We will show the experimental results on the development datasets (see details on these set of data in Section 3.4), followed by a discussion of the limitations of our approach.

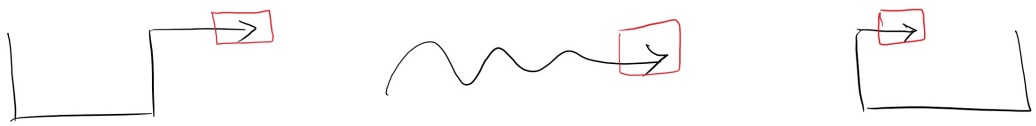


Figure 6.1: An example of three drawn arrows with different shafts but similar heads.

6.1 Training: Connector Head Localisation

Our aim of training a connector recogniser is to learn only about the connector heads and to be able to reject any other candidate. The classification of the heads is not necessary for our purpose of sketch recognition, as we only require to localise them. If there are different connector heads with different semantics, we could use a classifier to distinguish them from each other, however, this is not the case for the datasets we use in this thesis. In this chapter, when we refer to the connector recogniser, we are referring to the connector head localisation, which is the process of finding a valid connector head. In the following, we describe how we train the connector recogniser.

We use the same feature representation and same steps of forming clusters and finding thresholds as described in Chapter 5. The described feature representation is rotation variant, but the connector heads could be drawn in any orientation. We therefore rotate the connector heads to point to a predefined direction. Training the connector recogniser requires several steps: finding the connector heads of the training set, rotating them, and forming the cluster. In the following, we describe how we detect the location of the head, followed by a description of the rotation process.

6.1.1 Connector Head Detection

To form the cluster for the connector heads, we need some training samples. Usually a labelled dataset only contains labels for the whole connector, not the head area. It is difficult to only label the connector head area that we want to learn about (the highlighted one with a red box on the right in Figure 6.1). Therefore, we need an automated system to identify these heads to be used for training.

Each connector might be drawn with single or multiple strokes. We assume the head is located around one of the endpoints of the shaft strokes. We put a bounding box centred at the connector strokes' endpoints and capture all the connector stroke points that would fall within that bounding box; each representing a head candidate. For each connector, we would have $2N$ connector head candidates (where N is the number of drawn strokes for that connector). Figure 6.2 shows an example of a connector in the flowchart dataset drawn with two strokes, one for the shaft and one for the arrow head. In this figure, the two bounding boxes fitted around the two endpoints of the shaft stroke are shown in red.

Each arrow head candidate is turned into its five feature images as described in Section 5.1 (without the smoothing and downsampling steps to have a clearer representation of the candidate). We then merge these five images representing the head candidate and pick the head candidate image with the highest number of pixels in it. An alternative would be to count the number of stroke points in the bounding box but

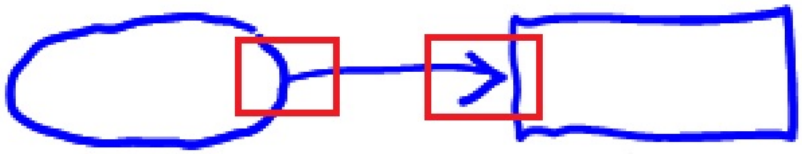
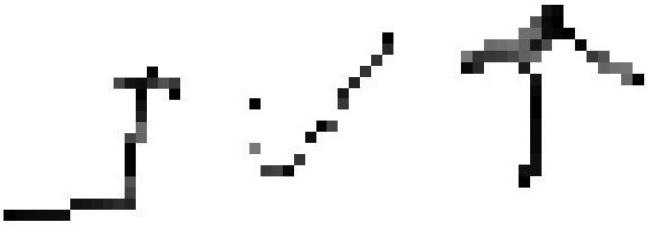


Figure 6.2: An example of the two bounding boxes fitted around connector ends.



(a) Drawn arrow (b) Arrow Head Candidate 1 (c) Arrow Head Candidate 2

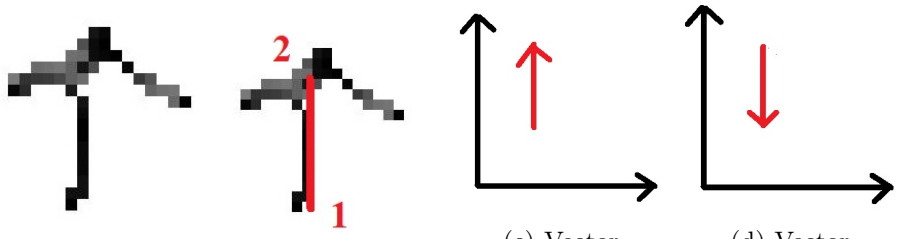
Figure 6.3: (a): The merged feature image of a drawn arrow, (b): The merged feature image of the bounding box around the bottom-left endpoint, (c): The merged feature image of the bounding box around the top-right endpoint

this would not always identify the correct head candidate because some touch-up strokes might occur. Figure 6.3a shows the merged image of an example of a drawn arrow with a single stroke in the flowchart dataset, and the merged images of the two endpoints of the arrow are shown in Figures 6.3b and 6.3c. In this example, the second arrow head candidate (Figure 6.3c) has the higher number of pixels, so is chosen as representing the head candidate.

The size of the bounding box is a sensitive parameter. If it is too large we might have some parts of the shaft that is not needed for training, and if it is chosen too small, the whole head would not fit in the box. We chose this parameter experimentally for the development datasets.

6.1.2 Rotation of Connector Heads

The identified connector heads can point to any direction. Therefore, we need to rotate all the selected connector heads to point to the same direction; e.g. all pointing to the right (for example, the selected arrow head in Figure 6.3c needs to be rotated about 90° clockwise). For this we need to find the shaft line and the location of the head to identify its direction. Once we know the direction of the connector head, we can rotate it to point in the pre-defined direction. This requires three steps: (*i*) finding the shaft line, (*ii*) identifying the direction and (*iii*) rotation.



(a) The head candidate (b) A line on the shaft (c) Vector direction from point 1 to point 2 (d) Vector direction from point 2 to point 1

Figure 6.4: A head candidate merged image drawn in the flowchart diagram.

Finding the Shaft Line The Hough transform is a widely used line detection algorithm (Duda & Hart 1972, Illingworth & Kittler 1988) in the computer vision and image processing community. We find the lines using the Hough transform algorithm provided in EmguCV library (*EmguCV* 2019) (a .NET wrapper to the OpenCV library (*OpenCV* 2019)). We consider the longest detected line as the arrow shaft.

Identifying the Direction If we think of the shaft as a vector, identifying its direction allows us to determine the angle we need to rotate to point to the desired direction. For example, Figure 6.4a shows the merged image of a connector head from the flowchart dataset. Figure 6.4b shows a line that could fit on the shaft with its two ends (1 and 2). If we think of the line as a vector, Figure 6.4c shows if point 1 is picked as the tail of the vector and Figure 6.4d shows if point 1 is picked as the head of the vector. We assume from the two endpoints of the shaft line that the one that has a higher density of pixels around it is where the head is located. We put a 5×5 window around the two endpoints and pick the one that has a higher number of pixels in it. We count the number of pixels by binarising the image with an intensity threshold value more than 0, regardless of what the intensity value is.

Rotation Once we know the direction of the connector, we can calculate how much it should be rotated to point in the desired direction. We rotate the head in the point level (not its image representation) to make sure no information is lost. Figure 6.5 shows some of the arrow heads, the fitted line of the shaft using the Hough Transform algorithm and the rotated heads in the flowchart dataset.

6.1.3 Forming the cluster

In this stage, we have all the connector head candidates pointing in the same direction that we need to learn about. We use the method described in Section 5.2.1 to form the cluster and find thresholds. In addition, similarly to regular shapes, we perform a

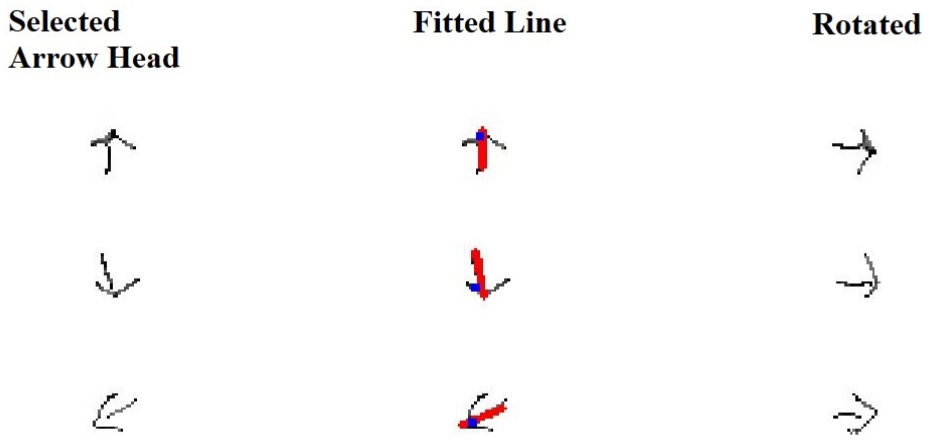


Figure 6.5: Examples of selected arrow heads (the first column) with the fitted line in red (the second column) and the rotated head (the last column)

pre-processing outlier detection within a cluster before finding the cluster thresholds.

6.1.4 Summary of Connector Head Training Process

To summarise, we perform the following to train a recogniser that can accept or reject the head candidates:

1. Find the connector heads in a labelled dataset.
2. Rotate all connector heads to point to the same direction.
3. Form clusters around connector heads.
4. Perform outlier detection.
5. Find the thresholds for clusters.

6.2 Connector Grouping

The connector grouper is responsible for finding the connectors in a diagram and putting the strokes of each connector together. The connector grouper works simultaneously with a recogniser (described in the previous section) for validation of a connector candidate. Our approach is to find connector heads around a shape and then find the remainder of the shaft strokes. The input to the connector grouper is a list of unrecognised strokes (U) and recognised shapes (S).

We assume the unrecognised neighbouring strokes (N) of a shape belong to a connector, hence, we perform a search around each recognised shape. For each recognised

shape, we put a Bounding Box (BB) centred at the endpoints of each stroke in N and capture any point from strokes in U that would fall within the bounding box. In this step, we only include the points from the strokes that have at least one endpoint in the bounding box. We treat the BB around each endpoint as a connector head candidate, therefore, each BB is turned into its merged feature image and rotated to point in a predefined direction (same direction used in the training process). The head candidate is then given to the connector recogniser for validation. If a head candidate is accepted by the recogniser, we then look for the rest of the shaft strokes. The pseudocode for the connector grouping and recognition can be seen in Algorithm 3.

We assume the connector strokes (CS) that fall within the bounding box include the head stroke(s) and part of the shaft (if not all of the shaft). We look for the rest of the shaft strokes in U that have an endpoint close to an endpoint of strokes in CS. If a stroke (u) in U has an endpoint close to any of the strokes' endpoints in CS, we add it to CS. We can continue this process until we reach another shape. However, since the other end of the connector might be a shape that is not recognised (a false negative), we stop when we cannot add any more strokes to the CS. The pseudocode for finding the shaft strokes can be seen in Algorithm 4.

Algorithm 3 Connector Grouping and Recognition

Input:

$S =$ {List of recognised shapes},

$U =$ {List of unrecognised strokes},

Output: S, U, C {S: List of shapes, U: List of unrecognised strokes, C: List of recognised connectors}

loop over elements of S: s_i
    N = Spatial neighbours of s_i in U
    loop over elements of N: n
        Put a bounding box around both endpoints of n and pick the more dense one. Label it e.
        if (e is a valid connector head) **then**
            CS = Find the shaft strokes (see Algorithm 4)
            $C = C \bigcup \{CS\}$
            $U = U \setminus \{CS\}$
        end if
    end loop
end loop
Return S , U , C

Algorithm 4 Find Shaft Strokes

Input:

$CS =$ {The list of connector head strokes},

$U =$ {List of unrecognised strokes},

Output: CS {CS: The list of connector strokes}

```
1: while (new stroke is added to CS) do
2:     loop over elements of CS: c
3:         loop over elements of U: u
4:             if (c has an endpoint near either of u's endpoints) then
5:                 CS = CS ⋃ {u}
6:                 U = U \ {u}
7:             end if
8:         end loop
9:     end loop
10: end while
11: Return CS, U
```

Figure 6.6 shows an example of a drawn ellipse connected to a square. The ellipse and the square are drawn with a single stroke, while the connector is drawn with 4 strokes. Assuming the shapes (i.e. the ellipse and the square) are already recognised from the previous step, the connector recogniser tries to find the connectors around these shapes. The algorithm starts by checking for connector head around the first recognised shape (i.e. the ellipse). In step 1, shown in Figure 6.7a, stroke 2 is picked as the neighbouring stroke of the ellipse. In step 2, a Bounding Box (BB) is put around both endpoints of stroke 2. Step 3, in Figure 6.7a shows the BB with higher number of points (from unrecognised strokes) is picked and is given to the recogniser as a connector head candidate. Since this head candidate is not a valid connector head, the recogniser rejects it. Since there are no more strokes in the neighbouring list, N, the algorithm picks the next recognised shape and checks for a head close to it (see step 4 in Figure 6.7b).

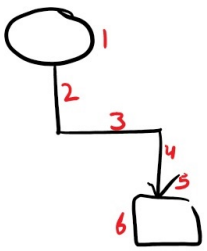
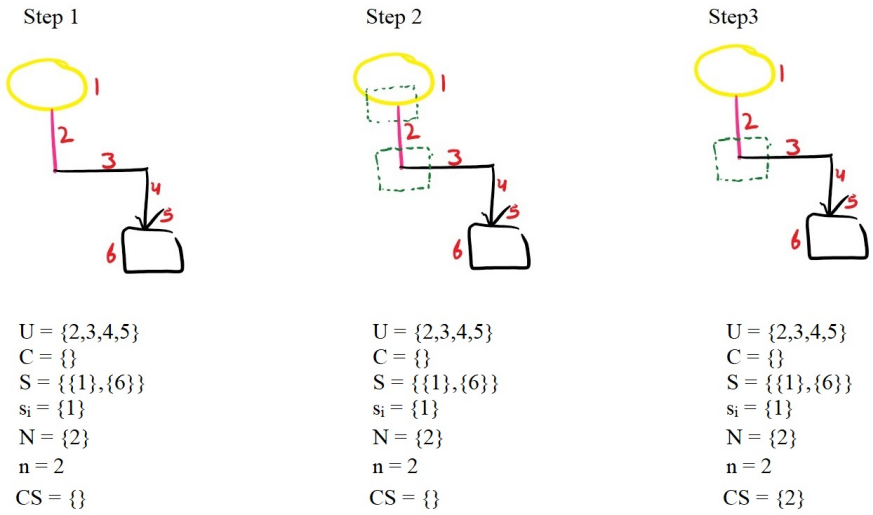
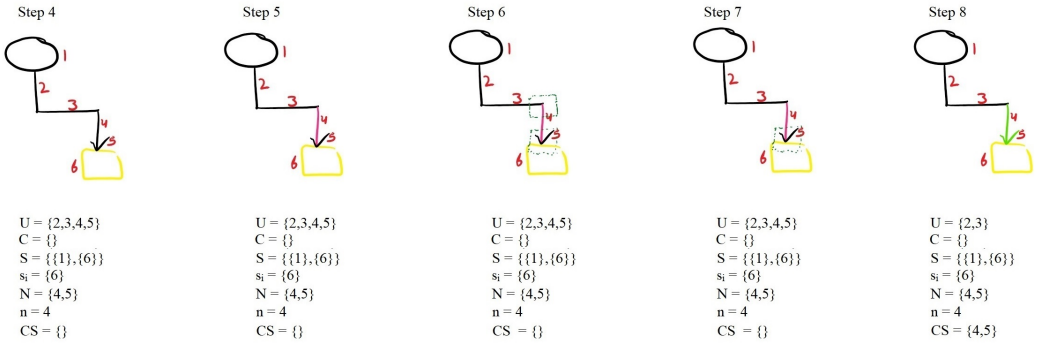


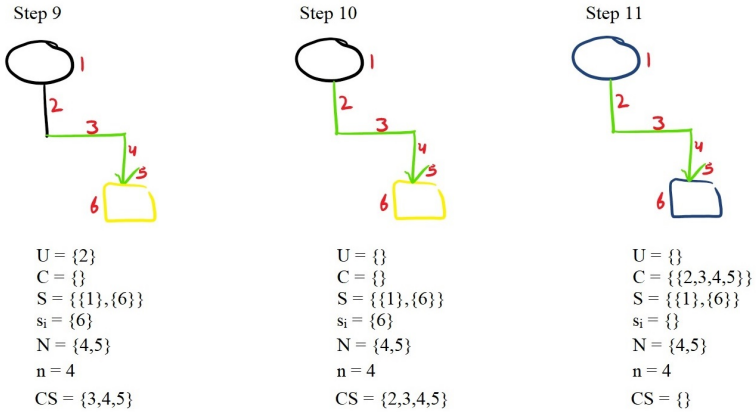
Figure 6.6: An example of a drawn sketch



(a) Steps 1 to 3 of the connector recognition algorithm



(b) Steps 4 to 8 connector recognition algorithm



(c) Steps 9 to 11 connector recognition algorithm

Figure 6.7: Different steps of connector recognition algorithm.

