CENTER FOR MACHINE PERCEPTION

CZECH TECHNICAL UNIVERSITY IN PRAGUE

# Understanding Formal Arrow-Connected Diagrams and Free-form Sketches

Martin Bresler

Ph.D. Programme: Electrical Engineering and Information Technology

Branch of Study: Artificial Intelligence and Biocybernetics

breslmar@fel.cvut.cz

CTU–CMP–2016–04

May 12, 2016

Available at
ftp://cmp.felk.cvut.cz/pub/cmp/articles/bresler/Bresler-TR-2016-04.pdf

**Thesis Advisors: prof. Ing. Václav Hlaváč, CSc., RNDr. Daniel Průša, Ph.D.**

Supported by projects: TE01020197, SGS13/205/OHK3/3T/13, SGS16/085/OHK3/1T/13

**Research Reports of CMP, Czech Technical University in Prague, No. 4, 2016**

Published by

Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27 Prague 6, Czech Republic
fax +420 2 2435 7385, phone +420 2 2435 7637, www: http://cmp.felk.cvut.cz

PhD THESIS

ISSN 1213-2365

# Understanding Formal Arrow-Connected Diagrams and Free-form Sketches

Martin Bresler

May 12, 2016

## Copyright Note

I hereby declare that this dissertation describes my own original work developed under the supervision of RNDr. Daniel Průša, Ph.D. and prof. Ing. Václav Hlaváč, CSc. I acknowledged all sources used and cited these in the bibliography section. Additionally, we have collaborated with other researchers in order to tackle some of the tasks reported in this thesis.

The adaptive tool for segmentation of objects in free-form sketches presented in Section 5 was created during my internship at Media Interaction Lab (MIL), University of Upper Austria, where I collaborated with Florian Perteneder, Eva-Maria Grossauer, Joanne Leong, and Dr. Michael Haller. They are experts on user interfaces, while I focused on handwriting recognition and machine learning.

Finally, we have used public software and libraries in some of the developments of this thesis. We have explicitly acknowledged them in the corresponding sections of the thesis.

# Acknowledgement

I am thankful to my two supervisors RNDr. Daniel Průša, Ph.D. and prof. Ing. Václav Hlaváč, CSc. for fruitful discussions and brilliant ideas which led me in my research towards the fulfillment of the Ph.D. degree. I would also like to thank to my family, colleagues at work in the Center for Machine Perception (CMP), and to my friends for all their support that made it possible for me to finish this thesis.

I am grateful that I could participate in the following internships during my studies. They were often closely related to my research and I gained valuable experience:

- I was an exchange student in the Nakagawa Laboratory at the Tokyo University of Agriculture and Technology (TUAT), Tokyo, Japan from September to December 2013. I was supervised by Prof. Masaki Nakagawa. I thank him for his wise guidance. I was working on the diagram recognition framework and I was able to benefit from the deep knowledge of handwriting recognition in the lab. Specifically, I got insight into the task of mode detection in the sequence of handwritten strokes thanks to my colleague Phan Van Truyen. The internship was part of the Student Exchange Program (Short-Term Study in Japan) supported by the Japan Student Services Organisation (JASSO).

- I was a research intern in the Media Interaction Lab (MIL) at the University of Upper Austria, Hagenberg, Austria from September 2014 till January 2015. I was supervised by Prof. Michael Haller and I thank him for being a great and motivating mentor. I joined the team in the ongoing project Idea Garden (funded by the European Union Seventh Framework Programme FP7/2007-2011 under grant agreement n° 318552) . We worked on cooperative free-form sketching using large surface interactive whiteboards. I contributed mainly into segmentation of object of interest in the sketches.

- I participated in a summer internship at the Apple, Inc. company from July to September 2015. Although I am not allowed to give details due to a very strict Non-Disclosure Agreement (NDA), it was a source of valuable experience.

Finally, I gratefully acknowledge that my research was supported by the Technology Agency of the Czech Republic under Project TE01020197 Center Applied Cybernetics and the Grant Agency of the CTU under the projects SGS13/205/OHK3/3T/13 and SGS16/085/OHK3/1T/13.

# Abstract

Drawing has been a natural way for humans to express their thoughts and ideas since ancient times. People got used to create and understand illustrations. Many well-defined formal notations have evolved including technical drawings, musical scores, and various diagrams. On the other hand, people use free-form sketches or ad hoc intuitive notations often. Electronic devices equipped with a touch screen take part in our daily lives and make it is easier to create and share drawings. Naturally, it is more and more desired to have methods for automatic recognition and understanding of drawings. It is much easier and desired in the case of formal drawings consisting of well defined entities. A computer system can work further with such recognized drawing – beautify the drawing, rearrange a diagram, perform some actions according to a diagram, or manufacture something according to a technical drawing. On the other hand, it is extremely difficult and potentially unnecessary in the case of free-form sketches. It is often enough to provide the user with tools for easier creation and manipulation with sketches. The reason is that these sketches are meant to be understood by humans only and the computer system serves for their creation, storage, and sharing. Obviously, we can classify drawing into formal drawings and free-from sketching.

This thesis deals with two tasks: recognition of formally defined diagrams and segmentation of object of interest in free-form sketches. These two apparently different topics are strongly related. We assume that the drawing is created on an electronic device and thus it is in form of a sequence of strokes rather than a raster image.

We propose a recognition framework for arrow-connected diagrams. We introduce a model for recognition by selection of symbol candidates, based on evaluation of relations between candidates using a set of predicates. It is suitable for simpler structures, in which the relations are explicitly given by symbols, arrows in the case of diagrams. Knowledge of a specific diagram domain is used. The two domains are flowcharts and finite automata. We created a benchmark database of diagrams from these two domains. Although the individual pipeline steps are tailored for these, the system can be readily adapted for other domains. The recognition pipeline consists of the following major steps: text/non-text separation, symbol segmentation, symbol classification, and structural analysis. We performed a comparison with state-of-the-art methods for recognition of flowcharts and finite automata and verified that our approach outperforms them. We also analysed our system thoroughly and identified most frequent causes of recognition failures.

The situation is more complicated in the case of a free-from sketching where the user can draw and write anything freely. We cannot expect any particular structure. We can often find a combination of pictorial drawing with more structured figures and text in form of labels. The classical segmentation by recognition can not be used here. In some cases, the understanding of a sketch might be difficult even for humans. Specifically, the identification of an individual object in a drawing may differ from user to user, case to case. Recent research showed that a single linkage agglomerative clustering of strokes with trainable distance function can be used for segmentation of objects from predefined symbol classes in formal drawings. The proper distance function can be learned from annotated data. It requires a lot of data and time. We propose an approach combining several pre-trained distance functions for particular structures together to segment object in free-form sketches. We show that the desired segmentation can be achieved in many cases by combining together distance functions trained for very general object types like rows, columns, words, or compact images. We also show that the best combination of distance functions can be found from a very limited data in real time. We propose a segmentation approach, which estimates the optimal combination of clustering distance functions from an initial selection of one object. It results in segmentation of objects, which have similar characteristics to the initial one. Based on this approach, we designed a selection tool bringing additional functionality allowing to select and manipulate the segmented objects seamlessly. The method is suitable for fast rearrangement of sketches during collaborative content creation (brainstorming).

# Abstrakt

Kresba je pro člověka přirozeným nástrojem k vyjádření jeho myšlenek a nápadů již od dávných dob. Lidé si zvykli vytvářet a chápat ilustrace. Vyvinulo se mnoho dobře definovaných formálních notací zahrnujících technické výkresy, hudební zápisy, či různé diagramy. Na druhou stranu lidé stále často používají volnou kresbu či skici s intuitivními ad hoc notacemi. Díky tomu, že se elektronická zařízení s dotykovou obrazovkou stala běžnou součástí našich životů, je stále jednodušší vytvářet a sdílet kresby. Přirozeně jsou tak stále více žádány metody pro automatické rozpoznávání a porozumění kresbám. Zajisté je daleko snazší a žádanější dosáhnout toho u formálních kreseb skládajících se z jasně definovaných entit. Počítačový systém může s rozpoznanou kresbou dále pracovat. Může kresbu zkrášlit, přeuspořádat diagram, vykonat určité akce vyjádřené diagramem nebo zhotovit výrobek na základě technického výkresu. Na druhou stranu je to velmi obtížné a často zbytečné v případě volné kresby. V takovém případě je totiž často požadováno spíše poskytnout uživateli nástroje pro snadnější tvorbu a manipulaci s kresbou. Důvod je ten, že kresbě má porozumět pouze člověk a počítačový systém slouží pouze k jejímu vytvoření, uchování a sdílení. Očividně tak můžeme kresbu rozdělit na formální a volnou.