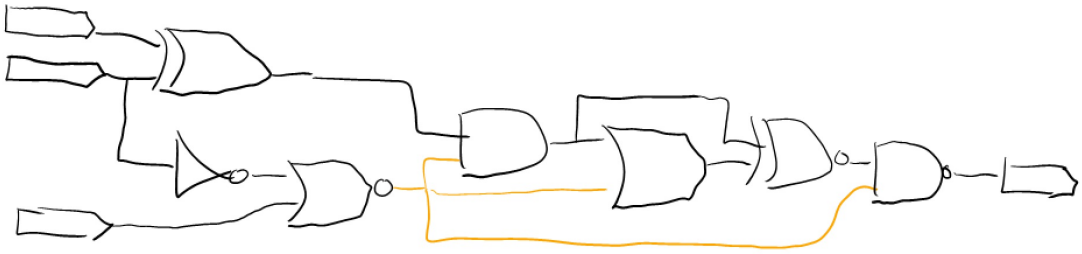


Figure 6.8: An example of a digital circuit diagram with a connector that connects multiple shapes (the strokes shown in orange)

	FC	FA	Class	Flowchart
#Strokes	2736	1501	477	512
#Connectors	1260	796	190	249

Table 6.1: The details connectors in development datasets

In step 5, shown in Figure 6.7b, the neighbouring strokes of the square are picked $N = \{4, 5\}$. In step 6, stroke 4 is picked as the head of the list and a BB is put around both endpoints. Step 7 shows the selected BB with the higher number of points. Since this is a valid head candidate, this should be accepted by the recogniser. The strokes that have fallen in the BB are 4 and 5, which are moved to the Connector Stroke (CS) list from U in step 8. The algorithm tries to find the rest of the shaft strokes.

In step 9 shown in Figure 6.7c, stroke 3 is added to the CS since it has an endpoint close to an endpoint of stroke 4, and consequently is removed from U. In step 10, stroke 2 is moved from U to the CS as it has an endpoint to that of stroke 3. At this point no more strokes can be added to the CS. Hence, in step 11, the {2,3,4,5} connector is added to the connector list (C) and all its strokes are removed from U. At this point all the strokes are recognised and the algorithm stops.

6.3 Experiments

In this section we evaluate the performance of our connector recognition system on the development datasets. We exclude the digital circuit dataset from the experiments of this section since our connector recogniser does not support this diagram's connectors. This is because the connectors in digital circuit diagram are undirected (no head) and also a connector in this domain will often connect multiple shapes together. For example, Figure 6.8 shows an example of a digital circuit diagram where a connector is connecting multiple shapes together. The details of connectors in the remaining development datasets can be seen in Table 6.1.

For the experiments of this section, we carried out 5-fold user holdout cross validation. Similar to the regular shapes, we use the original feature representation (the 720 dimensions), with the Hellinger + Correlation as the choice of proximity metrics. The size of bounding box is set to 700×700 himetric units. The distance threshold between the endpoints of the connector strokes is also set to 300 himetric units. These parameters are chosen empirically. For these experiments, we evaluated the connector localisation and grouping system in isolation. For that, we have perfectly grouped and recognised all the shapes, and only evaluate the connector recognition system's performance. Our connector recogniser system can correctly identify (localise and group) 88.8%, 86.6%, 82.6% and 84.3% of the connectors in the FC, FA, class and flowchart datasets, respectively. Since there is no reported results for only connectors on these datasets, we cannot make a direct comparison with other approaches.

6.4 Limitations

During the experiments we observed that the connector rejection is not robust enough in rejecting invalid head candidates. This can also be seen in Figure 6.9, which shows the similarity score of each connector head to its cluster centre in the flowchart dataset. Unlike regular shapes, the similarity of the heads to the cluster centre are in a wider range. This is because forming a connector head candidate has different steps and each step can cause an error which results in an unwanted connector head that the recogniser learns about. Figure 6.10 shows some of the examples of incorrect connector heads. Figure 6.10a shows when an incorrect side of a connector is picked as a connector head (see Section 6.1.1). Figure 6.10b shows a case where the fitted line on the arrow head is longer than the shaft, and is incorrectly picked as the shaft, which results in an incorrect direction for the arrow head. Figure 6.10c shows the case where the shaft is drawn with a little bit of angle towards the head. This also results in an incorrect direction for the arrow head after rotation. Figure 6.10d also shows that sometimes the shaft part that is fitted in the bounding box is not just a straight line. In such cases, even though the shaft line is picked correctly, the rotated arrow head is not similar to the rest of the arrow heads in the training set. All the aforementioned errors cause the recogniser to be less robust at rejection.

In addition to the recogniser, the connector grouper has some limitations. For example, when trying to put a bounding box around the connector head, another connector head which is spatially close might fit in the bounding box. Since the rejector is not robust enough, the connector head candidate might not get rejected. Figure 6.11 shows an example of this situation where the two connectors pointing to the same shape and are spatially close and putting a bounding box around one would include the stroke from the other one as well. In such cases two connectors would be missed to be grouped

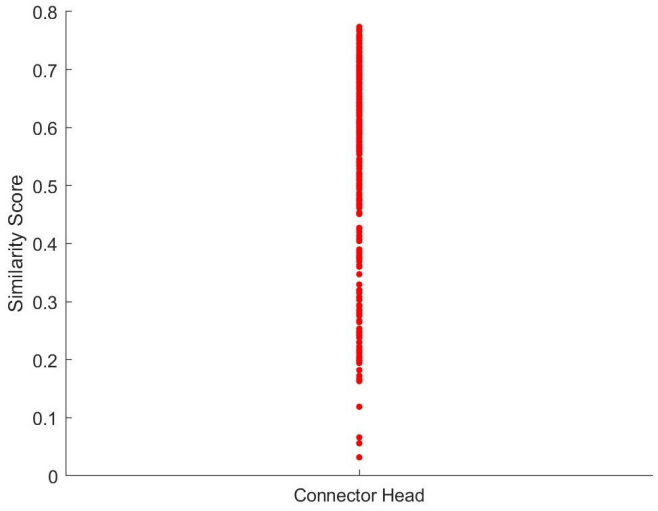


Figure 6.9: The plot of similarities of each connector head to its cluster centre in the flowchart dataset

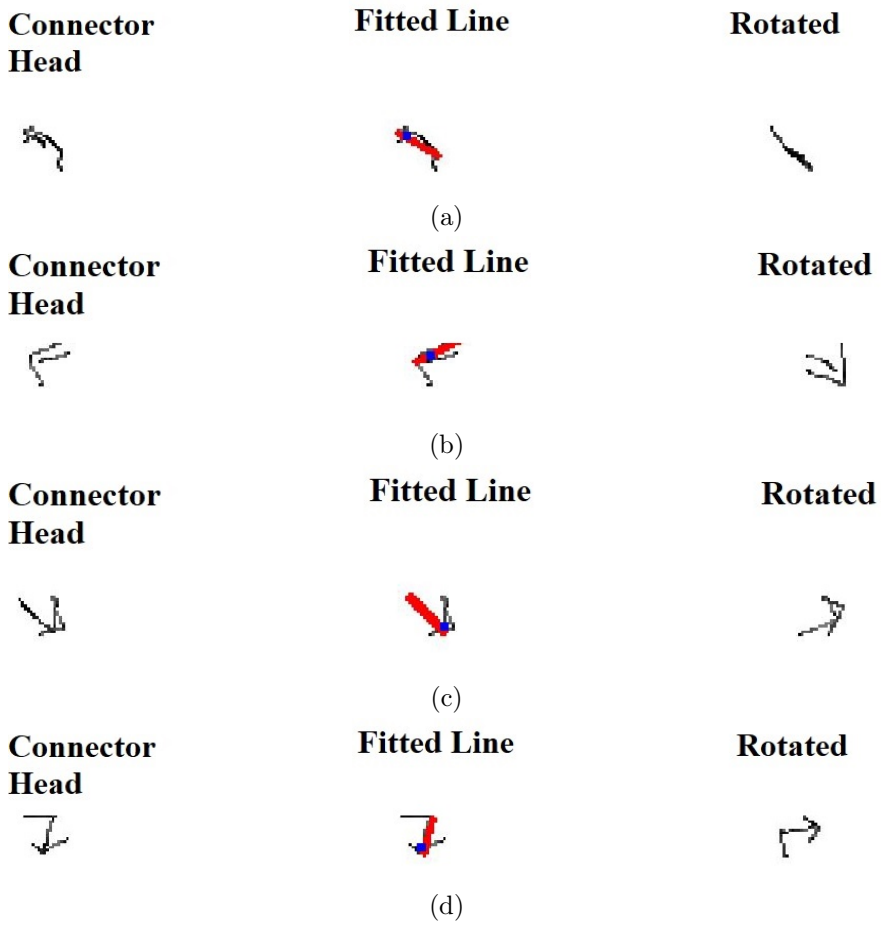


Figure 6.10: Some of the examples of incorrect connector heads

and recognised correctly.

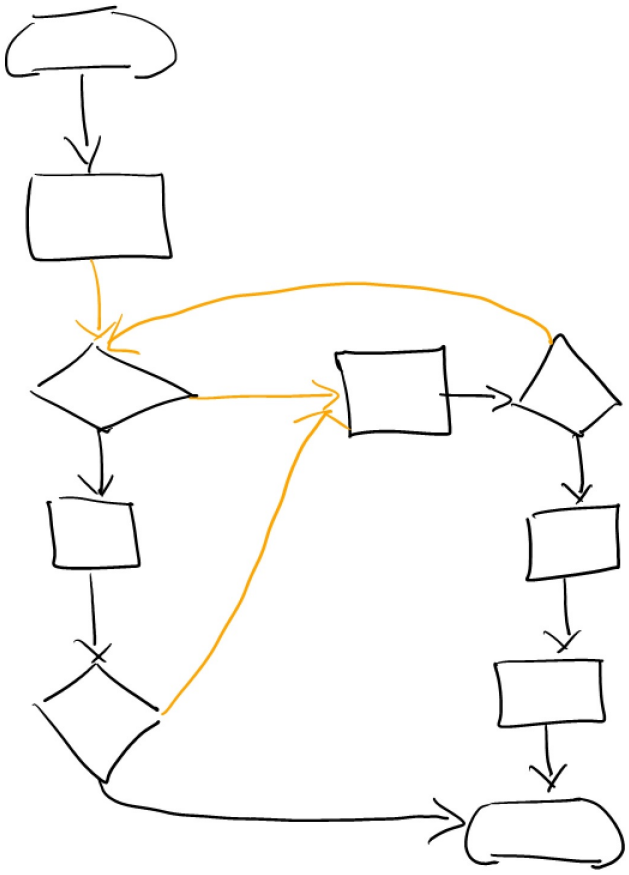


Figure 6.11: An example of a drawn sketch in the flowchart dataset with spatially close arrows (shown in orange colour)

Furthermore, as mentioned earlier, the proposed grouper only works for domains where a connector connects two shapes. For example, in the digital circuit diagram (see Figure 6.8), a connector might connect multiple shapes. In such cases, the grouper would fail to group the connector strokes. Moreover, the proposed connector recognition system works only for directed connectors (having a connector head only at one end). In addition, if two shapes are rejected by the shape rejector, and there is a connector between them, the grouper would not be able to find and recognise the connector.

6.4.1 Summary

In this chapter we introduced a connector recognition system using the direction features introduced in Chapter 5. Our system focuses on learning about the connector heads and grouping the shaft based on the head localisation. Since the choice of feature representation is rotation variant, while connector heads can be drawn in any orientation, we rotate them to point to the same direction. We also introduced our connector

grouping algorithm that tries to find the connectors around the recognised shapes. In the experiments, we showed that the connector recognition is able to correctly find 85.7% of connectors on average in the development dataset.

Chapter 7

Evaluation and Comparison

In the previous chapters we evaluated each component of our recognition system on the development datasets. In this chapter, we evaluate our entire sketch recognition system on new evaluation datasets, that our system has not been tested on before. The goal of our evaluation is to make sure our approach achieves comparable results to other approaches on datasets that have not been used during development. We measure the performance of our system in terms of computation time and accuracy. We first summarise the settings for the different components of our sketch recognition system. Next, we describe a full comparative evaluation of our method against the work of Stahovich et al. (2014). We also report our results for the two publicly available datasets used during development (FC and FA) and compare results from our work with the reported results in the literature.

7.1 Settings

Our approach has several parameters to be set. Table 7.1 shows the numerical values used for these parameters in our system. The settings in Table 7.1 are the same for all the experiments in this chapter. In terms of rejection method, we use the Hellinger + Correlation on the original feature space, since this achieved the best results (see Chapter 5 for the details of this method).

7.2 Full Comparative Evaluation

In Chapter 2, we categorised the existing grouping and recognition methods into two broad categories of simultaneous and sequential. We also highlighted the four important characteristics of a sketch recognition system for each approach. Among the subcategories of these approaches, the hard-clustering approaches (see Section 2.2.2 for the details) have the three important characteristics of a sketch recognition system: being

	Parameter	Purpose	Value
Grouper	DT	Distance threshold between the points of strokes for spatial proximity	600 himetric unit
	Max_s	The initial value for the Max_s	5
	UB	Upper bound threshold for the Max_s	12
Rejection	OD_T	Outlier detection threshold for each cluster	0.65
Connector	DT	Distance threshold between the endpoints of the connector strokes	300 himetric unit
	BB	The size of the bounding box around a connector head	700×700 himetric unit

Table 7.1: Numerical parameters used for the grouper and the rejection system

domain independent, supporting free sketching environment and being computationally inexpensive. One of the hard-clustering methods with these characteristics is the work done by Stahovich et al. (2014), which is our choice of comparison since it has the mentioned characteristics and is simpler to implement compared to other hard-clustering methods. We compare our results with the work of Stahovich et al. (2014) on new evaluation datasets. In this section, we first describe the work of Stahovich et al. (2014), then the evaluation datasets, followed by the results of the comparative evaluation.

7.2.1 The work of Stahovich et al. (2014)

Stahovich et al. (2014) propose a grouping technique that uses two levels of classification. In the first level each stroke is classified into a coarse class. For example, for family tree diagrams (see Figure 7.1), each stroke is classified as shape, connector or text. For this first level of classification, an AdaboostM1 classifier is trained with 27 features characterising the stroke's shape, location, drawing kinematic, size and the relationship between the stroke and other strokes in the sketch.