Tato práce se zabývá dvěma úlohami: rozpoznáváním formálně definovaných diagramů a segmentací objektů zájmu ve volně kreslených skicách. Tato dvě zdánlivě odlišná témata mají mnoho společného. V obou případech předpokládáme, že kresba byla vytvořena na elektronickém zařízení a skládá se z posloupností tahů. Nejedná se tedy o rastrový obrázek.

Navrhujeme rámec pro rozpoznávání šipkami pospojovaných diagramů. Představujeme model rozpoznávání založený na výběru vhodných kandidátů na symboly prostřednictvím vyhodnocení vztahů mezi kandidáty za použití predikátů. Je to vhodné pro jednodušší struktury, kde jsou vztahy explicitně dané samotnými symboly, šipkami v případě diagramů. Znalost specifické domény diagramů je využita. Dvěma vybranými doménami jsou vývojové diagramy a konečné automaty. Pro tyto dvě domény jsme vytvořili referenční databázi. Ačkoliv jsou jednotlivé kroky procesu rozpoznávání přizpůsobeny na míru zmíněným doménám, lze systém snadno přizpůsobit doménám dalším. Proces rozpoznávání se skládá z následujících významných kroků: oddělení textu od zbytku tahů, segmentace symbolů, klasifikace symbolů a strukturní analýzy. Provedli jsme srovnání s nejlepšími alternativními metodami pro rozpoznávání vývojových diagramů a konečných automatů a ověřili, že náš postup je předčí. Současně jsme důkladně analyzovali náš systém a identifikovali nejčastější příčiny selhání rozpoznávání.

Situace se komplikuje v případě volné kresby, kde může uživatel nakreslit prakticky cokoliv. Nelze očekávat žádnou konkrétní strukturu skici. Často narážíme na kombinaci obrázku s více strukturovanými schématy a textem v podobě různých popisků. Klasická segmentace prostřednictvím klasifikace zde nemůže být použita. V některých případech může být porozumění skici obtížné i pro člověka. Obzvláště názor na identifikaci jednotlivých objektů v obrázku se může lišit podle uživatele, konkrétního případu a kontextu. Nedávný výzkum ukázal, že lze úspěšně segmentovat objekty z předem definovaných tříd symbolů ve formálních kresbách sdružováním tahů hierarchickým shlukováním pomocí metody nejbližšího souseda. Použije se naučitelná vzdálenostní funkce. Vhodné parametry vzdálenostní funkce lze naučit z anotovaných dat. Vyžaduje to ovšem dostatek dat a času. My navrhujeme postup, který kombinuje několik předem naučených vzdálenostních funkcí pro určité struktury dohromady tak, aby bylo možno segmentovat objekty ve volné kresbě. Ukazujeme, že požadované segmentace lze často dosáhnout tím, že se zkombinují vzdálenostní funkce naučené pro velmi obecné typy objektů jako jsou řádky, sloupce, slova nebo kompaktní náčrtky. Ukazujeme také, že nejlepší kombinace těchto vzdálenostních funkcí lze nalézt v reálném čase pomocí velmi omezeného množství dat. Navrhujeme postup, který odhaduje optimální kombinaci vzdálenostních funkcí shlukovacího algoritmu z iniciálního výběru jednoho objektu uživatelem. Vede to na segmentaci objektů, které mají podobnou charakteristiku jako uživatelem vybraný objekt. Na základě tohoto postupu jsme vytvořili nástroj, který umožňuje velice snadno a rychle vybrat jednotlivé objekty a manipulovat s nimi. To je velmi užitečné například při reorganizaci skic během kolaborativní práce na interaktivní tabuli.

# Contents

# 1. Introduction

Handwriting and drawing is a natural way for humans to express and record their thoughts. Two-dimensional structure of drawing allows to cram more information into a limited space of a drawing canvas. Structure of the drawing is very important. The useful information is not only contained in the individual entities of the drawing, but in the relations between them as well. These relations are given by the relative position of individual entities. People commonly use various kinds of schemas, diagrams, technical drawings or sketches to illustrate things, which would be too difficult to describe in words. It is often said that a picture is worth a thousand words. Moreover, people figured out how to draw long time before the first alphabet and writing had been invented.

Research in handwritten document analysis has shifted from the recognition of plain text to recognition of more structured inputs such as mathematical and chemical formulas, music scores, or diagrams. The substantial attention has been paid to recognition of mathematical formulas. The syntax of mathematical notation is well formalized and the structure is often recursive and very rich. In this thesis, we deal with understanding of sketches with a simple non-recursive structure, specifically *arrow-connected diagrams*. While the recognition of mathematical expressions and other recursive complex structures is often based on parsing probabilistic grammars, we show that this approach is impractical for simpler structures where the relations are explicitly given by symbols, arrows in the case of diagrams. We propose a model for *recognition by selection* of *symbol candidates*, based on evaluation of relations between candidates using a set of predicates.

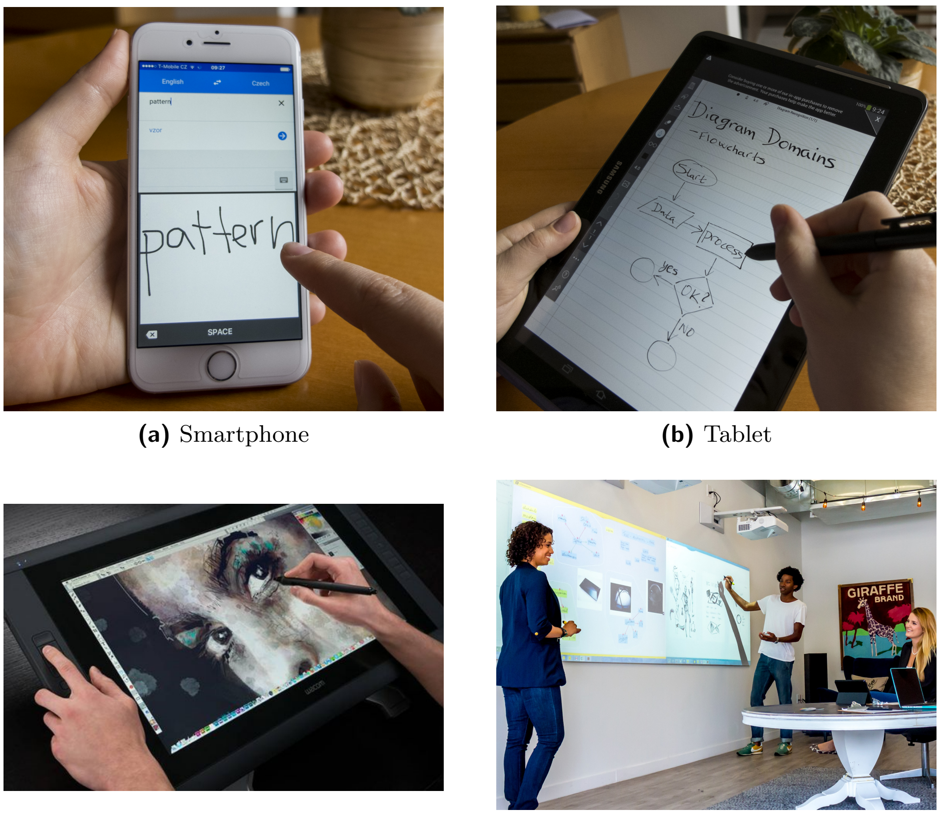
Second topic covered in this thesis deals with sketches of no explicit structure. We speak about *free-form sketches* where people use ad hoc notions to express their ideas. Combinations of pictures, schemas, tables, text labels or whole paragraphs are common. The full understanding of a free-form sketch is extremely difficult and thus we often seek another goal: to design *smart tools* making it easier and more convenient to create or edit such sketches. One of the most common tasks is a *rearrangement of objects*. These objects must be identified first. Usually, it is done by the user using a *selection tool*. The number of objects can be large and repeating selection can be tedious. We designed a tool for automatic segmentation of all objects at once. It tries to find a structure in the sketch from an initial selection of one single object.

This chapter presents the motivation for recognizing hand-drawn schemas and shows its applications in Section 1.1. State-of-the-art and related work is surveyed in Section 1.2. We formulate the problem and describe the goals of this thesis in Section 1.3. The specific contribution of this thesis is pointed out in Section 1.4. Finally, the structure of this thesis is outlined in Section 1.5.

## Chapter Outline

## 1.1. Motivation

Despite the fact that handwriting and drawing is a natural way for humans to express and record their thoughts, it is still not a typical way of *human-computer interaction*. We can find two exceptions here: handwritten plain text and professional graphic design. Recognition of handwritten plain, especially English, text is one of the biggest achievements. Today's recognizers are so accurate, that we may consider it a solved problem. A big advantage is that the input of handwritten plain text is one dimensional and can be done word by word and thus it can be mediated through a very small touch screen. Users write using only their fingers very often. In that case, we are talking about *touch input*. This was a significant motivation for development of successful recognizers and we can find them in every current touch input device. However, a bigger canvas is usually necessary to create structured two-dimensional drawing conveniently. Moreover, it is highly impractical to draw with a finger and thus a stylus is required. In that case, we are talking about *ink input*. The second exception is a graphic design, which is usually performed on professional tablets. These devices have been available for a long time. However, they are expensive and only professionals use them commonly. We can characterize it as art and these drawings usually do not require any recognition or formalization. They are not meant to be understood by computers. They are designed to be understood by another people. A computer system is used to mediate their creation, storage, and propagation.



**(a)** Smartphone

**(b)** Tablet

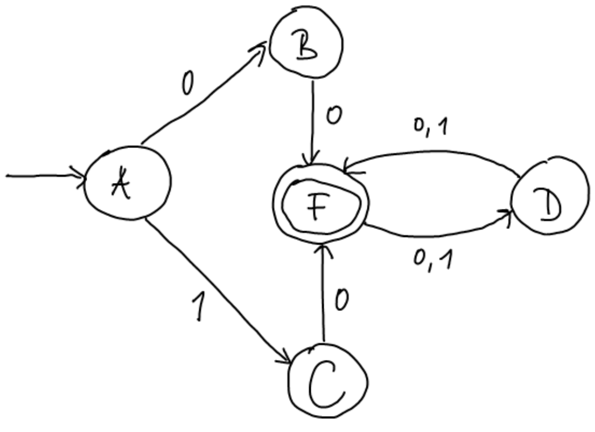
**(c)** Professional tablet

**(d)** Interactive whiteboard[1]

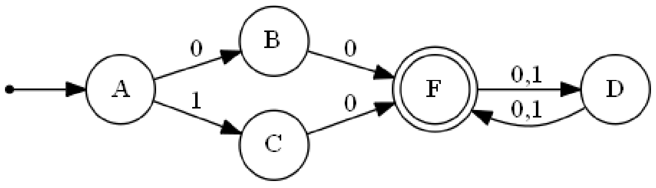
**Figure 1.1.** Illustrative images of devices designed to mediate human-computer interaction based on handwriting and drawing.

[1]Courtesy of we-inspire (`http://we-inspire.com/`)

Even with the recent advancement in the technology, it is still common that users use mouse and keyboard to create various drawings and schemas using old style drag and drop interfaces. It is more natural and faster to draw such drawings directly. However, it requires advanced algorithms able to recognize and formalize such input to provide the same result as achieved when using classical user interfaces. Their availability is still very limited. This situation is getting gradually better with the rise of devices allowing the ink input. Smart phones and tablets are getting larger screens with more precise and accurate styluses. These devices are becoming a part of our daily lives. It is the reason for the demand on algorithms, which make the human-computer interaction based on structured two-dimensional handwriting and drawing more common.



**(a)** A sketched finite automaton.



**(b)** Recognized result visualized by Graphviz.

**Figure 1.2.** An example of (a) a typical input and (b) the desired output of the proposed diagram recognizer.

Illustrative examples of the most common devices mediating human-computer interaction based on handwriting and drawing are shown in Figure 1.1. We can divide them into two main groups with respect to their basic usage: personal devices and interactive white boards (smartboards). Personal devices are smart phones, tablets, or tablet PCs. Usually only one user is drawing at one time and it is the owner naturally. It is thus advantageous to adapt the recognition algorithm to her/his writing style. On the other hand, an interactive whiteboard is usually placed in a seminar room and it is used simultaneously by multiple users. The task of the recognizer is thus much more difficult since multiple writing/drawing styles may appear in one document. Moreover, simultaneous editing must be solved to avoid conflicts. Smartphones are usually not proper devices for drawing more complex sketches. They have smaller screens usually not equipped by a stylus. Users often use touch input only and very rarely draw more complex drawings. The touch input is used for handwriting and making special gestures. The gestures do not represent handwritten words or drawn symbol directly. They are rather associated to some actions: paste a particular word/symbol, open/close a particular application. The advantage is that the gestures can be designed in such a way to be easily performed by the user and recognized by the system. On the other hand, the disadvantage is that the user must remember them, which limits their maximal number directly. Tablets and interactive whiteboards are two devices really appropriate for sketching.

When a drawing is created using an ink input device, we talk about *on-line* input and recognition. An on-line input is considered to be a sequence of handwritten strokes, in which a stroke is a sequence of points captured by an ink input device between pen-down and pen-up events. Every stroke point is defined by its coordinates on the planar drawing canvas. Additional data like a time stamp or a pressure value may be provided. A complementary *off-line* input is represented by a picture, which is obtained by scanning a paper or taking a photo of a whiteboard with a drawing. All dynamic information (order of strokes, speed, pressure) is missing and recognition is thus more difficult. Applications of an off-line recognition are typically different (digitization of old documents) and it is not dealt with in this thesis.The output is a structure, which describes the sketched diagram syntactically. Individual symbols are identified and relations between them are detected. Additionally, text is divided into logical blocks with known meaning. Several formats can be used to represent the recognition result, as exchange formats for diagrams have not been unified. We use the DOT graph description language, supported by the popular graph visualizer Graphviz[2]. Once we have such a representation, we can use it. A very common goal is a beautification of the drawn diagram. Beautified diagrams might be used in presentations, for example. Figure 1.2 shows an example of a drawn finite automaton and its beautified version created by Graphviz.

Our personal research motivation in this topic is based on the experience of our research group with recognition of mathematical expressions using a grammar-based structural analysis. It has many common aspects with the recognition of diagrams. However, the complex structure of mathematical expressions is much different from the simple structure of arrow-connected diagrams. We wanted to explore possibilities in designing a structural analysis, which would not be based on a 2D grammar. Our goal was to show that using a grammar on such a simple structure can be cumbersome. We hoped to find a simpler solution capable of easy adaptation for new domains without necessity of an expert to create a new grammar. Additionally, we have seen the success of recognizers for mathematical expressions and felt that there is a gap in research, because there was no successful recognizer for diagrams like flowcharts when our research began.

## 1.2. State of the Art

This section will survey work related mainly to the first topic of this thesis – recognition of online hand-drawn diagrams. Additional work related to the second topic of the thesis (segmentation of objects in free-from sketches) will be introduced later in the introduction of the corresponding Chapter 5.

Research in handwritten document analysis has shifted from the recognition of plain text to recognition of more structured inputs. Probably the most attention and effort have been put into recognition of mathematical expressions. The mathematical notation is a well-known language that has been used all over the world for hundreds of years. It has very rich and well defined recursive structure. Its recognition has many practical applications. Research in this field laid the foundations for further development in other domains of structured handwritten input. Therefore, we will pay an extra attention

[2]`http://graphviz.org/`

to this topic in the next section. After that we will survey research in recognition of various diagrams and then we will focus specifically on *flowcharts* and *finite automata*. Finally, we will explore alternative approaches to support design of diagrams, mostly flowcharts. Here we leave the pure recognition approaches and reveal efforts to make the recognition easier using certain restrictions on the way how users create diagrams.

Before we do so, we must mention that there has been also research in *off-line diagram recognition* [Refaat et al., 2008]. In that case, input is an image and thus any temporal information is missing. Off-line recognition faces different challenges (especially in segmentation) and typically leads to different applications. This thesis does not consider these topics. We will focus on on-line data and recognition only.

### 1.2.1. Mathematical Formulas

Recognition of mathematical formulas is a useful example of structural recognition as it has brought seminal methods and successful recognizers. Its research has begun a long time ago with the work of Andreson [1968]. Although it deals with printed mathematical expressions, it laid the foundations of syntactic analysis. First work focused on on-line handwritten expressions was done by Belaid and Haton [1984]. They combined top-down and bottom-up syntactic parsers to tackle the structural analysis. Since then, recognition of mathematical expressions has evolved into today's mature research field.

The Competition on Recognition of On-line Handwritten Mathematical Expressions (CROHME) has further boosted research and provides a reliable comparison ground. The commercial winner of CROHME 2014 [Mouchère et al., 2014], MyScript[3], achieves 62.7 % accuracy of correctly recognized formulas, while the non-commercial winner – Álvaro et al. [2014] – achieves 37.2 %. Recognizer designed by [Le et al., 2014] was another very successful participant of the competition.