Once each stroke is classified into one of the coarse classes, the second level classification is performed on each pair of strokes from the same class to determine whether they should be joined to be part of the same shape. They have proposed two techniques for this: threshold pairwise classifier and Inductive Pairwise Classifier (IPC). The threshold pairwise classifier is simple, and only uses distance and time thresholds for the classification. The IPC method uses 13 pairwise features to train an AdaboostM1 classifier to classify each pair either as "No Join", "Near Join" or "Far Join". The *Near Join* label is assigned to the pairs that belong to the same shape and are close to each other. The *Far Join* label is assigned to the pairs that are part of the same shape, but are far from

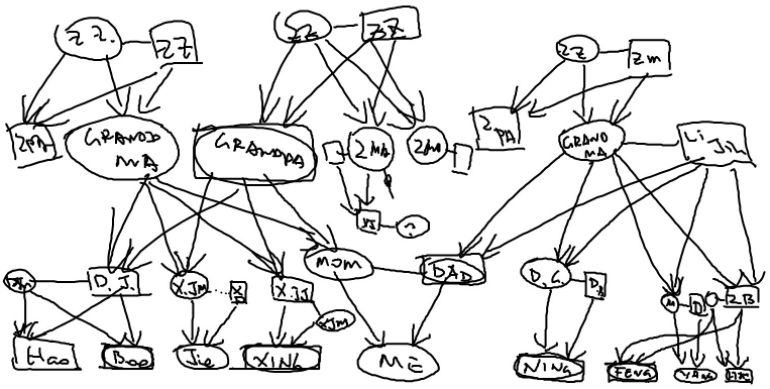


Figure 7.1: An example of a family tree diagram

each other. All other pairs are assigned *No Join* label. Two techniques are proposed to label the pairs in the training set: Minimum Distance (MD), and Iterative Relabelling (IR). The MD method uses a set of thresholds to assign a label to a pair, while the IR method initially uses the MD method to assign a label to the pairs and iteratively modifies the labels to optimise the accuracy of the classifier. Once each pair is classified through the classifier, the pairs with the *Near Join* label are chained together. If a stroke is joined with another stroke that already belongs to a shape, all those strokes are chained to form a single shape.

The experimental results show that in cases where the differences in accuracy were statistically significant, the IPC-IR and IPC-MD methods outperformed the thresholding method while both the IPC-IR and IPC-MD methods achieve comparable performance. For the experiments in this section, we re-implemented the IPC-MD method since is easier to implement and achieved comparable results to the IPC-IR method.

Validation of our implementation of Stahovich et al. (2014)

Since we were not able to obtain the code from the authors, we re-implemented the work of Stahovich et al. (2014) for a full comparative evaluation. In order to confirm the validity of our re-implementation, we aimed to reproduce results in the original paper. The work of Stahovich et al. (2014) has been evaluated on three datasets: family tree diagrams from the EtchaSketches corpus [1], digital circuit diagrams, and static solution sketches. We only have access to the family tree diagrams, which we used for validation of our re-implementation. For their evaluation on the family tree dataset, they excluded sketches with less than five strokes, and sketches that are subsets of other sketches, which results in 27 sketches. We excluded the sketches with fewer than 5 strokes, which resulted in 30 sketches. After observing these 30 remaining sketches, it was not clear which sketches were considered subset of the others in the original

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

research, hence we included all 30 sketches for the experiments. Table 7.2 shows the details of the 27 sketches that are reported in the paper and the 30 sketches that we used in our experiments.

	Paper (27 sketches)		**The 30 Sketches**	
	Strokes	Shape Count	Strokes	Shape Count
Shape	407	293	477	332
Arrow	636	397	743	453
Text	617	107	686	126

Table 7.2: The details of the family tree diagrams reported by Stahovich et al. (2014) for 27 sketches compared to the 30 sketches we experimented on

Stahovich et al. (2014) reported the results on the first level of classification as well as the correctly grouped shapes in the family tree diagrams. In the paper it is mentioned that user holdout cross validation is used, however, the number of folds is not mentioned. User holdout cross validation is a widely used model evaluation method in which data from one user is used for testing and the data from the other users are used for training (Stahovich et al. 2014). We follow the common 80/20 rule (where 80% of the data is used for training and 20% for testing) and perform a 5-fold user holdout cross validation. Since text is out of the scope of our research, we also performed the experiments for the case that the text is manually excluded. The results are reported in Table 7.3.

	Multiway Classification (first level)			**Grouper (both levels)**		
	Original	Ours (text)	Ours (no text)	Original	Ours (text)	Ours (no text)
Shape	83.0%	79.9%	81.7%	85.3%	81.0%	87.6%
Arrow	86.8%	83.9%	94.3%	68.3%	51.0%	46.2%
Text	92.1%	85.5%	-	51.4%	31.7%	-

Table 7.3: Comparison of our re-implementation of Stahovich et al. (2014) work as compared to the reported results. Ours(text): Our implementation results for the case that text is included. Ours(no text): Our implementation results for the case text is manually excluded.

The results in Table 7.3 show that the re-implementation results differ from the original results reported in the paper. We believe this is due to the three additional diagrams in our experiments. In particular, the results from our re-implementation

produces lower rates of classification than the reported results, for all classes. All the experiments using our re-implementation were carried out on 30 sketches, while Stahovich et al. reported their results on 27 sketches. To account for this discrepancy in the datasets used, we re-calculated the results by excluding the three diagrams that had the lowest rates of classification (see Table 7.4). These results are comparable to the reported results in the paper (about 2% higher for shapes, 4% lower for arrows and 1% lower for texts). Since we do not have the details about the results reported in the paper such as average and standard deviations, we are not able to perform paired t-tests for significant testing.

	Paper	**27 best**
Shape	85.3%	87.6%
Arrow	68.3%	64.2%
Text	51.4%	50.4%

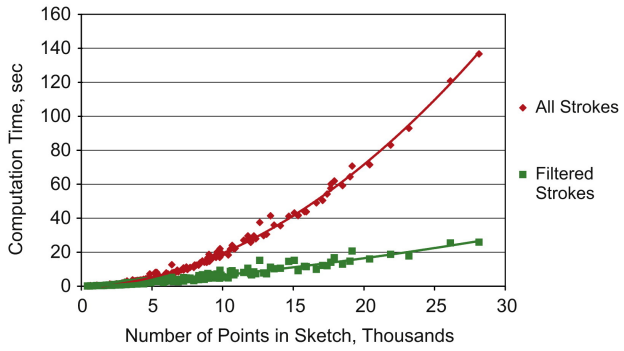
Table 7.4: The results of our implementation of Stahovich et al. (2014) work after excluding three of the sketches compared to the reported results

The computationally expensive part of Stahovich et al.'s approach is where the pairwise features are being calculated. Since the cost of computing pairwise features is determined by pointwise distance calculations on each stroke, the cost of this approach is $O(n^2)$, where n is the number of points in the sketch. In order to reduce this time, the pairwise features are only computed for the pairs of strokes where their bounding boxes are less than 5000 himetric units apart. Figure 7.2a shows the reported computation time of feature computation vs. the number of points in a sketch. The choice of datasets for the plot in Figure 7.2a is not known. Figure 7.2b shows the same computation time plot achieved from our implementation of their work. It should be noted that the number of points in the diagrams we have access to are up to 8000, therefore, only the plot for the range of 0 to 8 on the x axis of Figure 7.2a should be compared with the plot in Figure 7.2b. The plot in Figure 7.2b shows that using the bounding box distance reduces the computation time in our implementation.

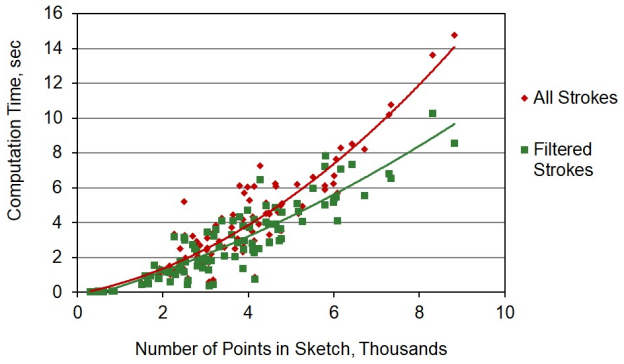
7.2.2 Evaluation Datasets

During the development of our system, we used the five development datasets (described in Chapter 3) to perform initial testing of each component. In this chapter, we use new datasets that have not been used during development. We use these datasets to compare the results of our approach with that of reported by Stahovich et al. (2014).

We chose two connected diagrams for the choice of evaluation dataset: family tree and process diagrams. Connected diagrams are generally more challenging than those



(a) Reported by Stahovich et al. (2014)



(b) Our implementation

Figure 7.2: Computation time for feature computations as a function of points in the sketch

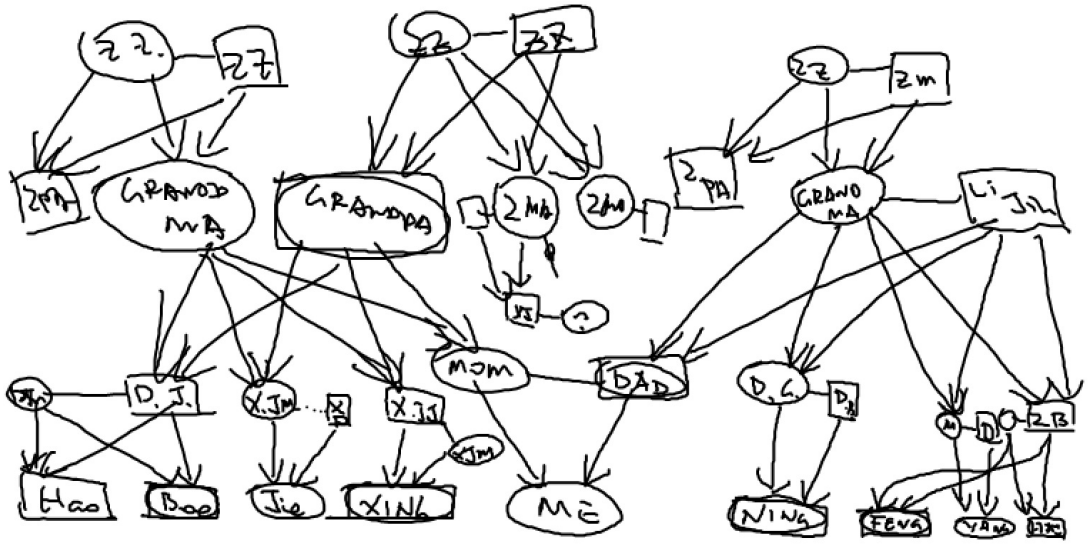


Figure 7.3: An example sketch from the family tree dataset

that are not connected (e.g. user interface sketches), as connectors are spatially close to shapes. In the following we provide details about the evaluation datasets.

Family Tree Diagrams

The family tree diagram is the same that we used for validation of our re-implementation of Stahovich et al. (2014) work. Similar to the work of Stahovich et al. (2014), we exclude all the sketches with less than five strokes, which remains with 30 sketches, collected from 30 participants on a Tablet PC. The dataset is composed of two regular shapes (ellipse and rectangle), text and arrows. Figure 7.3 shows an example diagram from this dataset.

The ground truth for the dataset is provided for three broad classes of shape, connector and text. This information is sufficient for the work of Stahovich et al. (2014). However, in order to evaluate our approach on this dataset, we re-labelled it to provide labels for each of the regular shapes (i.e. ellipse or rectangle). This is because our approach uses the ground truth to form the clusters. We will perform the comparison on shapes, connectors and overall, regardless of their shape class. For the experiments on this dataset, we manually excluded text strokes, since text division is out of the scope of this research. The details of the dataset after relabelling and exclusion of text strokes can be seen in Table 7.5.

One of the main challenges of this dataset is the connectors. As can be seen in Figure 7.3, the diagram is highly connected, and the arrow heads are spatially very close to each other. Figure 7.4 shows the plot of the regular shapes in the dataset in 2D space after applying MDS.

	#Shapes	#Strokes
Rectangle	185	329
Ellipse	154	156
Arrow	452	742
Overall	791	1227

Table 7.5: The details of family tree dataset

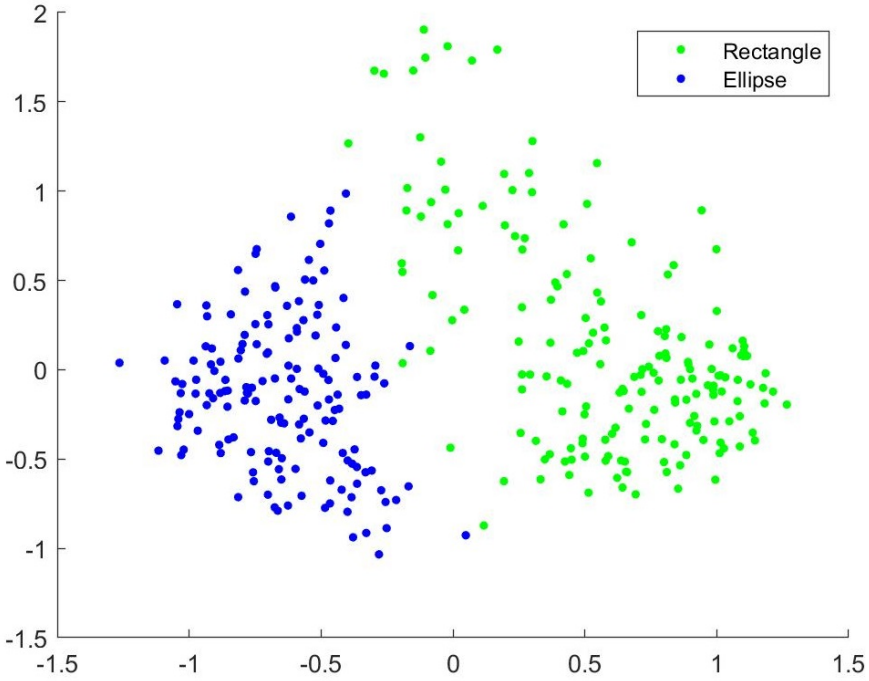


Figure 7.4: The plot of the family tree diagram's regular shapes after mapping the data into 2D space using MDS.