Recognition of mathematical formulas and diagrams face some similar problems. Individual symbols have to be segmented, recognized, and embedded into the domain structure. However, the structure of mathematical formulas is strong and recursive, and grammars are thus suited to their expression. All of the participants of the CROHME contest used grammar driven parsing for the structural analysis. In contrast, diagram structure is simpler and less formalized and grammars do not seem to be the best model for their capture.

Recognizer of mathematical expressions finds its place in many applications. MyScript created an ink-based calculator for smartphones. It is more often used at schools for automatic evaluation of exams or to support the education. Another practical example of working recognizer is the Math Input Panel delivered by Microsoft since Windows 7. Taranta and LaViola [2015] are authors of Math Boxes, a pen-based user interface for simplifying the task of hand writing difficult mathematical expressions. It is an example showing the importance to provide a recognizer with a powerful user interface. The recognition result is immediately visualized – visible bounding boxes around certain subexpressions are automatically generated as the system detects specific relationships including superscripts, subscripts, and fractions. Upon accepting new characters, box boundaries are dynamically resized and neighbouring terms are translated to make a room for the larger box. A feedback on structural recognition is given via the boxes themselves and the feedback on character recognition is by morphing the user's individual characters into a cleaner version stored in an ink database. Therefore, the recognized expression is beautified but it still keeps handwritten appearance.

[3]https://dev.myscript.com/technology/math/

Chan and Yeung [2000] created a survey of the research in the field of recognition of mathematical expressions. Although it is an older paper, it shows the main principles, which remain valid till today. Especially the approaches used in structural analysis like parsing driven by a *context-free grammar* find its place even in different domains of structured handwriting. More recent research was surveyed by Zanibbi and Blostein [2012]. The paper also covers various topics related to the recognition: retrieval of mathematical expressions, user interfaces that seamlessly integrate recognition and retrieval, representation of the expressions, applications and especially exploitation in education.

### 1.2.2. Diagrams in General

Feng et al. [Feng et al., 2009] proposed a recognizer for on-line sketched *electric circuits*. Hypotheses for symbol segmentation and classifications are generated using Hidden Markov Models (HMM), and the selection of the best hypotheses subset relies on 2D dynamic programming. A drawback is an extensive search space due to a large number of hypotheses. This makes the system slow and prohibits it from practical use. Sezgin and Davis [Sezgin and Davis, 2005] used similar approach for recognition of objects in sketches from various domains like *stick-figures*, *UML diagrams*, or *digital circuits*. They use specific stroke orderings to reduce the search space. Although multiple HMMs are used to model different sketching styles and thus different natural stroke orderings, so called delayed strokes may impose a problem for this approach. ChemInk, a recognition system for *chemical formula sketches* [Ouyang and Davis, 2011], represents elements and bonds between them. A hierarchy of three levels of details is used: inkpoints, segments, and candidate symbols. The final recognition is performed by a joint graphical model classifier based on Conditional Random Fields (CRF), which combines features from the levels in the classification hierarchy. A similar approach was used by Qi et al. [Qi et al., 2005] to recognize simple diagrams. The advantage of such methods is the joint training of the classifier for all levels of features. It helps to incorporate the context into the classification, however it makes the training of the system more difficult. Our approach differs. Although we train the symbol classifier independently of the structure, we do not make any hard decisions at this point. Relations between the symbols are defined later with the help of context. Using binary predicates of the general *max-sum labeling problem* ([Werner, 2007]) rather than pairwise features does not require additional training and yields optimal solutions. The model is thus much simpler and more open to adaptations.

When dealing with arrow-connected diagrams, it is advantageous to exploit the key property of arrows – they link two symbols together. Kara and Stahovich [2004] realized that and designed SimuSketch, a sketch-based interface for Matlab Simulink software package. It examines the sequence of strokes to identify the arrows in the sketch first. The knowledge of the arrows is then used to segment the remaining symbols. They comprise of stroke clusters indicated by the position of the arrows. It remains only to classify them. However, it is difficult to recognize general arrows and thus they put constraints on how the arrows can be drawn – they may consist of exactly one or two strokes and the shape of the head is given. The classification is based on extraction of five *characteristic points* of the arrow. Angles between these points and the speed profile around them are examined. Stoffel et al. [2009] use exactly the same approach to recognize *commutative diagrams*. They are formed by arrows that join relatively simple mathematical expressions. As we already mentioned, the disadvantage of this approach is that it is difficult to recognize arrows this way and it is a source of errors.

Additionally, it cannot be used in domains where arrows consist of more strokes or the arrow heads have different than classical shapes. Our approach also exploits the property of arrows. The difference is that we do it in the opposite direction. We identify remaining symbols first and then find arrows linking them without examination of their exact appearance. Importance of arrows as connectors between two symbols has be also shown by Freeman and Plimmer [2007]. They incorporated a generic recognition technique of connectors into the recognition engine of their diagramming toolkit InkKit. They combined techniques for syntactic and semantic recognition for various connectors like directed arrows, undirected edges, or directed edges where the direction is given by the semantics. They experimented with domains of *UML Class diagrams* or *Entity Relationship (ER) diagrams*.

Liwicki and Knipping [2005] presented a system for recognizing sketched *digital logic circuits*. A recognized circuit can be graphically simulated later (i.e. the signal is propagated from the input to the output and the logical levels are visualized along the circuit). This system was developed for *school education* where it can practically demonstrate how individual logic gates and the whole circuits work. Their design is fully based on hand-drawing, which is fast and natural. Even the simulation is controlled by hand-drawings – inputs to circuits can be defined by writing numbers next to them. Circuit gate symbols are recognized using a multilayer perceptron network. Alvarado and Lazzareschi [2007] also studied the *digital logic circuits* and analysed considerable drawing style variation between students to design a recognizer for simulation software for schools. Digital logic circuits are diagrams with similar characteristics to flowcharts of finite automata. They consist of interconnected symbols – gates. The difference is that the symbols are not connected by arrows but undirected connectors. It makes recognition easier, because recognition of arrow heads is challenging and it is missing here. On the other hand, the symbols might be rotated and thus the symbol classifier needs to deal with this. This is common for different domains like the already mentioned electric circuits.

Finally, there were also attempts to develop *universal formalisms* for sketch recognition applicable to various domains. LADDER [Hammond and Davis, 2005] is a sketch description language that can be used to describe how shapes are drawn as well as the whole syntax specifying a domain. A *multi-domain sketch recognition* engine SketchREAD [Alvarado and Davis, 2004] is based on this language. Authors evaluated the capabilities of the engine on *family trees* and *electrical circuits*. The parsing is based on dynamically constructed Bayesian networks and it combines bottom-up and top-down algorithms. Although this framework laid the foundations of multi-domain sketch recognition, it has limitations. Individual shapes must be composed solely of predefined *primitives* according to a fixed *graphical grammar*. Individual strokes must be thus decomposed into primitives. Although the framework is designed to be recoverable from low-level errors, it still imposes an additional source of errors.

### 1.2.3. Flowcharts and Finite Automata

To our knowledge, little work has been published in the domains of flowcharts and finite automata. Lemaitre et al. [Lemaitre et al., 2011] proposed a grammar based recognition system for flowcharts which uses the DMOS (Description and MOdification of the Segmentation) method for structured document recognition. The applied grammatical language EPF (Enhanced Position Formalism) provides a syntactic and structural description of flowcharts, which is used for joint symbol segmentation and classification. Carton et al. [Carton et al., 2013] further improved the system by combining structural and statistical approaches; they exploited the nature of the symbols in flowcharts, which are *closed loops*. Such closed loops are detected first and classified later using the structural approach. Although statistics are used, it is hard to find a suitable threshold determining if a loop is really closed. Users often draw carelessly and the appearance of symbols can be far from closed loops. Additional difficulties are caused by the need to divide strokes into line segments. Experiments demonstrated that the grammar based approach has still troubles with a big uncertainty in the input. Experiments were performed on a benchmark database, which allows the comparison. Their grammar-based approach is difficult to adapt for a new domain since new grammatical rules need to be defined for each symbol class. An example of a grammatical description of a quadrilateral follows:

```
quadrilateral Q ::=
AT(wholePage) &&
oneEdge C1 &&
AT(edgeEnd C1) &&
oneEdge C2 &&
AT (edgeEnd C2) &&
oneEdge C3 &&
AT (edgeEnd C3) &&
oneEdge C4 &&
checkEdgesAreClose C1 C4 &&
addScore Q.
```

The work by Szwoch and Mucha [Szwoch and Mucha, 2013] is another effort to recognize flowcharts using grammars. The authors assume that symbols consist of single strokes, and this simplification forbids experiments on the benchmark database and thus it cannot be compared with other methods.