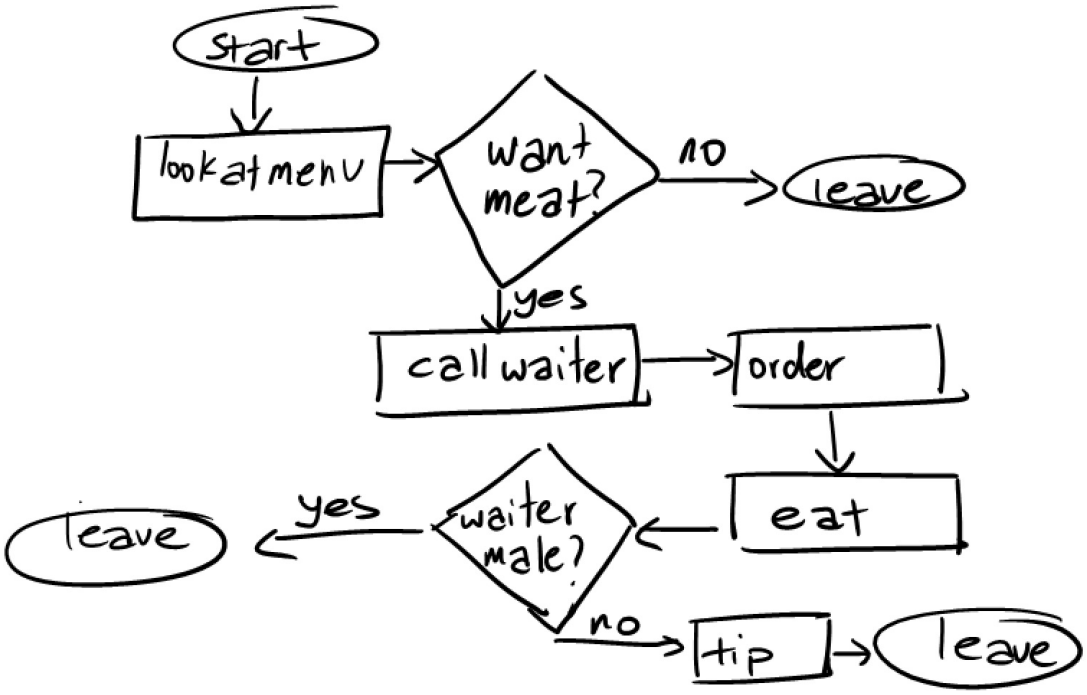


Figure 7.5: An example sketch from the process diagram dataset

	#Shapes	#Strokes
Rectangle	158	352
Ellipse	67	76
Diamond	65	153
Arrow	294	594
Overall	584	1175

Table 7.6: The details of process diagram dataset

Process Diagrams

The process diagram dataset (Schmieder 2009) comprises 33 sketches collected by 33 participants on a Tablet PC. The dataset includes three regular shapes (rectangle, ellipse and diamond) connected by arrows. Similar to the family tree diagram, this dataset contains text strokes, which we have manually excluded for our evaluations. Figure 7.5 shows an example diagram from this dataset.

The ground truth for the dataset is provided to discriminate shapes vs text vs connectors, which is sufficient for evaluation of the work of Stahovich et al. (2014). For the evaluation of our approach, similar to the family tree diagrams, we re-labelled the dataset to include the label for each of the three regular shapes (i.e., rectangle, ellipse and diamond). The details of the dataset can be seen in Table 7.6, and the plot of the regular shapes in the dataset in 2D space after applying MDS can be seen in Figure 7.6.

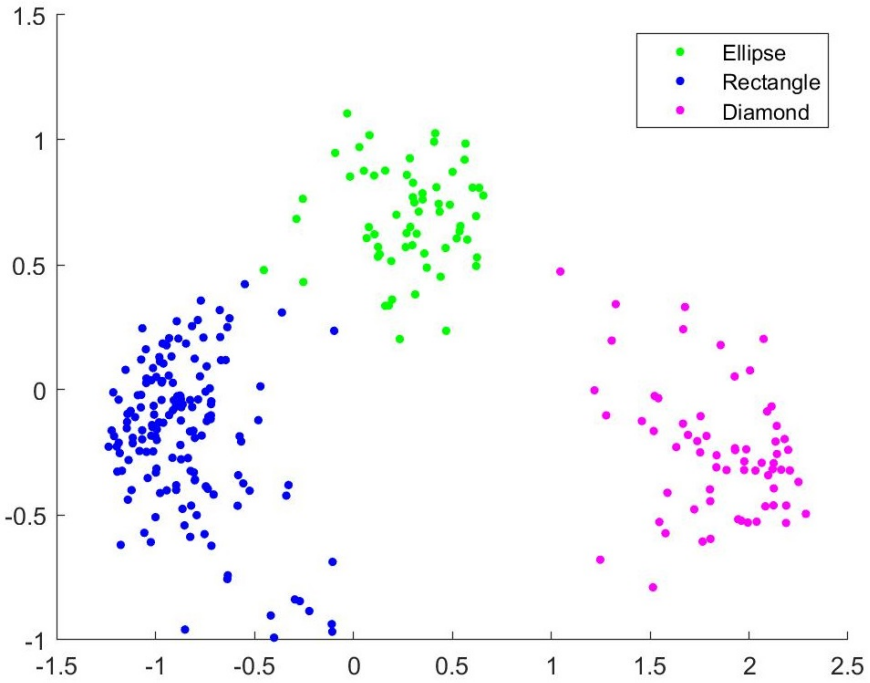


Figure 7.6: The plot of the process diagram's regular shapes after mapping the data into 2D space using MDS.

7.2.3 Results

We perform 5-fold user holdout cross validation, where four folds are used for training and one for testing. For the evaluations in this section, we disregard the shape classification results to make sure the shape classification accuracy does not affect the results. Therefore, we only check if a shape is correctly grouped and accepted by a rejector. Similarly, for the work of Stahovich, we only check if the shape is correctly grouped, regardless of what class the shape belongs to. This allow us to evaluate the systems in how accurately shapes are correctly being found. We use several metrics to evaluate our system and compare the results with Stahovich's, which are described in the following.

We first evaluate our system in terms of shape level accuracy, which measures the portion of shapes that are correctly grouped in the whole dataset. It is calculated by dividing the total number of correctly grouped (and accepted) shapes divided by the total number of shapes in the dataset (without any missing or extra strokes - see Equation 7.1). We have reported the results for regular shapes, connectors and overall (the shapes and the connectors together) for each dataset. We also report the overall accuracy across the two datasets for shapes, connectors and overall, which is calculated based on Equation 7.2. Table 7.7 shows the results of shape level accuracy.

$$\text{Shape Level Accuracy} = \frac{\#\text{Correctly grouped and accepted shapes in the dataset}}{\#\text{Shapes in the dataset}}$$

(7.1)

$$\text{Datasets Overall Accuracy} = \frac{\#\text{Correctly grouped and accepted shapes in all datasets}}{\#\text{Shapes in all datasets}} \tag{7.2}$$

The shape level accuracy is based on a strict method of perfect grouping without any errors. This means that if a grouped shape has even one missing or extra stroke, it is counted as an error. Similar to the evaluations in Section 5.3.4, we use three other metrics to measure the performance of our sketch recognition system, including *MissingStroke*, *ExtraStroke* and *1Error*. The *MissingStroke* counts a shape as correctly grouped if has one missing stroke, whereas the *ExtraStroke* allows one extra stroke. *1Error* shows the percentage of the correctly grouped and accepted shapes and the ones that have either a missing stroke or an extra stroke (not both at the same time). The evaluation results using these metrics for the shapes and connectors can be seen in Tables 7.11 and 7.12, respectively. In these tables the accuracy column represents the shape-level accuracies that are reported in Table 7.7.

In addition, we report the results for diagram level accuracy, which is a common method of evaluation (Stahovich et al. 2014, Arandjelović & Sezgin 2011, Sezgin & Davis 2008). This metric measures the accuracy of the system across diagrams. We measure the accuracy of each drawn diagram by dividing the number of correctly grouped shapes in a diagram by the total number of shapes in that diagram. We report the average and standard deviation of diagram level accuracies in Table 7.8 for regular shapes, connectors and overall (the shapes and the connectors together). See Equation 7.3 for calculation of the average of diagram level accuracy for n diagrams. We also report the results of paired t-tests in this table. In the following, we will provide more detailed discussion of the results for each dataset.

$$\text{Diagram Level Accuracy} = \frac{1}{n}\sum_{k=1}^{n}\text{Diagram Accuracy(k)}$$
$$\text{Diagram Accuracy(k)} = \frac{\#\text{Correctly grouped and accepted shapes in diagram k}}{\#\text{Shapes in diagram k}} \tag{7.3}$$

Family Tree Diagram

As can be seen in Table 7.7, our approach finds 86.7% of the shapes in the family tree dataset, which is higher than the 81.9% of Stahovich et al. (2014). Table 7.9 shows the accuracy for each shape class in the dataset. For the work of Stahovich, we check the label of the correctly grouped shape and report the results. As can be seen in Table 7.9, our approach finds 94.0% of ellipses (compared to the 91% in Stahovich's) and 80% of

		Ours	Stahovich
Family Tree	Shape	**86.7%**	81.9%
	Connector	42.9%	**55.8%**
	Overall	61.7%	**66.8%**
Process	Shape	**90.1%**	81.7%
	Connector	**77.9%**	77.6%
	Overall	**83.8%**	79.6%
Overall	Shape	**88.2%**	81.8%
	Connector	56.5%	**64.3%**
	Overall	71.0%	**72.2%**

Table 7.7: Shape level accuracy on the evaluation datasets.

		Ours	Stahovich	t-value	p-value
Family Tree	Shape	84.3% ±15%	88.2% ± 15%	0	0.33
	Connector	57.4% ±32%	64.2% ± 29%	0	0.4
	Overall	68.3% ±19%	76.2% ± 19%	0	0.11
Process	Shape	**89.0%** ±11%	82.7% ±16%	1	< 0.5
	Connector	77.4% ± 17%	77.2% ±17%	0	0.97
	Overall	83.4% ± 12%	80.0% ±12%	0	0.26

Table 7.8: Diagram level accuracy on the evaluation datasets

the rectangles (compared to 61% in Stahovich's) in the family tree diagram. Although our approach achieves 84.3% accuracy on shapes compared to the 88.2% for Stahovich, for diagram level accuracy (Table 7.8), a paired t-test shows that this is not a significant difference.

However, our approach only groups 42.9% of the connectors in the family tree dataset, which is lower than Stahovich's results (55.8%). The connector heads in this diagram are in very close spatial proximity to each other (as can be seen in Figure 7.3), and often the connector heads from different connectors are grouped together. Since the number of connectors in this dataset is higher than the regular shapes (see Table 7.2), the overall accuracy of Stahovich's work is higher than ours for the family tree dataset (66.8% of Stahovich's compared to 61.7% achieved using our approach for shape level). Table 7.8 shows that when the diagram accuracy is considered, there is no significant difference between our approach and Stahovich's for the connectors and the overall accuracies on the family tree diagrams.

Table 7.11 shows that when 1 error is allowed, 92.6% of the shapes are found in the family tree dataset, which is higher than 86.7% of the strict method of the perfect grouping without any error. For the connectors, allowing 1 error increases the accuracy to 86.1% from 42.5% of the strict method (see Table 7.12). As mentioned earlier, the connectors are spatially close in this dataset and often parts of a head of a connector

	Ours	Stahovich
Ellipse	**94.1**%	91.1%
Rectangle	**80.5**%	61.3%

Table 7.9: The accuracy of each shape class in family tree dataset

gets grouped with the strokes of another connector. Tables 7.11 and 7.12 also show that errors due to a missing stroke is higher than the ones caused by an extra stroke.

Overall, our approach is more accurate in finding the shapes than Stahovich's, however, the difference is not significant. For the connectors, Stahovich's work is more accurate than our approach, but not significantly better. It should be noted that this dataset is used in Stahovich's paper, whereas for our approach this dataset was new. We will also report the computation times later in this section.

Process Diagram

As can be seen in Table 7.7 our approach is more accurate than the work of Stahovich for shapes, connectors and the overall accuracy for the Process diagram dataset. Table 7.8 also shows that our approach achieves significantly better results for shapes when diagram level accuracy is considered. In addition, Table 7.10 shows that our approach achieves better results on all of the shape classes in the process diagram dataset. Similar to the family tree dataset, allowing a missing stroke increases the reported accuracy more than allowing an extra stroke (see Tables 7.11 and 7.12). Allowing one error in the grouping, increases the shape accuracy to 94.9% from 90.2% and for connectors it increases the accuracy to 92.9% from 78.0%.

Overall, our approach is significantly more accurate than Stahovich's in finding the shapes in the process diagram dataset. Our approach is also more accurate in finding the connectors, however the difference is not significant. This dataset was new for both our approach and Stahovich's.

	Ours	Stahovich
Ellipse	**96.8**%	93.6%
Rectangle	**87.8**%	74.6%
Diamond	**88.8**%	86.1%

Table 7.10: Process

		Accuracy	Missing Stroke	Extra Stroke	1 Error
Family Tree	Ours	**86.7%**	**91.7%**	**87.6%**	**92.6%**
	Stahovich	81.9%	89.4%	84.3%	91.8%
Process	Ours	**90.2%**	**93.8%**	**91.3%**	**94.9%**
	Stahovich	81.8%	86.2%	81.8%	86.2%
Overall	Ours	**88.2%**	**92.68%**	**89.3%**	**92.7%**
	Stahovich	81.8%	87.9%	83.1%	89.1%

Table 7.11: The accuracy results for evaluation datasets with different metrics for shapes.

		Accuracy	Missing Stroke	Extra Stroke	1 Error
Family Tree	Ours	42.5%	75.2%	53.3%	86.1%
	Stahovich	**55.8%**	**91.3%**	**60.9%**	**96.4%**
Process	Ours	**78.0%**	85.4%	**85.4%**	**92.9%**
	Stahovich	77.6%	**89.8%**	77.6%	89.8%
Overall	Ours	59.2%	80.0%	68.4%	89.3%
	Stahovich	**66.0%**	**90.5%**	**68.7%**	**93.2%**

Table 7.12: The accuracy results for evaluation datasets with different metrics for connectors.

Computation Time

We also evaluate the performance of our approach in terms of computation time. As mentioned in Section 3.4, all the experiments for this thesis are carried out on a Surface Pro 4 (Core i7-6650U, 16GB RAM). All the code is written in C# using Visual Studio 2017. We measure the computation time for each diagram from the time the input strokes are given to the system, to the time the system returns the results for all the shapes and connectors using the Stopwatch class in C#. We ran each experiment 10 times and reported the average of computation time in Table 7.13. The results in Table 7.13 shows that our approach is significantly faster than Stahovich's on the process diagram dataset. For the family tree dataset, our approach is faster, however, the difference is not significant.

	ours	Stahovich	t-value	p-value
Family	3.5 ±6	5.2 ±11	0	0.5
Process	1.5 ±0.8	7.3 ±6	1	<0.05

Table 7.13: The computation time (in seconds) of our approach compared to Stahovich's

7.2.4 Summary

In this section we first described the details of Stahovich et al. (2014) work, followed by a description of the evaluation datasets. As mentioned in Section 7.2, we chose the work of Stahovich et al. (2014) as a choice of comparison because of its characteristics of being domain independent, having a low computation time, and allowing free sketching environment. The downside of this approach was the accuracy. We compared our approach on the two evaluation datasets (family tree and process diagrams), and the results show that our approach finds 88.2% of the regular shapes across the two datasets, which is higher than the 81.8% for Stahovich's. The results also show that our approach is significantly more accurate at finding shapes in the process diagram datasets. Moreover, our approach is faster than the work of Stahovich et al. (2014). In the next section, we will compare our approach with the reported results on two public datasets.

7.3 Evaluation on Public Datasets

Two of the datasets we used during development are the FC and FA datasets (described in Chapter 3), which are publicly available. Various approaches in the literature have reported their results on these two datasets. We also evaluate our approach in terms of accuracy and computation time on these two datasets to be able to compare our results with that of the reported in the literature. Tables 7.14 and 7.15 show the details of FC and FA datasets, respectively.

As described in Chapter 3, the precursor to our sketch recognition system is a divider that separates the text strokes from shape strokes. The FC and FA datasets include text strokes and the majority of the reported results on the FC and FA datasets in the literature include the text results as well. In order to have a fair comparison with the reported results in the literature, we obtained the divider's executable code of Bresler et al. (2016*a*) work, one of the best reported results on these public datasets in the literature to separate the text strokes from the shape ones. The work of Bresler et al. (2016*a*) has been compared against other approaches on these datasets, which will be used to make a comparison to our work.

For the evaluation of our approach on these two datasets, the class of the accepted

	Training set		Test set	
	Count	**Strokes**	**Count**	**Strokes**
Terminator	299	458	203	446
Data	416	1382	294	924
Decision	310	1055	211	695
Process	598	1828	407	1130
Connection	177	190	125	136
Arrow	1829	3882	1260	2736
Text	1911	14560	1291	9629
Overall	5540	23355	3791	15696

Table 7.14: The details of the FC dataset

	Training set		Test set	
	Count	**Strokes**	**Count**	**Strokes**
State	720	733	284	287
Final State	342	688	129	260
Text	2833	5448	1114	2077
Arrow	2043	3980	796	1501
overall	5938	10849	2323	4125

Table 7.15: The details of the FA dataset

shapes are determined by a trained SVM classifier that uses the original space with the three added stroke level features (described in Section 5.1). The SVM uses a linear kernel function with $C = 3$.

We calculate the accuracy of our system at the shape level and stroke level. The shape level accuracy is measured for each shape by dividing the number of correctly grouped and labelled shapes (without any missing or extra stroke) by the total number of shapes of that class. The overall accuracy is measured by dividing the number of correctly grouped, accepted and classified shapes by the total number of shapes in the dataset. In Section 7.2.3 we disregarded the shape classification results to only consider the grouping and acceptance of shapes. However, in this section, we consider the shape class that the classifier assigns to an accepted shape, since other approaches have reported their results in the same way. In these datasets the number of text strokes are very large compared to the regular shapes or the connectors, and its results are not our contribution, we also report the overall accuracy by excluding the text results. In addition, we report the overall shape-only accuracy as well.

During drawing of the sketches, sometimes users try to correct or beautify a drawn shape, which leads to some redundant strokes. These strokes may not get grouped with the rest of the shape's strokes. On the other hand, our system sometimes accepts an incomplete shape, where the incomplete shape is classified correctly. Hence, we

measure the accuracy of the system at the stroke level as well, which is a common way of evaluation in the literature. The stroke level accuracy is measured by dividing the number of strokes that are correctly classified by the total number of strokes in that class (see Equation 7.4 for calculating the stroke level accuracy for a shape class). Similar to the shape level, we report the results on the shapes only as well.

$$\text{Stroke Level Accuracy} = \frac{\#\text{Correctly classified strokes in a shape class}}{\#\text{strokes of a shape class}} \tag{7.4}$$

7.3.1 Evaluation on FC dataset

The FC dataset (flowcharts) is divided into a training set (248 sketches) and test set (171 sketches). In total there are more than 9000 shapes from 6 classes (text, arrow, data, connection, terminator, process, decision). The ground truth for the dataset is provided as well (see Section 3.4 for the full details of the dataset). For our experiments on this dataset, the settings for the grouper and the rejection are the same as mentioned in Table 7.1. Since there is no hierarchy of shapes in this domain (no shape is extendable to another shape), we set the list of extendable shapes (E) to empty.

Table 7.16 shows the results of our approach on the FC dataset in the shape level compared to other reported results (we have only reported the results from the literature that achieved the overall accuracy of 80% or higher).

As can be seen in Table 7.16, overall our system achieves the best results by correctly grouping and recognising 86.8% of the shapes/connectors in the dataset. This is better than the best reported results of Bresler et al. (2016*a*) and Wang et al. (2016) work, achieving 84.2% and 84.3%, respectively. We also achieve better results on terminator, data, process and connection classes. The last row in Table 7.16 shows that our approach correctly groups and recognises 91.7% of the shapes in the FC dataset, which is higher than all other approaches. Since the division of text is not our contribution, we have reported the accuracy by excluding the accuracy of text division (the second to last row in Table 7.16), which includes shapes and arrows. The results show that our approach successfully groups and recognises 85.3% of the shapes and arrows in the FC dataset, which is higher than other approaches.

Table 7.17 shows the accuracy of our approach in the stroke level compared to other approaches. As can be seen in Table 7.17, our approach achieves overall 97.53% accuracy in the stroke level, outperforming all other approaches. We also achieve better results in the stroke level on all classes. The results of Delaye (2014) work in Table 7.17 is available only for the overall accuracy and not per class, therefore, we could not report the other results for this approach.

In addition, Julca-Aguilar et al. (2017) and Bresler et al. (2013*a*) have reported their results on the FC dataset for the case that text strokes are manually excluded. We carried out the same experiments and reported the results in Table 7.18. As can

	Bresler et al. (2014)	Wang et al. (2016)	Wu et al. (2015)	Bresler et al. (2016*a*)	Ours
Terminator	88.1%	80.8%	90.6%	89.0%	**93.1%**
Data	88.8%	84.4%	78.5%	90.5%	**93.5%**
Decision	74.1%	76.9%	**78.9%**	72.9%	75.4%
Process	87.2%	89.2%	88.3%	88.6%	**96.6%**
Connection	93.6%	79.8%	73.4%	95.1%	**96.8%**
Arrow	74.4%	**83.4%**	80.3%	76.6%	79.0%
Text	87.9%	85.8%	86.0%	**89.7%**	**89.7%**
Overall	82.8%	84.3%	83.2%	84.2%	**86.8%**
Overall without text	80.2%	83.5%	81.7%	81.8%	**85.3%**
Overall shapes	86.1%	83.6%	83.2%	87.1%	**91.7%**

Table 7.16: The shape level accuracy of our sketch recognition system on FC dataset.

	Bresler et al. (2014)	Wang et al. (2016)	Wu et al. (2015)	Bresler et al. (2016*a*)	Ours
Terminator	85.5%	-	91.6%	90.7%	**95.29%**
Data	95.6%	-	87.6%	95.3%	**98.05%**
Decision	90.8%	-	89.7%	88.2%	**97.70%**
Process	93.7%	-	91.8%	96.3%	**97.88%**
Connection	93.3%	-	73.3%	94.1%	**97.79%**
Arrow	85.3%	-	87.6%	87.5%	**93.09%**
Text	99.0%	-	98.8%	**99.2%**	**99.2%**
Overall	95.2%	95.8%	94.9%	96.3%	**98.30%**
Overall without text	89.2%	-	88.6%	90.7%	**95.53%**
Overall without text and arrow	92.5%	-	89.4%	93.4%	**97.53%**

Table 7.17: The stroke level accuracy of our sketch recognition system on FC dataset.

be seen in Table 7.18, our approach can correctly group and recognise 88.1% of the shapes/connectors in the FC dataset, outperforming the other two reported results for the case that text strokes are manually excluded.

	Stroke Level			Shape Level		
	1	2	Ours	1	2	Ours
Terminator	-	-	95.5%	-	-	93.6%
Data	-	-	98.0%	-	-	94.6%
Decision	-	-	97.8%	-	-	76.3%
Process	-	-	97.9%	-	-	97.1%
Connection	-	-	97.8%	-	-	96.8%
Arrow	-	-	93.3%	-	-	82.1%
Overall	91.1%	-	95.7%	85.5%	74.3%	88.1%

Table 7.18: The accuracy of our system on FC dataset for the case that text is manually excluded. 1 = (Julca-Aguilar et al. 2017), 2 = (Bresler et al. 2013*a*)

We also evaluated our system's performance in terms of computation time and reported the results in Table 7.19 (we have reported the available results from the papers in the literature). All the papers that reported their accuracies on the FC dataset have not reported their computation time. We measure the computation time for each sketch from the time the input strokes of the sketch are given to the system, to the time the

system returns the results for grouping and recognition. As can be seen in Table 7.19, our approach is slower than Wu et al. (2015) and Bresler et al. (2016*a*) work, similar to Bresler et al. (2014) work and faster than Carton et al. (2013) work. The average running time for our approach is 1.55 seconds, which is not long enough for the user to wait to see the results.

	Minimum	**Maximum**	**Average**	**Median**
(Bresler et al. 2014)	0.23	8.83	1.39	1.02
(Wu et al. 2015)	-	-	0.53	-
(Carton et al. 2013)	-	-	1.94	-
(Bresler et al. 2016*a*)	0.19	4.61	0.78	0.71
Ours	0.27	7.28	1.55	1.25

Table 7.19: The computation time (in second) of the our approach as compared to other approaches on FC dataset.

Overall, our approach achieves the best results for the shapes on the FC dataset, with correctly finding and recognising 91.7% of the shapes. Similarly, on a stroke level, our approach achieves the best results on each class and overall. Each diagram takes 1.55 seconds in average to be recognised.

7.3.2 Evaluation on FA dataset

Another publicly available dataset that was introduced in Section 3.4 is the finite automata (FA) dataset. This dataset contains 300 diagrams composed of two classes for regular shapes (state and final state), arrow and text. In this domain, a *state* (with a circle shape, e.g. state A in Figure 7.7) can be extended to a *final state* (a circle within another circle, e.g. state *F2* in Figure 7.7), hence, the list of extendable shapes (E) includes the *state*. The settings for the grouper and rejector are the same as given in Table 7.1. Similar to our experiments for the FC dataset, we use the same divider used by Bresler et al. (2016*a*), and compare our results against their approach. The work of Bresler et al. (2016*a*) has been compared to other approaches, we will compare against those approaches as well.

Bresler et al. (2014) mention that the arrows entering the initial state in a finite automata diagram look similar, hence, they have labelled such arrows as "arrow in". The labelling of these arrows did not make any difference to our results, hence, we treat these arrows same as other arrows. There is another type of arrow in this dataset that starts and ends with the same state, making a circular shape. For example, in Figure 7.7, states C and D have a circular arrow with label 1, and state B has a circular arrow that has a "0 + 1" label. During our experiments, we realised the arrows that start and

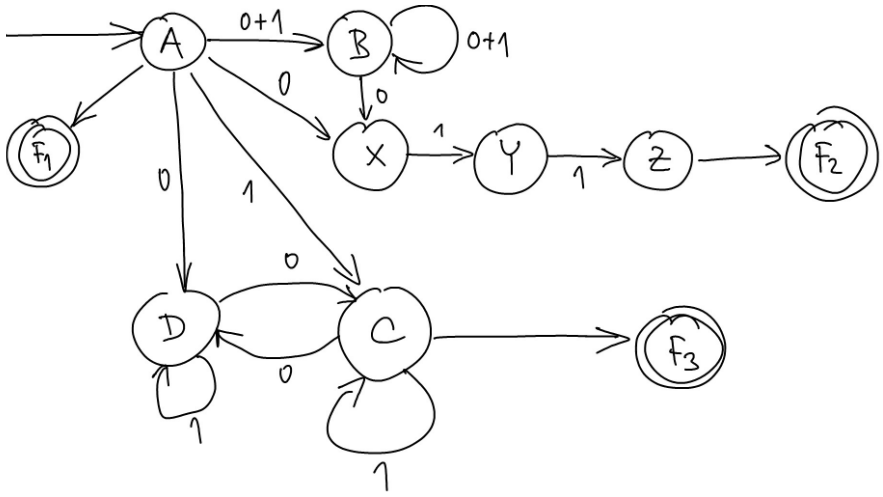


Figure 7.7: An example of a drawn sketch in the FA dataset (Bresler et al. 2014)

end with the same state have a circular shape and often get confused with a state class. We labelled such instances as "circular arrow". During the shape grouping process, if an input is classified as a *circular arrow* we treat it as a negative input and reject it. Later, in the connector grouping process, all the connectors are treated the same and no distinction is made between the circular ones and the rest.

Table 7.20 shows the results we achieved on the FA dataset compared to other approaches in the shape level. Since others have not reported their results for the case that text strokes are manually excluded, we have included the results of this experiment in the same table (last column). We have also reported the accuracy of our approach in the stroke level, in Table 7.21. We have also reported our results for the case that text strokes are manually excluded.

As can be seen in Table 7.20, if the text strokes are excluded, our approach can find and recognise 98.2% of *state* shapes and 97.7%% of the *final state* shapes. Using the divider used by Bresler et al. (2016*a*) has some errors, which affect the results. As can be seen in Tables 7.20 and 7.21, including the text strokes is affecting the overall accuracy of our system at both the shape level and the stroke level. This is because the text strokes are in the neighbourhood of the shapes, and therefore are often grouped with shapes, rather than rejected.

The results show that the work of Bresler et al. (2016*a*), outperforms ours and other approaches. It is worth mentioning that this dataset is collected by the authors of this paper. In addition, the work of Bresler et al. (2016*a*) is designed for arrow connected diagrams and performs optimisations based on some domain relations. This makes this approach domain dependent. Although we do not use such specific domain information, our approach can still find and recognise 90.6% of the shapes, connectors and texts in this domain.

	1	**2**	**3**	**4**	**5**	**Ours**	**Ours no text**
State	94.5%	-	-	91.2%	**98.2%**	93.7%	**98.2%**
Final State	93.8%	-	-	89.1%	**99.2%**	96.1%	97.7%
Text	96.0%	-	-	98.1%	**99.2%**	**99.2%**	-
Arrow	84.4%	-	-	95.3%	**97.5%**	76.3%	78.8%
Arrow Initial	80.0%	-	-	-	97.3%	-	-
Overall	91.5%	97.1%	97.2%	95.8%	**98.5%**	90.4%	84.6%
Overall without text and arrow	87.7%	-	-	93.6%	**97.7%**	82.5%	85.3%
Overall without text	94.2%	-	-	90.5%	**98.5%**	94.4%	98.0%

Table 7.20: The shape level accuracy of our sketch recognition system on FA dataset. 1 = (Bresler et al. 2014), 2 = (Delaye 2014), 3 = (Delaye & Lee 2015), 4 = (Wang et al. 2016), 5 = (Bresler et al. 2016*a*)

	1	**2**	**3**	**4**	**Ours**	**Ours without text**
State	95.2%	-	91.6%	**98.3%**	97.5%	99.6%
Final State	96.1%	-	96.5%	**99.2%**	98.0%	98.4%
Text	99.1%	-	99.0%	**99.7%**	**99.7%**	-
Arrow	89.3%	-	97.7%	**98.0%**	93.8%	94.0%
Arrow Initial	78.5%	-	-	97.3%	- %	-
Overall	94.5%	98.4%	98.0%	**99.0%**	96.5%	95.4%

Table 7.21: The stroke level accuracy of our sketch recognition system on FA dataset. 1 = (Bresler et al. 2014), 2 = (Delaye 2014), 3 = (Wang et al. 2016), 4 = (Bresler et al. 2016*a*)

	Minimum	Maximum	Average	Median
(Bresler et al. 2014)	0.25	15.86	2.37	1.73
(Bresler et al. 2016*a*)	0.27	1.43	0.69	0.62
Ours	1.15	13.18	3.73	3.01

Table 7.22: The computation time (in second) of the our approach as compared to other approaches on FA dataset.

Table 7.22 also shows the computation time of our approach on the FA dataset, compared with that of other approaches. Our approach takes longer to find and recognise the shapes as compared to the reported results. This is because the number of connector strokes are often higher than the number of shape strokes in a diagram. This means that there is a larger search space for the shape grouping/recognition algorithm, as there are more unrecognised strokes (i.e. connectors) in the search space. In addition, the state shapes (which make up double that of final state shapes in the dataset) are extendable and do not get removed from the search space when they are recognised. These issues result in our algorithm having to generate and test more shape candidates.

Overall, our approach can find and recognise 94.4% of the shapes in the FA dataset, which is the second best results, coming after the work of Bresler et al. (2016*a*) with achieving 98.5% accuracy. Manually removing the text strokes increases the accuracy of our approach to 98.0%, which is comparable to the 98.5% achieved by Bresler et al. (2016*a*). Our approach takes 3.7 seconds on average to find and recognise the shapes in a diagram in FA dataset.

7.4 Summary

In this chapter, we first introduced the work of Stahovich et al. (2014), followed by description of the evaluation datasets. We performed a full comparative study and compared our results with that of reported by Stahovich et al. (2014) on the evaluation datasets. The results show that our approach is significantly more accurate than the work of Stahovich et al. (2014) in finding shapes in process diagrams, and is also significantly faster for this dataset. On the family tree dataset, our approach has a comparable performance to Stahovich's work in terms of accuracy and computation time. Next, we evaluated our approach on two public datasets that were used during the development, and compared the results with other reported results in the literature. The results show that our approach can successfully group and recognise 91.7% of the shapes in the FC dataset, which is higher than that of all other reported results in the literature. For the FA dataset, our approach correctly finds and recognises 94.4% of the shapes in the dataset. This is comparable to many results in the literature, and only outperformed by a method that uses domain information.