Delaye [Delaye, 2014] has recently introduced a purely statistical approach to diagram recognition based on strokes clustering and CRFs, where clusters represent graph nodes. A hierarchical model is used by applying several values of clustering thresholds. The graphs created are trees, and thus the task can be solved efficiently by the Belief Propagation algorithm, which makes the system extremely fast. However, the approach is purely statistical, which does not use information about the diagram structure. Inconsistent labelings can occur.

### 1.2.4. Alternative Approaches

Though there are few systems directly comparable to ours, interest in diagram design/sketching is evident. Miyao and Maruyama [Miyao and Maruyama, 2012] created a *flowchart designer* based on the iterative recognition principle. Input is processed in small pieces and immediate user feedback is awaited. If the user does not indicate any error in the recognition, it is considered as ground truth. It is further possible to connect symbols by gestures and to input text for a selected symbol. This works well for flowcharts since symbols are loopy. However, the system puts unnatural requirements on the user. A practical example of a successful diagram design based on iterative recognition where one symbol is being recognized at a time can be found in MS PowerPoint Ink Tools. Its functionality “Convert to Shapes” allows precisely this and can be thus used to create flowchart right in the presentations. Text can be added in a traditional way once the shape has been recognized.

In some cases, it is desired to keep the *sketchy appearance* of diagrams, thus smart sketching tools allowing common editing operations without any formalization of the input have been proposed [Arvo and Novins, 2005; Plimmer et al., 2010].

In conclusion, although there exist various systems for structured handwriting, there is no system for flowchart-like diagrams allowing practical use. Existing methods are either purely statistical and do not use diagram structure adequately, or they rely on grammars, which are too impractical for the minimalistic structure of diagrams.

## 1.3. Problem Formulation

As previously described, handwriting and drawing have many aspects. There exist different scenarios and goals how to use them. Therefore, we divide this section into two parts as we seek solutions for *formally defined diagrams* and *free-form sketches*.

When dealing with formally defined diagrams, we want to create a complete recognition system for on-line hand-drawn diagrams from domains of arrow-connected diagrams. We want to design a formal structural analysis, which would exploit the simplicity of the diagram structure. Our goal is to show that it is faster than grammar-based approaches and it is easier to adapt the system for a new domain.

When dealing with free-from sketches, we want to show that despite the fact that probably any recognition cannot be done in this case, there is still a lot we can do for the user using knowledge of machine learning and user interface design. Object selection and sketch rearrangement are two very common tasks, which could be performed much faster and with ease if automatic or semi-automatic segmentation of objects of interest was possible. We want to contribute into this problem.

The scientific goals of this thesis related to the two described topics are listed in the two following sections.

### 1.3.1. Goals in Diagram Recognition

- We want to create a complete recognition pipeline, which will undergo all necessary steps for recognition of on-line hand-drawn diagrams and its speed and precision allow for practical use.

- We see a grammar-based structural analysis as a cumbersome overkill for recognition of documents with such a simple structure as arrow-connected diagrams. We want to show that structural analysis based on the max-sum labeling problem choosing the optimal combination of symbol candidates can produce better consistent and valid solution and can be faster. We also want to show that this approach allows easier adaptation for new domains.

- Text/non-text separation is considered to be a good approach to divide the recognition problem and lower the complexity. We want to verify our hypothesis that it makes sense to use a biased classifier which makes a smaller error in the non-text class at the cost of a higher error in the text class. We argue that the recognizer is robust to remaining text strokes while missing strokes of symbols cause serious problems.

- Arrows have varying appearance and thus they are difficult to recognize using appearance based recognizers. We believe that it makes sense to exploit their key feature – they link two symbols together. We want to show that it is feasible and more accurate to consider each pair of previously detected non-arrow symbols and search for arrows as arbitrarily shaped connectors between two symbols. Here we show that arrow heads can be found using relative stroke positioning.

- Latest research has shown that it is possible and advantageous to generate artificial samples for training of a symbol classifier. It helps to balance the training database. We experimented with the promising Kinematic Theory and the distortion of the Sigma-Lognormal parameters. We show that it increases precision and robustness of classifiers for symbols from diagram domains. Especially, it helps to boost the rejection ability of the classifier.
- We consider an important goal to evaluate and to analyse the proposed recognizer. We need to compare our approach with state-of-the art alternatives using traditional criteria like correct stroke labelling. However, this is not informative enough and thus we compared whole diagram structures for better understanding of the quality of the recognition. Additionally, we analysed the effect of individual steps of the recognition pipeline to the overall performance and found out the causes of recognition failures.

### 1.3.2. Goals in Segmentation of Objects in Free-From Sketches

- We study how users sketch on interactive whiteboards, what these sketches look like, and how would users cluster strokes into objects. Intuitively, every user might have a different point of view on a sketch and also one user might have multiple points of view on one sketch depending on his intentions with the sketch.
- We explore the possibilities of trainable clustering algorithm to segment objects of interest. It proved to be suitable for segmentation of objects from predefined symbol classes in formal diagrams or objects having relatively stable appearance like individual words. We verified that it achieves promising results in the case that the objects are more loosely defined.
- Objects of interest might have fundamentally different characteristics from case to case and from user to user. Therefore, it is necessary to train several clustering algorithms to cover the variability in the free-form sketches. There is a need for a framework which allows to combine these pre-trained algorithms together. We verified our hypothesis that desired segmentation can be achieved in many cases by combining together algorithms trained for very general object types like rows, columns, words, compact images, visual subgroups, etc.
- We designed a user interface allowing to indicate the user's intentions. This information is essential to create a suitable combination of pre-trained clustering algorithms. It is crucial to allow the user to fix the segmentation if necessary and to manipulate with the segmented objects with ease. The emphasis is put on rearrangement of the whole sketch, which is very important during brainstorming sessions at interactive whiteboards.

## 1.4. Thesis Contribution

This thesis contributes into two related topics: recognition of structured arrow-connected diagrams and segmentation of objects of interest in free-from sketches. The first one was done by me and the second one was conducted in collaboration with Florian Perteneder during my internship in Media Interaction Lab, University of Upper

Austria. The thesis presents the advancements in both areas because they both belong to general problem of sketch understanding. Specifically, my contributions are the following:

1. **Benchmark database of flowcharts and finite automata.** There was a lack of good annotated data in the field of on-line handwritten diagrams. Therefore, we gathered and annotated our own data and created a publicly available database of on-line handwritten flowcharts and finite automata. It allows training of the recognition system as well as its testing and comparison with others. We believe that this helps to motivate the community to accelerate research in this field. The annotation provides information about individual symbols, relations between them and meaning of text blocks.

2. **Diagram recognition pipeline.** We designed a general recognition pipeline consisting the followings steps: 1) preprocessing, 2) text/non-text separation, 3) uniform symbol segmentation, 4) symbol classification, 5) arrow detection, 6) structural analysis, 7) text blocks detection and recognition. We contributed to each of the steps to create a recognizer with state-of-the-art performance in domain of flowcharts and finite automata [Bresler et al., 2016]. The major contribution related to the arrow detection and structural analysis are described later.

3. **Use of max-sum model for structural analysis.** We proposed a model for recognition by selection of symbol candidates, based on evaluation of relations between candidates using a set of predicates [Bresler et al., 2013, 2014]. This selection is formulated as an optimization max-sum problem where the goal is to maximize the sum of scores of selected symbol candidates that fulfil all the constraints given by the unary and binary predicates. It is suitable for simpler structures where the relations are explicitly given by the symbols, arrows in the case of diagrams. A grammar-based structural analysis proven to be superior for dealing with a complex recursive structure of mathematical expressions seems to be an over-kill when dealing with this simple structure. We showed that it is slower, harder to adapt for new domains, and achieves worse results in our comparison of state-of-the-art methods. The advantage of our approach is that there is no need for additional training. Although the max-sum problem is generally an NP-hard problem, the instances generated during diagram recognition are solved relatively fast.

4. **Arrow detector based on relative stroke positioning.** We showed that common appearance-based classifiers used to recognize symbols from various domains are not suitable for recognition of arrows. These *uniform* symbols have relatively stable appearance. On the other hand, arrows have varying sizes and shapes. However, they have a very important property – they connect two uniform symbols. We exploited this property and designed a specialized arrow detector [Bresler et al., 2015a]. It searches for arrows as arbitrarily shaped connectors between two symbols. A head of the arrow (and thus its direction) is detected using a Long Short Term Memory Recurrent Neural Network (LSTM-RNN) as a classifier based on relative positions of the head strokes and arrow end-points.

The joint contribution is following:

5. **Segmentation of objects of interest in free-form sketches.** We designed a tool for segmentation and manipulation of objects of interest in free-form

sketches [Perteneder et al., 2015]. We used a Single-Linkage Agglomerative Clustering (SLAC) with a trained distance function. First, we conducted a background study to find out what users consider as objects of interest. It turned out that there usually exist multiple points of view on one particular sketch. Second, we trained the clustering algorithm to segment different kinds of object. This way, we obtained several clustering tools expressing existing points of view, so called perspectives. We then combined these perspectives to obtain desired result indicated by an initial selection. Third, we designed several tools to allow the user to seamlessly modify the created cluster and to manipulate with them.

## 1.5. Thesis Organization

This Chapter 1 introduces the problem of recognizing handwritten diagrams, provides a survey on state-of-the-art methods, sets out scientific goals, and summarizes contributions of this thesis. The rest of the thesis is organized as follows:

- **Chapter 2** describes the structure of diagrams supported by the recognition system. It introduces the selected representative domains of flowcharts and finite automata with their specifications and benchmark databases used for training and testing of the system. It describes how we created our benchmark databases. We also discuss the possibility to adapt the recognition system for other domains.
- **Chapter 3** presents the recognition pipeline of our recognition system. It describes in detail individual steps of the pipeline and provides their intermediate evaluation.
- **Chapter 4** evaluates the whole recognition system on the benchmark database and shows a comparison with other state-of-the-art methods. It also provides thorough analysis of the system including failure analysis.
- **Chapter 5** deals with free-form sketching and segmentation of objects in documents without any particular structure.
- **Chapter 6** contains conclusion of our work, summarizes the contribution of the thesis, and outlines possible future work.

# 2. Diagram Structure and Supported Domains with Benchmark Databases

The diagram recognition system we propose is general and can be used in several domains of arrow-connected diagrams. Supported diagrams consist of symbols with a relatively stable appearance (called *uniform symbols*), interconnected by arrows. Arrows and uniform symbols may consist of arbitrary number of strokes and both are possibly assigned a text label: text inside the uniform symbol or in the vicinity of the arrow. The domain syntax can bring additional constraints, e.g., forbid connecting some symbols. Although the described structure is very simple, we can find it in various diagram domains like UML use case diagrams, Simulink diagrams, Concur Task Trees, or business process diagrams (see illustrations in Figure 2.2). We worked in two of the most common diagram domains, flowcharts (FC) and finite automata (FA). This chapter describes these domains in detail and introduces the benchmark databases we used for training and testing of the proposed recognizer.

The recognizer was initially developed for flowcharts. This choice was motivated by an existing benchmark database created and published by Awal et al. [2011]. Later, we decided to test the adaptability of the system for another domains. Here we chose finite automata. Unfortunately, there was no benchmark database of handwritten finite automata. Therefore, we decided to create our own database. With the experience of benchmark database creation we also decided to create a new benchmark database of flowcharts to obtain more data of a higher quality.

The rest of this chapter is organized as follows. Section 2.1 and 2.2 introduce the domains of flowcharts and finite automata, respectively. They also describe the benchmark databases we used. In Section 2.3, we discuss the ability of the proposed system to be adapted for other domains. Domains with a structure violating the requirements we put on the supported structure are especially interesting. Finally, Section 2.4 describes the process of creation of our benchmark databases. Statistics of all the used databases are overviewed in Table 2.1.

## Chapter Outline

## 2.1. Flowcharts

Flowcharts are very general diagrams used to express arbitrary processes or algorithms. Although the set of symbols is unlimited due to the fact that the user may define her/his own symbols, there exists a subset of the most frequently used ones. In our work, we consider six symbol classes listed in Figure 2.1 together with an example of a whole flowchart. This small subset is sufficient to assemble diagrams representing the most common and famous algorithms (factorial numbers, bubble sort, neural network training, etc.).

We use two benchmark database to train and test the recognizer in the domain of flowcharts. The first one is the already mentioned benchmark database published by Awal et al. [Awal et al., 2011]. We further reference it as FC_A. The database consists of 327 diagrams drawn by 35 writers. Predefined diagram patterns representing well known algorithms were used. The samples are divided into training and test datasets. Several state-of-the-art methods were already tested on this dataset and thus it allows comparison. The biggest disadvantage of the database is the lack of annotations providing information about the diagram structure. Only individual symbols are identified and no temporal information is available. Additionally, the data is of low quality mostly due to a low sampling frequency.

These deficiencies have motivated us to collect our own flowchart database and make it public[1]. We reference this database as FC_B. We collected 28 diagram patterns drawn by 24 writers, resulting in 672 samples. They were divided into training, validation, and test datasets. Some of the patterns were taken from the FC_A database, and others represent procedures for daily tasks. The database contains annotation of symbols and relations among them. Arrows are provided with connection points and heads annotated. Text blocks have their meaning attached.

## 2.2. Finite Automata

These diagrams are also called *Finite State Machines* (FSM) and represent mathematical models of computations used to design both computer programs and sequential logic circuits. A particular finite automaton is defined by a list of its states, and the triggering condition for each transition. Its behaviour can be observed in many devices in modern society that perform a predetermined sequence of actions depending on a sequence of events with which they are presented.

The finite automata domain includes three uniform symbol classes: *state* (a circle), *final state* (two concentric circles), and *initial arrow* (straight arrow entering the initial state). The text is usually simple – often just a single letter naming the state or indicating an input attached to the arrow. It may also contain a lower index. Arrows are typically curved, except the initial arrow which does not act as a connector of two states. An example is shown in Figure 1.2.

No publicly available database of on-line sketched finite automata was known at the time we started our research. We gathered and annotated our own database (referenced as the FA database) and made it public[1]. The database contains samples of 12 diagram patterns drawn by 25 users, which results in 300 diagrams divided into training, validation, and test datasets. As in the case of the FC_B database, it contains annotations of symbols and relations among them. Additionally, arrows are provided with connection points and heads annotated. Text blocks have their meaning attached.

[1] http://cmp.felk.cvut.cz/~breslmar/diagram_database

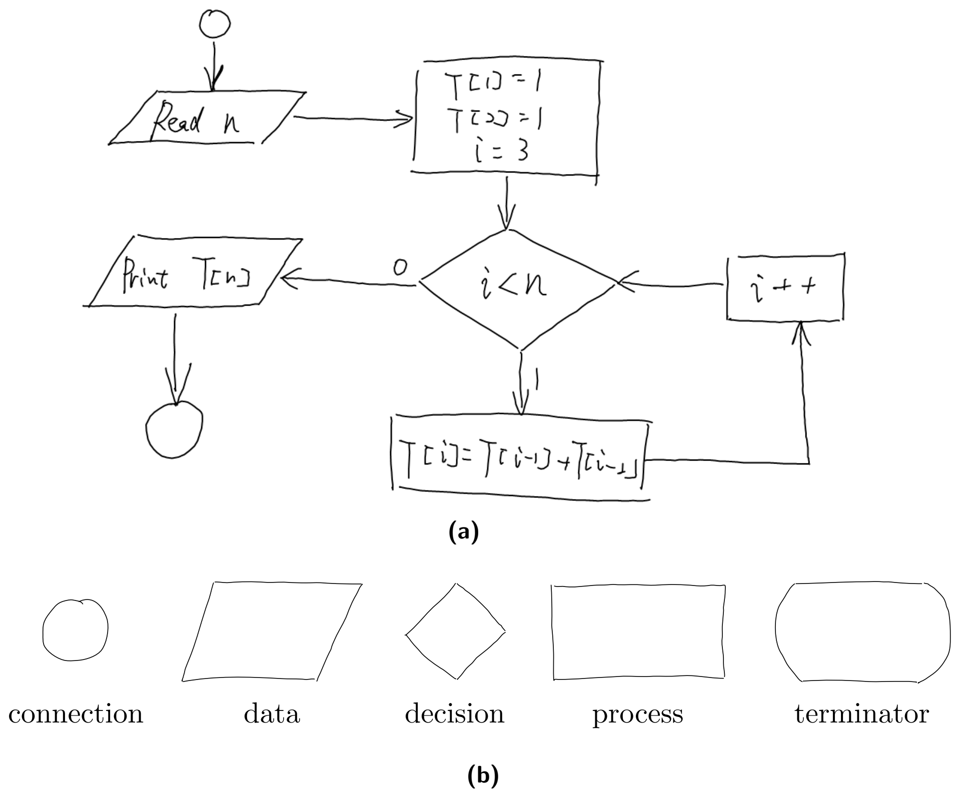


**Figure 2.1.** Example of a flowchart (a) with examples of uniform symbol classes (b).

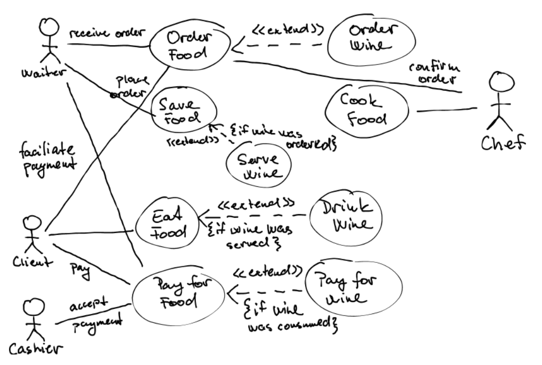
| Database | # of writers | # of patterns | # of diagrams | | | |
|---|---|---|---|---|---|---|
| | | | Training | Validation | Test | Sum |
| FC_A | 35 | 28 | 200 | – | 127 | 327 |
| FC_B | 24 | 28 | 280 | 196 | 196 | 672 |
| FA | 25 | 12 | 132 | 84 | 84 | 300 |