Chapter 8

Conclusion and Future Work

In this thesis we built a recognition system for online sketched diagrams. In particular, we focused on connected diagrams, which are more challenging due to the close spatial proximity of connectors and shapes. This chapter provides a summary of the thesis and review of the contributions made to area of sketch recognition. We also discuss possible future research.

8.1 Contributions

The key contributions of this work are:

- A deterministic grouping algorithm. The grouper hypothesises shape candidates, in a continuous interaction with a recogniser. We showed that the grouper in isolation can perform the grouping task efficiently.
- A rejection system that uses novelty detection techniques for identifying invalid shape candidates. We explored different rejection methods, and showed that the proposed rejection technique in isolation is accurate in rejecting invalid shape candidates.
- A connector recognition system. Our connector recognition learns about the connector heads and tries to localise them around the recognised shapes.

The objective of this research was to develop a system that has the following characteristics:

- Domain independence: The nature of our approach is domain independent. However, we showed for domains with extendable shapes (such as finite automata or digital circuits), the list of extendable shapes as an input to the system can significantly reduce the computation time.

- Being fast: In the evaluations we showed our approach performs significantly faster than Stahovich's system (Stahovich et al. 2014) on one of the datasets. On public datasets, our approach performs slightly slower than some of the results reported by Bresler et al. (2016*a*) and Wu et al. (2015), comparable to results obtained by Cowans & Szummer (2005) and faster than those obtained by Carton et al. (2013).

- Supportive of free sketch environments: Our approach does not put any restriction on how shapes should be drawn in a diagram, therefore, it supports a free sketching environment for the shapes. However, there are some limitations with the connector recognition system, such as bi-directed and undirected connectors or connectors connecting multiple shapes.

- Accuracy: In the evaluations we showed that our approach has comparable results with other approaches. We showed that our approach is significantly more accurate than that taken by Stahovich et al. (2014) in finding the shapes in process diagrams. Moreover, our approach is more accurate than all the reported results in the literature for the FC dataset. On the FA dataset our approach achieves the second best results being less accurate than those obtained by Bresler et al. (2016*a*), which use domain information.

8.2 Future Work

We thoroughly investigated simultaneous grouping and recognition system for sketch recognition. Further improvements to this field could be made by exploring the possibility of adding a backtracking process to the grouping. The described grouping algorithm is greedy, in that as soon as a shape is identified, its strokes are combined, and all those strokes are treated as a single element. This can lead to problems when a negative shape candidate is incorrectly accepted. We believe a backtracking process can be used after the recognition of shapes and connectors. The idea of backtracking is to borrow strokes from neighbouring shapes to build new shape candidates.

To further improve the accuracy of the system, some domain information (context) needs to be automatically learned. In the work of Alvarado & Davis (2004), *domain patterns* are defined to be combinations of domain shapes that are likely to occur, which are hard-coded to the system using grammar and language. For example, when an arrow is detected in a family tree domain, there should be a parent and child shapes before and after the arrow. We believe a domain independent system that is capable of automatically inferring domain patterns from training examples would be a further step forward in this area. The learnt domain patterns could be used to examine the validity of relationships between recognised shapes and connectors. This could work in

concert with a backtracking process, that could be launched if the recognised shapes and connectors do not conform to the known domain patterns. A possible way forward to learning such patterns is association analysis, which is used to find the relations and patterns in a dataset (Tan et al. 2018).

Another area of possible improvement to our system is the connector recognition, which is the main weakness of our approach. We showed that connector head rotation is an error prone process. We believe using different feature representation or rotation invariant features for the connector heads could improve the performance of the connector recogniser. In addition, the connector recognition system needs to be extended to have support for bi-directional connectors as well as the connectors that connect multiple shapes together.

As the dimensionality of the features increases, the data becomes increasingly sparse in the space that it occupies. Therefore, the training samples may not be representative of all possible samples. This is phenomenon is referred to as the curse of dimensionality (Tan et al. 2018), which becomes problematic for machine learning algorithms leading to lower accuracy. In the experiments of Chapter 5, we noticed that the curse of dimensionality did not affect the results when different methods of proximity measurements were used. In the future work, the reasons for this can be investigated.

We have shown that a deterministic system of simultaneous grouping and recognition for domain independent systems can be made, that it can run in short times, and that the results are sufficient for it to be used. Starting from this system a more accurate system could be made by incorporating context to conform the recognised shapes and connectors with the learned domain information.

Appendix A

Rejection Evaluation Results on Original Features

Table A.1: The evaluation result of proximity-based rejection method with different proximity metrics on the original features.

		FC	**FA**	**Flowchart**	**Class**	**Digital**
Cosine	TPR	99.6%	100.0%	96.1%	99.4%	98.0%
	TNR	86.4%	93.2%	90.2%	74.2%	87.6%
	AUC	93.0%	96.6%	93.2%	86.8%	92.8%
Correlation	TPR	99.5%	100.0%	95.7%	98.8%	92.3%
	TNR	89.1%	95.2%	92.7%	79.3%	91.6%
	AUC	94.3%	97.6%	94.2%	89.1%	92.0%
Euclidean	TPR	99.8%	100.0%	94.2%	99.4%	96.0%
	TNR	43.1%	95.1%	90.6%	67.1%	74.4%
	AUC	71.4%	97.5%	92.4%	83.3%	85.2%
Euclidean + Cosine	TPR	99.5%	100.0%	94.2%	98.8%	94.7%
	TNR	86.7%	95.2%	90.9%	74.7%	88.9%
	AUC	93.1%	97.6%	92.6%	86.8%	91.8%
Euclidean + Correlation	TPR	99.4%	100.0%	93.8%	98.8%	91.1%
	TNR	89.2%	95.7%	92.9%	79.4%	92.0%
	AUC	94.3%	97.8%	93.3%	89.1%	91.5%
Hamming	TPR	99.8%	100.0%	95.2%	98.8%	91.5%
	TNR	77.2%	81.7%	92.2%	87.1%	95.3%
	AUC	88.5%	90.8%	93.7%	93.0%	93.4%
Hamming + Cosine	TPR	99.5%	100.0%	93.8%	98.8%	91.1%
	TNR	87.4%	93.7%	93.3%	87.1%	95.9%

Table A.1 Continued

		FC	FA	Flowchart	Class	Digital
	AUC	93.5%	96.8%	93.5%	93.0%	93.5%
Hamming + Correlation	TPR	99.5%	100.0%	93.3%	98.2%	88.7%
	TNR	89.5%	95.4%	94.5%	87.7%	96.6%
	AUC	94.5%	97.7%	93.9%	93.0%	92.6%
Bray Curtis	TPR	99.6%	100.0%	95.2%	99.4%	93.9%
	TNR	81.3%	94.3%	92.3%	81.5%	92.2%
	AUC	90.4%	97.1%	93.8%	90.5%	93.1%
Bray Curtis + Cosine	TPR	99.5%	100.0%	95.2%	98.8%	93.5%
	TNR	86.6%	94.4%	92.3%	82.1%	92.2%
	AUC	93.0%	97.2%	93.8%	90.4%	92.9%
Bray Curtis + Correlation	TPR	99.5%	100.0%	94.7%	99.4%	91.9%
	TNR	89.2%	95.3%	93.2%	81.6%	92.9%
	AUC	94.3%	97.6%	94.0%	90.5%	92.4%
Dice	TPR	99.9%	100.0%	99.5%	100.0%	99.2%
	TNR	40.7%	30.7%	89.0%	85.9%	51.4%
	AUC	70.3%	65.3%	94.3%	92.9%	75.3%
Dice + Cosine	TPR	99.6%	100.0%	95.7%	99.4%	97.2%
	TNR	87.0%	93.4%	93.9%	86.3%	91.1%
	AUC	93.3%	96.7%	94.8%	92.9%	94.1%
Dice + Correlation	TPR	99.5%	100.0%	95.2%	98.8%	91.5%
	TNR	89.4%	95.3%	95.0%	87.6%	94.0%
	AUC	94.5%	97.6%	95.1%	93.2%	92.8%
Hellinger	TPR	99.5%	100.0%	95.2%	99.4%	94.7%
	TNR	89.1%	96.8%	93.9%	84.7%	92.5%
	AUC	94.3%	98.4%	94.5%	92.1%	93.6%
Hellinger + Cosine	TPR	99.5%	100.0%	95.2%	98.8%	94.3%
	TNR	89.5%	96.8%	93.9%	84.7%	92.5%
	AUC	94.5%	98.4%	94.5%	91.7%	93.4%
Hellinger + Correlation	TPR	99.5%	100.0%	95.2%	98.8%	91.5%
	TNR	90.2%	96.9%	94.2%	86.0%	93.2%
	AUC	94.8%	98.4%	94.7%	92.9%	92.4%
Kulczynski	TPR	99.9%	100.0%	99.5%	100.0%	99.2%
	TNR	36.8%	21.4%	88.3%	85.2%	39.2%
	AUC	68.3%	62.0%	93.9%	92.6%	69.2%
Kulczynski +	TPR	99.6%	100.0%	95.7%	99.4%	97.2%

Table A.1 Continued

		FC	**FA**	**Flowchart**	**Class**	**Digital**
Cosine	TNR	87.0%	93.4%	93.9%	85.9%	90.9%
	AUC	93.3%	96.7%	94.7%	92.7%	94.1%
Kulczynski + Correlation	TPR	99.5%	100.0%	95.2%	98.8%	91.5%
	TNR	89.4%	95.3%	95.0%	87.4%	94.0%
	AUC	94.5%	97.6%	95.1%	93.1%	92.7%
Manhattan	TPR	99.9%	100.0%	94.2%	98.8%	96.4%
	TNR	45.9%	95.0%	89.6%	68.0%	81.2%
	AUC	72.9%	97.5%	91.9%	83.4%	88.8%
Manhattan + Cosine	TPR	99.5%	100.0%	94.2%	98.8%	95.5%
	TNR	86.7%	95.2%	90.6%	75.3%	89.5%
	AUC	93.1%	97.6%	92.4%	87.0%	92.5%
Manhattan + Correlation	TPR	99.5%	100.0%	93.8%	98.8%	91.9%
	TNR	89.2%	95.8%	93.0%	79.5%	92.3%
	AUC	94.3%	97.9%	93.4%	89.2%	92.1%
Ellipse	TPR	99.3%	100.0%	92.8%	98.2%	91.9%
	TNR	72.2%	95.7%	91.5%	80.3%	86.6%
	AUC	85.7%	97.8%	92.2%	89.2%	89.2%
Ellipse + Cosine	TPR	99.1%	100.0%	92.8%	98.2%	91.1%
	TNR	88.7%	95.7%	91.6%	80.3%	90.9%
	AUC	93.9%	97.8%	92.2%	89.2%	91.0%
Ellipse + Correlation	TPR	99.0%	100.0%	92.3%	98.2%	88.7%
	TNR	90.2%	96.1%	93.1%	81.2%	92.9%
	AUC	94.6%	98.0%	92.7%	89.7%	90.8%

Appendix B

Rejection Evaluation Results on Merged Features

Table B.1: The evaluation result of proximity-based rejection method with different proximity metrics on the merged features.

		FC	**FA**	**Flowchart**	**Class**	**Digital**
Euclidean + Cosine	TPR	99.6%	100.0%	98.6%	99.4%	98.0%
	TNR	79.7%	88.7%	93.3%	88.5%	79.9%
	AUC	89.7%	94.3%	95.9%	93.9%	88.9%
Euclidean + Correlation	TPR	99.6%	100.0%	98.6%	99.4%	97.6%
	TNR	87.3%	88.9%	94.1%	90.7%	86.0%
	AUC	93.5%	94.4%	96.3%	95.0%	91.8%
Hamming + Cosine	TPR	99.7%	100.0%	99.5%	99.4%	99.2%
	TNR	79.1%	78.2%	91.4%	87.6%	78.6%
	AUC	89.4%	89.1%	95.4%	93.5%	88.9%
Hamming + Correlation	TPR	99.7%	100.0%	98.6%	98.8%	98.0%
	TNR	86.5%	87.5%	94.3%	92.1%	88.5%
	AUC	93.1%	93.7%	96.4%	95.4%	93.3%
Bray Curtis + Cosine	TPR	99.8%	100.0%	98.6%	99.4%	98.0%
	TNR	76.6%	85.0%	93.3%	90.4%	87.7%
	AUC	88.2%	92.5%	95.9%	94.9%	92.9%
Bray Curtis + Correlation	TPR	99.8%	100.0%	98.6%	99.4%	97.6%
	TNR	86.3%	87.1%	93.9%	91.1%	89.0%
	AUC	93.1%	93.5%	96.2%	95.3%	93.3%
Dice + Cosine	TPR	99.7%	99.8%	99.0%	100.0%	99.6%
	TNR	80.9%	79.2%	96.2%	88.7%	80.7%

Table B.1 Continued

		FC	**FA**	**Flowchart**	**Class**	**Digital**
	AUC	90.3%	89.5%	97.6%	94.4%	90.1%
Dice + Correlation	TPR	99.7%	99.8%	98.6%	99.4%	98.0%
	TNR	86.8%	88.3%	97.1%	91.3%	91.9%
	AUC	93.2%	94.0%	97.8%	95.3%	94.9%
Hellinger + Cosine	TPR	99.7%	100.0%	98.1%	99.4%	97.2%
	TNR	87.8%	89.8%	96.1%	90.2%	90.6%
	AUC	93.8%	94.9%	97.1%	94.8%	93.9%
Hellinger + Correlation	TPR	99.7%	100.0%	98.1%	98.8%	96.8%
	TNR	88.2%	90.0%	96.1%	91.0%	91.0%
	AUC	93.9%	95.0%	97.1%	94.9%	93.9%
Kulczynski + Cosine	TPR	99.7%	99.8%	99.5%	100.0%	99.6%
	TNR	80.5%	78.7%	95.9%	88.2%	79.0%
	AUC	90.1%	89.2%	97.7%	94.1%	89.3%
Kulczynski + Correlation	TPR	99.7%	99.8%	98.6%	99.4%	98.0%
	TNR	86.7%	88.3%	97.1%	91.2%	91.8%
	AUC	93.2%	94.0%	97.8%	95.3%	94.9%
Manhattan + Cosine	TPR	99.7%	100.0%	98.1%	99.4%	98.4%
	TNR	82.3%	87.3%	90.6%	88.0%	79.6%
	AUC	91.0%	93.6%	94.4%	93.7%	89.0%
Manhattan + Correlation	TPR	99.7%	100.0%	98.1%	99.4%	98.0%
	TNR	87.4%	88.2%	92.9%	90.8%	86.2%
	AUC	93.6%	94.1%	95.5%	95.1%	92.1%
Ellipse + Cosine	TPR	99.6%	100.0%	94.8%	98.8%	93.5%
	TNR	85.2%	93.4%	96.1%	91.1%	90.7%
	AUC	92.5%	96.7%	95.4%	95.0%	92.1%
Ellipse + Correlation	TPR	99.6%	100.0%	94.8%	98.8%	93.1%
	TNR	88.3%	93.4%	96.1%	91.3%	91.3%
	AUC	93.9%	96.7%	95.4%	95.1%	92.2%

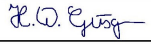
Appendix C

Statement of Contribution

MASSEY UNIVERSITY
GRADUATE RESEARCH SCHOOL

STATEMENT OF CONTRIBUTION
DOCTORATE WITH PUBLICATIONS/MANUSCRIPTS

We, the candidate and the candidate's Primary Supervisor, certify that all co-authors have consented to their work being included in the thesis and they have accepted the candidate's contribution as indicated below in the *Statement of Originality*.

Name of candidate:	Amir Ghodrati
Name/title of Primary Supervisor:	Prof. Hans Guesgen
Name of Research Output and full reference:	
Ghodrati, A., Blagojevic, R., Guesgen, H. W., Marsland, S., & Plimmer, B. (2018). The role of grouping in sketched diagram recognition. Expressive 18, pp. 2:1	
In which Chapter is the Manuscript /Published work:	Chapter 2
Please indicate:	
• The percentage of the manuscript/Published Work that was contributed by the candidate:	80%
and	
• Describe the contribution that the candidate has made to the Manuscript/Published Work:	
The review and summarisation of papers, and writing the paper draft.	
For manuscripts intended for publication please indicate target journal:	
Candidate's Signature:	
Date:	26/03/2020
Primary Supervisor's Signature:	
Date:	27/03/2020

(This form should appear at the end of each thesis chapter/section/appendix submitted as a manuscript/ publication or collected as an appendix at the end of the thesis)

References

Accord.NET (2019). [Online; accessed 1-July-2019].
URL: *http://accord-framework.net*

Alvarado, C. (2007), Sketch recognition for digital circuit design in the classroom, *in* '2007 Invited Workshop on Pen-Centric Computing Research', Citeseer.

Alvarado, C. & Davis, R. (2004), Sketchread: a multi-domain sketch recognition engine, *in* 'Proceedings of the 17th annual ACM symposium on User interface software and technology', ACM, pp. 23–32.

Alvarado, C. & Davis, R. (2006), Dynamically constructed bayes nets for multi-domain sketch understanding, *in* 'ACM SIGGRAPH 2006 Courses', ACM, p. 32.

Anthony, L. & Wobbrock, J. O. (2010), A lightweight multistroke recognizer for user interface prototypes, *in* 'Proceedings - Graphics Interface', Canadian Information Processing Society, pp. 245–252.

Anthony, L. & Wobbrock, J. O. (2012), $n-protractor: A fast and accurate multistroke recognizer, *in* 'Proceedings of Graphics Interface 2012', GI '12, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, pp. 117–120.
URL: *http://dl.acm.org/citation.cfm?id=2305276.2305296*

Arandjelović, R. & Sezgin, T. M. (2011), 'Sketch recognition by fusion of temporal and image-based features', *Pattern Recognition* **44**(6), 1225–1234.

Awal, A.-M., Feng, G., Mouchere, H. & Viard-Gaudin, C. (2011), 'First experiments on a new online handwritten flowchart database'.
URL: *https://doi.org/10.1117/12.876624*

Bhat, A. & Hammond, T. (2009), Using entropy to distinguish shape versus text in hand-drawn diagrams, *in* 'IJCAI International Joint Conference on Artificial Intelligence', Vol. 9, pp. 1395–1400.

Bishop, C. M., Svensen, M. & Hinton, G. E. (2004), Distinguishing text from graphics in on-line handwritten ink, *in* 'Proceedings - International Workshop on Frontiers in Handwriting Recognition, IWFHR', Vol. 4, pp. 142–147.

Blagojevic, R., Chang, S. H.-H. & Plimmer, B. (2010), The power of automatic feature selection: Rubine on steroids, *in* 'Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium', SBIM '10, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, pp. 79–86.
URL: *http://dl.acm.org/citation.cfm?id=1923363.1923377*

Blagojevic, R., Plimmer, B., Grundy, J. & Wang, Y. (2011), 'Using data mining for digital ink recognition: Dividing text and shapes in sketched diagrams', *Computers & Graphics* **35**(5), 976–991.

Bresler, M., Prusa, D. & Hlavác, V. (2013*a*), Modeling flowchart structure recognition as a max-sum problem, *in* '12th International Conference on Document Analysis and Recognition', IEEE, pp. 1215–1219.

Bresler, M., Prusa, D. & Hlavác, V. (2013*b*), Simultaneous segmentation and recognition of graphical symbols using a composite descriptor, *in* 'Computer Vision Winter Workshop', Vol. 13, pp. 16–23.

Bresler, M., Prusa, D. & Hlavác, V. (2015*a*), Detection of arrows in on-line sketched diagrams using relative stroke positioning, *in* 'IEEE Winter Conference on Applications of Computer Vision', IEEE, pp. 610–617.

Bresler, M., Prusa, D. & Hlavác, V. (2015*b*), Using agglomerative clustering of strokes to perform symbols over-segmentation within a diagram recognition system, CVWW '15, pp. 67–74.

Bresler, M., Prusa, D. & Hlavác, V. (2016*a*), 'Online recognition of sketched arrow-connected diagrams', *Int. J. Doc. Anal. Recognit.* **19**(3), 253–267.
URL: *http://dx.doi.org/10.1007/s10032-016-0269-z*

Bresler, M., Prusa, D. & Hlavác, V. (2016*b*), 'Online recognition of sketched arrow-connected diagrams', *International Journal on Document Analysis and Recognition (IJDAR)* **19**(3), 253–267.

Bresler, M., Prusa, D. & Hlavác, V. (2016*c*), Recognizing off-line flowcharts by reconstructing strokes and using on-line recognition techniques, *in* '2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)', pp. 48–53.

Bresler, M., Van Phan, T., Prusa, D., Nakagawa, M. & Hlavác, V. (2014), Recognition system for on-line sketched diagrams, *in* 'Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on', IEEE, pp. 563–568.

Brieler, F. & Minas, M. (2010), 'A model-based recognition engine for sketched diagrams', *Journal of Visual Languages & Computing* **21**(2), 81 – 97. Special Issue on Sketch Computation.

Buxton, B. (2007), *Sketching User Experiences: Getting the Design Right and the Right Design*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Carton, C., Lemaitre, A. & Coüasnon, B. (2013), Fusion of statistical and structural information for flowchart recognition, *in* '2013 12th International Conference on Document Analysis and Recognition', pp. 1210–1214.

Chang, C.-C. & Lin, C.-J. (2011), 'Libsvm: A library for support vector machines', *ACM Trans. Intell. Syst. Technol.* **2**(3), 27:1–27:27.
URL: *http://doi.acm.org/10.1145/1961189.1961199*

Chang, S. h.-h., Blagojevic, R. & Plimmer, B. (2012), 'Rata.gesture: A gesture recognizer developed using data mining', *Artif. Intell. Eng. Des. Anal. Manuf.* **26**(3), 351–366.
URL: *http://dx.doi.org/10.1017/S0890060412000194*

Chao, B., Zhao, X., Shi, D., Feng, G. & Luo, B. (2017), Eyes understand the sketch!: Gaze-aided stroke grouping of hand-drawn flowcharts, *in* 'Proceedings of the 22Nd International Conference on Intelligent User Interfaces', IUI '17, ACM, New York, NY, USA, pp. 79–83.

Cheema, S. (2014), 'Pen-based methods for recognition and animation of handwritten physics solutions'.

Chen, Q., Grundy, J. & Hosking, J. (2003), An e-whiteboard application to support early design-stage sketching of uml diagrams, *in* 'Proceedings of the 2003 IEEE Symposium on Human Centric Computing Languages and Environments', HCC '03, IEEE Computer Society, Washington, DC, USA, pp. 219–226.
URL: *http://dl.acm.org/citation.cfm?id=1153917.1153996*

Costagliola, G., De Rosa, M. & Fuccella, V. (2014), 'Recognition and autocompletion of partially drawn symbols by using polar histograms as spatial relation descriptors', *Computers & Graphics* **39**, 101–116.

Costagliola, G., Deufemia, V., Ferrucci, F. & Gravino, C. (2003), 'Exploiting xpg for visual languages definition, analysis and development', *Electronic Notes in Theoretical*

Computer Science **82**(3), 612 – 627.
URL: *http://www.sciencedirect.com/science/article/pii/S1571066105826313*

Costagliola, G., Deufemia, V. & Risi, M. (2005), Sketch grammars: A formalism for describing and recognizing diagrammatic sketch languages, *in* 'Eighth International Conference on Document Analysis and Recognition (ICDAR'05)', IEEE, pp. 1226–1230.

Costagliola, G., Rosa, M. D. & Fuccella, V. (2014), 'Local context-based recognition of sketched diagrams', *Journal of Visual Languages & Computing* **25**(6), 955 – 962. Distributed Multimedia Systems DMS2014 Part I.
URL: *http://www.sciencedirect.com/science/article/pii/S1045926X14001141*

Costagliola, G., Rosa, M. D. & Fuccella, V. (2015), 'Extending local context-based specifications of visual languages', *Journal of Visual Languages & Computing* **31**, 184 – 195. Special Issue on DMS2015.
URL: *http://www.sciencedirect.com/science/article/pii/S1045926X15000701*

Costagliola, G., v, V. & Risi, M. (2006), A multi-layer parsing strategy for on-line recognition of hand-drawn diagrams, *in* 'Proceedings of the Visual Languages and Human-Centric Computing', VLHCC '06, IEEE Computer Society, Washington, DC, USA, pp. 103–110.
URL: *https://doi.org/10.1109/VLHCC.2006.4*

Cowans, P. J. & Szummer, M. (2005), A graphical model for simultaneous partitioning and labeling, *in* 'Tenth International Workshop on Artificial Intelligence and Statistics', Citeseer.

Damm, C. H., Hansen, K. M. & Thomsen, M. (2000), Tool support for cooperative object-oriented design: Gesture based modelling on an electronic whiteboard, *in* 'Proceedings of the SIGCHI Conference on Human Factors in Computing Systems', CHI '00, ACM, New York, NY, USA, pp. 518–525.
URL: *http://doi.acm.org/10.1145/332040.332488*

Delaye, A. (2014), 'Structured prediction models for online sketch recognition', *International Journal on Interpretation* **1**(3), 4–16.

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

Delaye, A. & Liu, C.-L. (2012), *Text/non-text classification in online handwritten documents with conditional random fields*, Springer, pp. 514–521.

Deng, W., Wu, L., Yu, R. & Lai, J. (2013), On-line sketch recognition using direction feature, *in* 'IFIP Conference on Human-Computer Interaction', Springer, pp. 259–266.

Deufemia, V., Risi, M. & Tortora, G. (2014), 'Sketched symbol recognition using latent-dynamic conditional random fields and distance-based clustering', *Pattern Recogn.* **47**(3), 1159–1171.
URL: *http://dx.doi.org/10.1016/j.patcog.2013.09.016*

Ding, X., Li, Y., Belatreche, A. & Maguire, L. P. (2014), 'An experimental evaluation of novelty detection methods', *Neurocomputing* **135**, 313 – 327.
URL: *http://www.sciencedirect.com/science/article/pii/S0925231213011314*

Duda, R. O. & Hart, P. E. (1972), 'Use of the hough transformation to detect lines and curves in pictures', *Commun. ACM* **15**(1), 11–15.
URL: *http://doi.acm.org/10.1145/361237.361242*

EmguCV (2019). [Online; accessed 1-July-2019].
URL: *http://www.emgu.com/*

Everingham, M., Gool, L., Williams, C. K., Winn, J. & Zisserman, A. (2010), 'The pascal visual object classes (voc) challenge', *Int. J. Comput. Vision* **88**(2), 303–338.
URL: *http://dx.doi.org/10.1007/s11263-009-0275-4*

Fahmy, A., Abdelhamid, W. & Atiya, A. (2018), Interactive sketch recognition framework for geometric shapes, *in* L. Cheng, A. C. S. Leung & S. Ozawa, eds, 'Neural Information Processing', Springer International Publishing, Cham, pp. 323–334.

Feng, G., Viard-Gaudin, C. & Sun, Z. (2009), 'On-line hand-drawn electric circuit diagram recognition using 2d dynamic programming', *Pattern Recogn.* **42**(12), 3215–3223.
URL: *http://dx.doi.org/10.1016/j.patcog.2009.01.031*

Fonseca, M. J., Pimentel, C. & Jorge, J. A. (2002), Cali: An online scribble recognizer for calligraphic interfaces, *in* 'Sketch Understanding, Papers from the 2002 AAAI Spring Symposium', pp. 51–58.

Gennari, L., Kara, L. B., Stahovich, T. F. & Shimada, K. (2005), 'Combining geometry and domain knowledge to interpret hand-drawn diagrams', *Computers & Graphics* **29**(4), 547–562.

Géron, A. (2019), *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, O'Reilly Media.

Goldmeier, E. (1972), 'Similarity in visually perceived forms.', *Psychological issues* .

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. & Witten, I. H. (2009), 'The weka data mining software: an update', *ACM SIGKDD explorations newsletter* **11**(1), 10–18.

Hammond, T. A. & Davis, R. (2009), Recognizing interspersed sketches quickly, *in* 'Proceedings of Graphics Interface', Canadian Information Processing Society, pp. 157–166.

Hammond, T. & Davis, R. (2006), Ladder: A language to describe drawing, display, and editing in sketch recognition, *in* 'ACM SIGGRAPH 2006 Courses', ACM, p. 27.

Hammond, T. & Paulson, B. (2011), 'Recognizing sketched multistroke primitives', *ACM Transactions on Interactive Intelligent Systems (TiiS)* **1**(1), 1–34.

He, K., Zhang, X., Ren, S. & Sun, J. (2016), Deep residual learning for image recognition, *in* '2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)', pp. 770–778.

Herold, J. & Stahovich, T. F. (2012), The 1¢ recognizer: a fast, accurate, and easy-to-implement handwritten gesture recognition technique, *in* 'Proceedings - Sketch-Based Interfaces and Modeling, SBIM', Eurographics Association, pp. 39–46.

Hse, H. H. & Newton, A. R. (2005), 'Recognition and beautification of multi-stroke symbols in digital ink', *Computers & Graphics* **29**(4), 533–546.

Hse, H. & Newton, A. R. (2004), Sketched symbol recognition using zernike moments, *in* 'Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 1 - Volume 01', ICPR '04, IEEE Computer Society, Washington, DC, USA, pp. 367–370.
URL: *http://dx.doi.org/10.1109/ICPR.2004.838*

Hu, M.-K. (1962), 'Visual pattern recognition by moment invariants', *IRE transactions on information theory* **8**(2), 179–187.

Illingworth, J. & Kittler, J. (1988), 'A survey of the hough transform', *Computer Vision, Graphics, and Image Processing* **44**(1), 87 – 116.
URL: *http://www.sciencedirect.com/science/article/pii/S0734189X88800331*

Ioffe, S. & Szegedy, C. (2015), Batch normalization: Accelerating deep network training by reducing internal covariate shift, *in* 'Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37', ICML'15, JMLR.org, pp. 448–456.
URL: *http://dl.acm.org/citation.cfm?id=3045118.3045167*

Johnson, G., Gross, M. D., Hong, J. & Yi-Luen Do, E. (2009), 'Computational support for sketching in design: a review', *Foundations and Trends in Human-Computer Interaction* **2**(1), 1–93.

Johnston, D. & Alvarado, C. (2013), 'Sketch recognition of digital logical circuits'.

Julca-Aguilar, F. D. & Hirata, N. S. T. (2018), Symbol detection in online handwritten graphics using faster r-cnn, *in* '2018 13th IAPR International Workshop on Document Analysis Systems (DAS)', pp. 151–156.

Julca-Aguilar, F., Mouchère, H., Viard-Gaudin, C. & Hirata, N. S. T. (2017), 'A General Framework for the Recognition of Online Handwritten Graphics', *ArXiv e-prints* .

Kang, B., Hu, H. & LaViola Jr, J. J. (2014), Mixed heuristic search for sketch prediction on chemical structure drawing, *in* 'Proceedings of the 4th Joint Symposium on Computational Aesthetics, Non-Photorealistic Animation and Rendering, and Sketch-Based Interfaces and Modeling', ACM, pp. 27–34.

Kara, L. B. & Stahovich, T. F. (2005), 'An image-based, trainable symbol recognizer for hand-drawn sketches', *Computers & Graphics* **29**(4), 501–517.