**Table 2.1.** Statistics of the used benchmark databases.

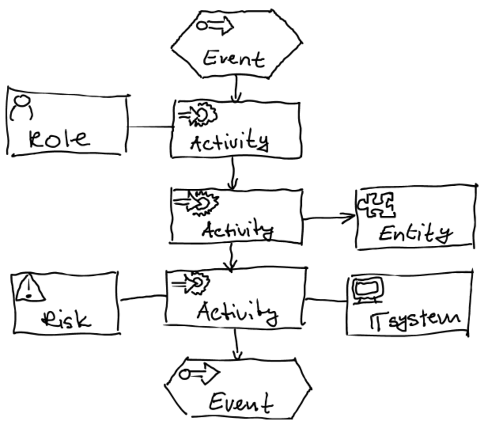
## 2.3. Other Compatible Domains

There exists a range of diagram domains with supported simple structure of arrow-connected diagrams. We already described the structure and introduce some examples of such domains. Illustrative examples of diagram from these domains are shown in Figure 2.2. Notice that uniform symbols or arrows from individual domains have some very specific characteristics. Nevertheless, an adaptation of the recognition system for these domains is straightforward because the recognition pipeline will remain exactly the same with the same model used in the structural analysis. All we need to do is to retrain or slightly adjust the following: the classifier separating text/non-text, the distance function used for over-segmentation, symbol classifier.

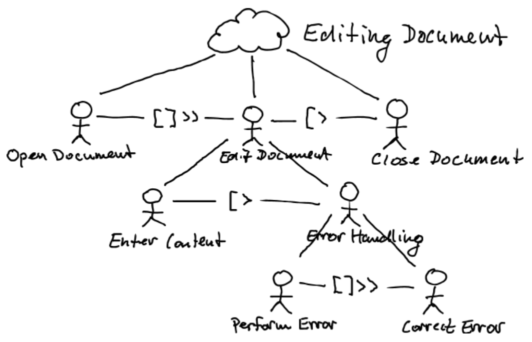
However, there exist diagrams with a structure which is an extension of the basic constructs. An example is an UML class diagram with structured text inside symbols. In this case, it is not enough to identify text blocks when the structure is known. Some kind of a hierarchical recognition would be necessary. The inner structure of each symbol would have to be recognized using another model for structural analysis. Of course, there are many domains beyond the scope of arrow-connected diagrams. An



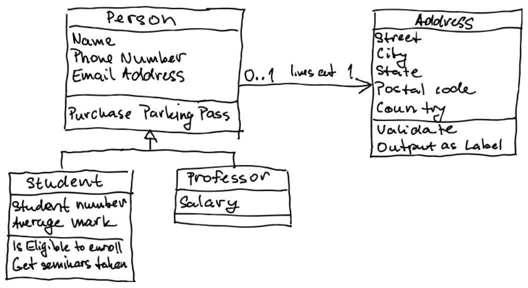
**(a)** UML Use Case diagram



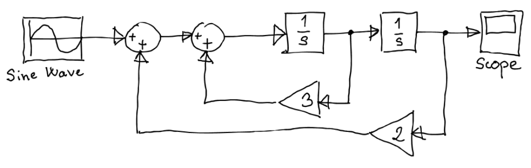
**(b)** Business Process



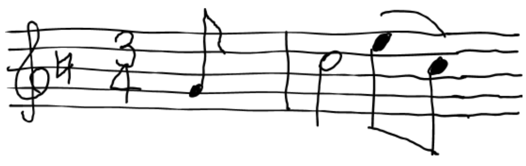
**(c)** Concur Task Tree (CTT)



**(d)** UML Class Diagram



**(e)** Simulink diagram



**(f)** Music Score

**Figure 2.2.** Examples of drawings from domains with potential application of our recognizer. a)–e) show domains of arrow-connected diagrams where the adaptation is more or less straightforward. However, a) shows that arrows might have dashed shafts which would require an adjustment of the arrow detector; b) shows that a uniform symbol might have a small glyph in its corner which would require adjustment of the uniform symbol classifier; c) shows that arrows might be replaced by undirected connectors possibly provided with a distinguishing glyph; d) shows that text inside a symbol might be further structured, which would require adjustment in the model of structural analysis; e) shows that arrows might be connected in nodes. Finally, f) shows a music score where the key relation between uniform symbols and arrows is replaced by the relation between staff lines and notes.

extension for these would require modification of the max-sum model used for structural analysis. It is possible if the fundamental relation between arrows and symbols can be replaced by another relation defining the structure of the handwriting. For example, the relation between notes and staff lines in the case of music scores.

## 2.4. Benchmark Database

Creation of a benchmark database consists of two steps: data collection and data annotation. We created an application for each of these two steps. They are implemented in C# and are meant to be run on a Tablet PC with Windows operating system.

We have gained experience with creation of benchmark databases during our work on recognition of mathematical expressions [Stria et al., 2012]. [Wolin et al., 2007] presented their effort to develop tools for a convenient annotation of diagrams from the digital logic circuits domain. These diagrams have similar characteristics with our chosen domains. We use very common approaches for individual tasks of the annotation process.

### 2.4.1. Data Collection

We asked the same 25 writers to draw diagrams from both domains (one writer could not participate in the collecting of flowcharts). We tried to cover the biggest spectrum of different writers to embrace as much writing styles as possible. There were people with 8 different nationalities in age of 16–58, 10 of them were female, and two of them were left-handed. To assure that the writers will draw meaningful diagrams of a reasonable complexity, we prepared patterns which were supposed to be redrawn by the writers. However, each writer was told to draw it in a way natural to him.