Kara, L. B. & Stahovich, T. F. (2007), Hierarchical parsing and recognition of hand-sketched diagrams, *in* 'ACM SIGGRAPH - International Conference on Computer Graphics and Interactive Techniques', ACM, p. 17.

Landay, J. A. & Myers, B. A. (1995), Interactive sketching for the early stages of user interface design, *in* 'Proceedings of the SIGCHI Conference on Human Factors in Computing Systems', CHI '95, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, pp. 43–50.
URL: *http://dx.doi.org/10.1145/223904.223910*

Landay, J. A. & Myers, B. A. (1996), Sketching storyboards to illustrate interface behaviors, *in* 'Conference Companion on Human Factors in Computing Systems', CHI '96, ACM, New York, NY, USA, pp. 193–194.
URL: *http://doi.acm.org/10.1145/257089.257257*

Lee, C., Jordan, J., Stahovich, T. F. & Herold, J. (2012), Newtons pen ii: an intelligent, sketch-based tutoring system and its sketch processing techniques, *in* 'Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling', Eurographics Association, pp. 57–65.

Lee, W., Kara, L. B. & Stahovich, T. F. (2007), 'An efficient graph-based recognizer for hand-drawn symbols', *Computers & Graphics* **31**(4), 554–567.

Leung, W. H. & Chen, T. (2002), User-independent retrieval of free-form hand-drawn sketches, *in* 'Acoustics, Speech, and Signal Processing', Vol. 2, IEEE, pp. 2029–2032.

Li, Y. (2010), Protractor: a fast and accurate gesture recognizer, *in* 'Proceedings of the SIGCHI Conference on Human Factors in Computing Systems', ACM, pp. 2169–2172.

LibSVMsharp (2019). [Online; accessed 1-July-2019].
URL: *https://github.com/ccerhan/LibSVMsharp*

Lin, H.-L. (2014), Estimating Student Competence in Engineering Statics From a Lexical Analysis of Handwritten Equations, Thesis.

Liu, C.-L. & Zhou, X.-D. (2006), Online japanese character recognition using trajectory-based normalization and direction feature extraction, IWFHR,.

Liu, D. C. & Nocedal, J. (1989), 'On the limited memory bfgs method for large scale optimization', *Mathematical Programming* **45**(1), 503–528.
URL: *https://doi.org/10.1007/BF01589116*

Manual, F. (1997), '101-5-1, operational terms and graphics', *Washington, DC, Department of the Army* **30**(3), 2.

Marsland, S. (2003), 'Novelty detection in learning systems', *Neural computing surveys* **3**(2), 157–195.

Marsland, S. (2015), *Machine learning: an algorithmic perspective*, CRC press.

Mouchère, H., Viard-Gaudin, C., Zanibbi, R. & Garain, U. (2016), Icfhr2016 crohme: Competition on recognition of online handwritten mathematical expressions, *in* '2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)', pp. 607–612.

Niels, R., Willems, D. & Vuurpijl, L. (2008), 'The nicicon database of handwritten icons for crisis management'.

Oltmans, M. (2007), Envisioning sketch recognition: a local feature based approach to recognizing informal sketches, PhD thesis, Massachusetts Institute of Technology.

OpenCV (2019). [Online; accessed 1-July-2019].
URL: *https://opencv.org/*

Ouyang, T. & Davis, R. (2007), Recognition of hand drawn chemical diagrams, *in* 'Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 1', AAAI'07, AAAI Press, pp. 846–851.

Ouyang, T. & Davis, R. (2009*a*), Learning from neighboring strokes: Combining appearance and context for multi-domain sketch recognition, *in* 'Advances in Neural Information Processing Systems 22', pp. 1401–1409.

Ouyang, T. Y. & Davis, R. (2009*b*), A visual approach to sketched symbol recognition, *in* 'Proceedings of the 21st International Jont Conference on Artifical Intelligence', IJCAI'09, Morgan Kaufmann Publishers Inc., pp. 1463–1468.
URL: *http://dl.acm.org/citation.cfm?id=1661445.1661680*

Ouyang, T. Y. & Davis, R. (2011), Chemink: a natural real-time recognition system for chemical drawings, *in* 'In International Conference on Intelligent User Interfaces (IUI '11', ACM, pp. 267–276.

Patel, R., Plimmer, B., Grundy, J. & Ihaka, R. (2007), Ink features for diagram recognition, *in* 'Sketch-Based Interfaces and Modeling - ACM SIGGRAPH/Eurographics Symposium Proceedings', ACM, pp. 131–138.

Paulson, B. C. (2010), Rethinking pen input interaction: enabling freehand sketching through improved primitive recognition, PhD thesis, Texas A&M University.

Paulson, B. & Hammond, T. (2008), Paleosketch: Accurate primitive sketch recognition and beautification, *in* 'Proceedings of the 13th International Conference on Intelligent User Interfaces', IUI '08, ACM, pp. 1–10.
URL: *http://doi.acm.org/10.1145/1378773.1378775*

Paulson, B., Rajan, P., Davalos, P., Gutierrez-Osuna, R. & Hammond, T. (2008), What!?! no rubine features?: using geometric-based features to produce normalized confidence values for sketch recognition, *in* 'HCC Workshop: Sketch Tools for Diagramming', pp. 57–63.

Pearl, J. (1988), *Probabilistic reasoning in intelligent systems: networks of plausible inference*, Elsevier.

Pereira, J. a. P., Jorge, J. A., Branco, V. A., Silva, N. F., Cardoso, T. D. & Ferreira, F. N. (2004), Cascading recognizers for ambiguous calligraphic interaction, *in* 'Proceedings of the First Eurographics Conference on Sketch-Based Interfaces and Modeling', SBM'04, Eurographics Association, pp. 63–72.

Peterson, E. J., Stahovich, T. F., Doi, E. & Alvarado, C. (2010), Grouping strokes into shapes in hand-drawn diagrams, *in* 'Proceedings of the National Conference on Artificial Intelligence', pp. 974–979.

Pimentel, M. A., Clifton, D. A., Clifton, L. & Tarassenko, L. (2014), 'A review of novelty detection', *Signal Processing* **99**, 215 – 249.
URL: *//www.sciencedirect.com/science/article/pii/S016516841300515X*

Plimmer, B. & Apperley, M. (2003), Evaluating a sketch environment for novice programmers, *in* 'CHI '03 Extended Abstracts on Human Factors in Computing Systems', CHI EA '03, ACM, New York, NY, USA, pp. 1018–1019.
URL: *http://doi.acm.org/10.1145/765891.766126*

Plimmer, B., Blagojevic, R., Chang, S. H.-H., Schmieder, P. & Zhen, J. S. (2012), Rata: codeless generation of gesture recognizers, *in* 'Proceedings of the 26th Annual BCS Interaction Specialist Group Conference on People and Computers', British Computer Society, pp. 137–146.

Plimmer, B., Purchase, H. C. & Yang, H. Y. (2010), Sketchnode: Intelligent sketching support and formal diagramming, *in* 'Proceedings of the 22nd Conference of the Computer-Human Interaction Special Interest Group of Australia on Computer-Human Interaction', OZCHI '10, Association for Computing Machinery, New York, NY, USA, p. 136–143.
URL: *https://doi.org/10.1145/1952222.1952249*

Qi, Y., Szummer, M. & Minka, T. P. (2005), Diagram structure recognition by bayesian conditional random fields, *in* 'Computer Vision and Pattern Recognition, IEEE Computer Society Conference on', Vol. 2, IEEE, pp. 191–196.

Ray, S., Herrera-Cámara, J. I., Runyon, M. & Hammond, T. (2019), *Flow2Code: Transforming Hand-Drawn Flowcharts into Executable Code to Enhance Learning*, Springer International Publishing, Cham, pp. 79–103.
URL: *https://doi.org/10.1007/978-3-030-17398-2*$_6$

Reaver, J., Stahovich, T. F. & Herold, J. (2011*a*), How to make a quick $: Using hierarchical clustering to improve the efficiency of the dollar recognizer, *in* 'Proceedings - SBIM: ACM SIGGRAPH / Eurographics Symposium on Sketch-Based Interfaces and Modeling', ACM, pp. 103–108.

Reaver, J., Stahovich, T. F. & Herold, J. (2011*b*), How to make a quick$: Using hierarchical clustering to improve the efficiency of the dollar recognizer, *in* 'Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling', SBIM '11, ACM, New York, NY, USA, pp. 103–108.
URL: *http://doi.acm.org/10.1145/2021164.2021183*

Ren, S., He, K., Girshick, R. & Sun, J. (2017), 'Faster r-cnn: Towards real-time object detection with region proposal networks', *IEEE Trans. Pattern Anal. Mach. Intell.*

39(6), 1137–1149.
URL: *https://doi.org/10.1109/TPAMI.2016.2577031*

Rubine, D. (1991), 'Specifying gestures by example', pp. 329–337.
URL: *http://doi.acm.org/10.1145/122718.122753*

Saund, E. & Lank, E. (2003), Stylus input and editing without prior selection of mode, *in* 'UIST: Proceedings of the Annual ACM Symposium on User Interface Softaware and Technology', ACM, pp. 213–216.

Schmieder, P. (2009), Comparing basic shape classifiers: a platform for evaluating sketch recognition algorithms, PhD thesis, University of Auckland.

Schäfer, B. & Stuckenschmidt, H. (2019), Arrow r-cnn for flowchart recognition, *in* '2019 International Conference on Document Analysis and Recognition Workshops (ICDARW)', Vol. 1, pp. 7–13.

Sezgin, T. M. & Davis, R. (2005), Hmm-based efficient sketch recognition, *in* 'Proceedings of the 10th international conference on Intelligent user interfaces', ACM, pp. 281–283.

Sezgin, T. M. & Davis, R. (2007*a*), 'Sketch interpretation using multiscale models of temporal patterns', *IEEE Computer Graphics and Applications* **27**(1), 28–37.

Sezgin, T. M. & Davis, R. (2007*b*), Temporal sketch recognition in interspersed drawings, *in* 'Proceedings of the 4th Eurographics Workshop on Sketch-based Interfaces and Modeling', SBIM '07, ACM, New York, NY, USA, pp. 15–22.
URL: *http://doi.acm.org/10.1145/1384429.1384436*

Sezgin, T. M. & Davis, R. (2008), 'Sketch recognition in interspersed drawings using time-based graphical models', *Computers & Graphics* **32**(5), 500–510.

Sezgin, T. M., Stahovich, T. & Davis, R. (2007), Sketch based interfaces: Early processing for sketch understanding, *in* 'ACM SIGGRAPH 2007 Courses', SIGGRAPH '07, Association for Computing Machinery, New York, NY, USA, p. 37–es.
URL: *https://doi.org/10.1145/1281500.1281548*

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

Shilman, M., Viola, P. & Chellapilla, K. (2004), Recognition and grouping of handwritten text in diagrams and equations, *in* 'Frontiers in Handwriting Recognition, 2004. IWFHR-9 2004. Ninth International Workshop on', IEEE, pp. 569–574.

Stahovich, T. F., Peterson, E. J. & Lin, H. (2014), 'An efficient, classification-based approach for grouping pen strokes into objects', *Computers & Graphics* **42**, 14–30.

Stevens, P. C., Blagojevic, R. & Plimmer, B. (2013), Supervised machine learning for grouping sketch diagram strokes, *in* 'Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling', SBIM '13, ACM, New York, NY, USA, pp. 43–50.
URL: *http://doi.acm.org/10.1145/2487381.2487383*

Stoffel, A., Tapia, E. & Rojas, R. (2009), Recognition of on-line handwritten commutative diagrams, *in* '2009 10th International Conference on Document Analysis and Recognition', pp. 1211–1215.

Szegedy, C., Ioffe, S., Vanhoucke, V. & Alemi, A. A. (2017), Inception-v4, inception-resnet and the impact of residual connections on learning, *in* 'Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence', AAAI'17, AAAI Press, pp. 4278–4284.
URL: *http://dl.acm.org/citation.cfm?id=3298023.3298188*

Tan, P.-N., Steinbach, M., Karpatne, A. & Kumar, V. (2018), *Introduction to Data Mining (2Nd Edition)*, 2nd edn, Pearson.

Taranta, II, E. M. & LaViola, Jr., J. J. (2015), Penny pincher: A blazing fast, highly accurate $-family recognizer, *in* 'Proceedings of the 41st Graphics Interface Conference', GI '15, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, pp. 195–202.
URL: *http://dl.acm.org/citation.cfm?id=2788890.2788925*

Tarjan, R. (1972), 'Depth-first search and linear graph algorithms', *SIAM journal on computing* **1**(2), 146–160.

Tirkaz, C., Yanikoglu, B. & Sezgin, T. M. (2012), 'Sketched symbol recognition with auto-completion', *Pattern Recognition* **45**(11), 3926 – 3937.

Torgerson, W. S. (1952), 'Multidimensional scaling: I. theory and method', *Psychometrika* **17**(4), 401–419.
URL: *https://doi.org/10.1007/BF02288916*

Tumen, R. S., Acer, M. E. & Sezgin, T. M. (2010), Feature extraction and classifier combination for image-based sketch recognition, *in* 'Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium', SBIM '10, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, pp. 63–70.

Van Phan, T. & Nakagawa, M. (2016), 'Combination of global and local contexts for text/non-text classification in heterogeneous online handwritten documents', *Pattern Recognition* **51**, 112 – 124.
URL: *http://www.sciencedirect.com/science/article/pii/S0031320315002721*

Vatavu, R.-D., Anthony, L. & Wobbrock, J. O. (2012), Gestures as point clouds: A $p recognizer for user interface prototypes, *in* 'Proceedings of the 14th ACM International Conference on Multimodal Interaction', ICMI '12, ACM, pp. 273–280.

Vatavu, R.-D., Anthony, L. & Wobbrock, J. O. (2018), $q: A super-quick, articulation-invariant stroke-gesture recognizer for low-resource devices, *in* 'Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services', MobileHCI '18, ACM, New York, NY, USA, pp. 23:1–23:12.
URL: *http://doi.acm.org/10.1145/3229434.3229465*

Wang, C., Mouchère, H., Lemaitre, A. & Viard-Gaudin, C. (2017), 'Online flowchart understanding by combining max-margin markov random field with grammatical analysis', *International Journal on Document Analysis and Recognition (IJDAR)* **20**(2), 123–136.
URL: *https://doi.org/10.1007/s10032-017-0284-8*

Wang, C., Mouchère, H., Viard-Gaudin, C. & Jin, L. (2016), Combined segmentation and recognition of online handwritten diagrams with high order markov random field, *in* '2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)', pp. 252–257.

Wobbrock, J. O., Wilson, A. D. & Li, Y. (2007), Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes, *in* 'UIST: Proceedings of the Annual ACM Symposium on User Interface Softaware and Technology', ACM, pp. 159–168.

Wu, J., Wang, C., Zhang, L. & Rui, Y. (2015), Offline sketch parsing via shapeness estimation, *in* 'Proceedings of the 24th International Conference on Artificial Intelligence', IJCAI'15, AAAI Press, pp. 1200–1206.
URL: *http://dl.acm.org/citation.cfm?id=2832415.2832416*

Wu, T.-F., Lin, C.-J. & Weng, R. C. (2004), 'Probability estimates for multi-class classification by pairwise coupling', *J. Mach. Learn. Res.* **5**, 975–1005.

Yesilbek, K. T. & Sezgin, T. M. (2017), 'Sketch recognition with few examples', *Computers & Graphics* **69**, 80 – 91.
URL: *http://www.sciencedirect.com/science/article/pii/S0097849317301516*

Yin, J. & Sun, Z. (2005), An online multi-stroke sketch recognition method integrated with stroke segmentation, *in* 'International Conference on Affective Computing and Intelligent Interaction', Springer, pp. 803–810.