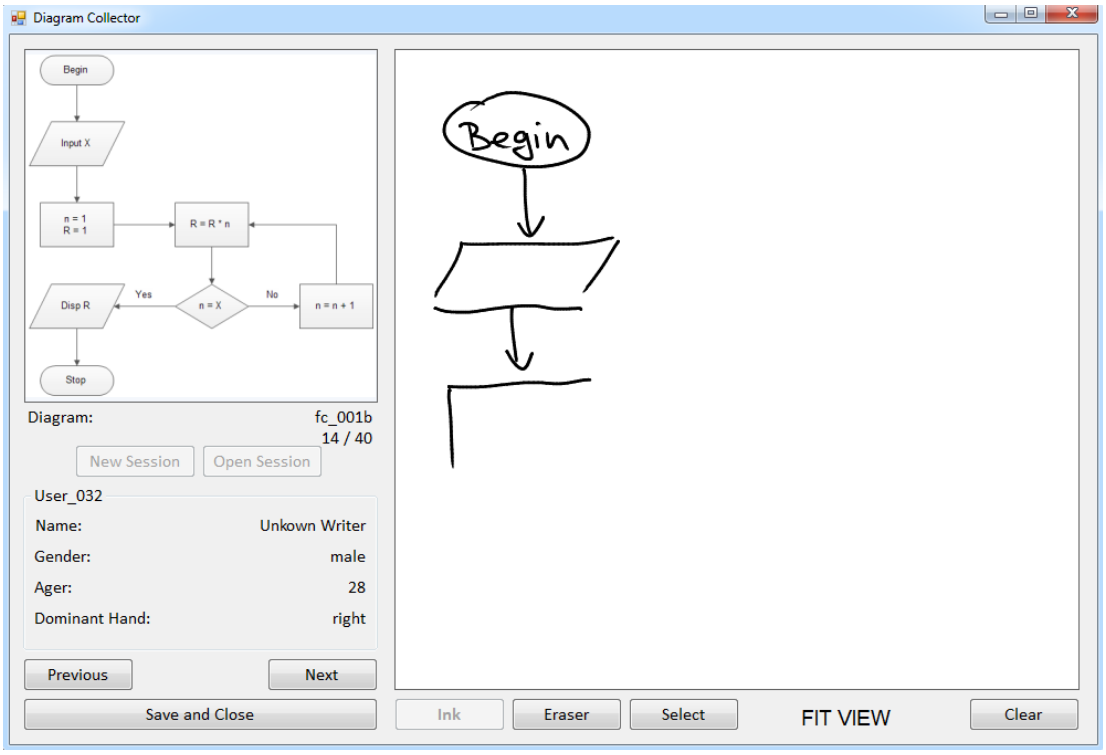


Diagram: fc_001b
14 / 40

User_032

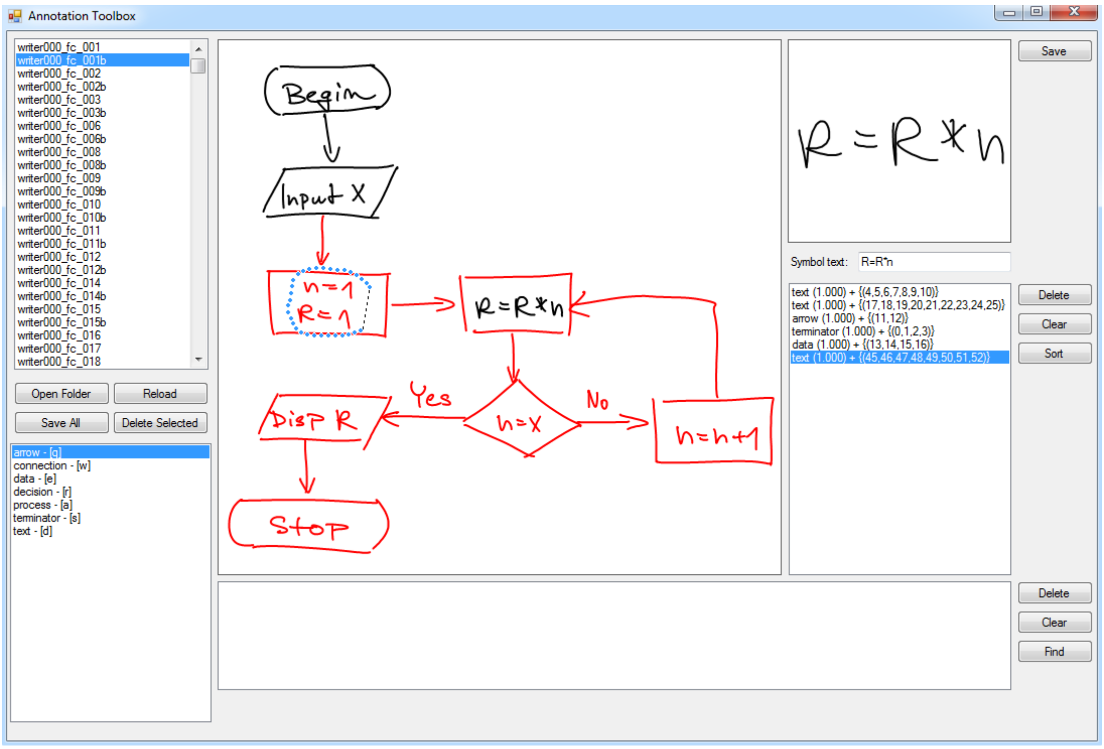
| | |
|---|---|
| Name: | Unkown Writer |
| Gender: | male |
| Ager: | 28 |
| Dominant Hand: | right |

Figure 2.3. Screenshot of the application for data collection.

Before the data collection may begin, the writer must create a new session where he fills all important personal information like name, gender, age, nationality, and dominant hand (the name is removed before the database is published). The application can always save and load the whole session. The writer always sees a pattern he is expected to redraw. If the drawing canvas is not big enough, it is possible to scroll easily by moving the cursor towards an edge of the canvas in a desired direction. Zoom is not allowed to keep a consistent scale within one particular drawing. However, to allow the writer to review the whole diagram at once, it is possible to fit the drawing into the canvas. It is done by holding the cursor on "FIT VIEW" label. It is just for reading and just for the time when the label is hold. The user can switch to eraser to perform some corrections or the whole canvas can be cleared. The writer can move to another diagram pattern or save and close the session when he is done.

## 2.4.2. Data Annotation

The annotation is done in two major steps: a) annotation of individual entities (uniform symbols, arrows, and text blocks); b) annotation of relations between the entities. To annotate an entity, the user must select strokes comprising the symbol using a lasso tool. Selected strokes are highlighted. While a subset of strokes is selected a symbol class is assigned by pressing the corresponding key. All annotated entities are listed in the right side of the window and new one is automatically selected. No stroke can be part of multiple entities. Therefore, the used symbols cannot be selected again and it is convenient to annotate text blocks inside a uniform symbols first. Unused strokes are red and used black. The user interface for annotating individual entities is shown in Figure 2.4.



writer000_fc_001
writer000_fc_001b
writer000_fc_002
writer000_fc_002b
writer000_fc_003
writer000_fc_003b
writer000_fc_006
writer000_fc_006b
writer000_fc_008
writer000_fc_008b
writer000_fc_009
writer000_fc_009b
writer000_fc_010
writer000_fc_010b
writer000_fc_011
writer000_fc_011b
writer000_fc_012
writer000_fc_012b
writer000_fc_014
writer000_fc_014b
writer000_fc_015
writer000_fc_015b
writer000_fc_016
writer000_fc_017
writer000_fc_018

arrow - [q]
connection - [w]
data - [e]
decision - [r]
process - [a]
terminator - [s]
text - [d]

Symbol text: R=R*n

text (1.000) + {(4,5,6,7,8,9,10)}
text (1.000) + {(17,18,19,20,21,22,23,24,25)}
arrow (1.000) + {(11,12)}
terminator (1.000) + {(0,1,2,3)}
data (1.000) + {(13,14,15,16)}
text (1.000) + {(45,46,47,48,49,50,51,52)}

**Figure 2.4.** Screenshot of the application for data annotation showing the annotation of individual entities. The lasso tool is visualized with a chain of blue dots. Symbol classes with assigned keys are shown in the bottom left list. The upper right canvas changes context according to the selected entity. It visualizes the entity and allows to provide some details. If a text block is selected, its meaning can be entered below the visualization canvas.

Upper right canvas visualizes a selected entity. It allows to provide some additional details about the entity. If a text block is selected, the user is supposed to write down its meaning. We use LaTeX notation in the case of subscripts or other mathematical expressions. If an arrow is selected, the user is supposed to annotate its connection points. It is done by right clicking in the window. The start point is selected first, selection of the end point follows then. Additionally, a new window pops up if an arrow is selected to annotate which strokes form its head (see Figure 2.5). It will not appear if the arrow head was already annotated. The remaining strokes automatically form its shaft. Each stroke can be split into two by clicking with the middle mouse button. The closest stroke point is selected and it belongs to both newly created strokes. It is useful when the head and the shaft of an arrow are drawn by one stroke.

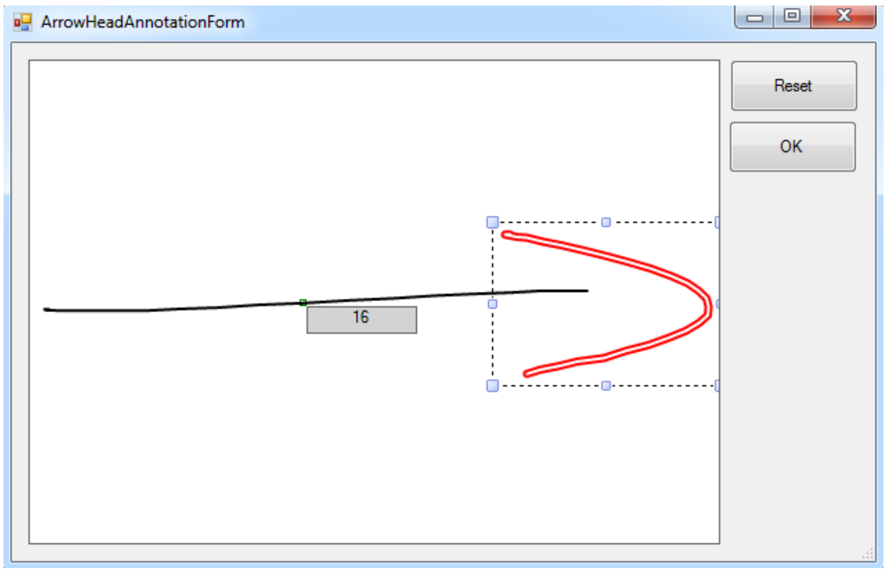
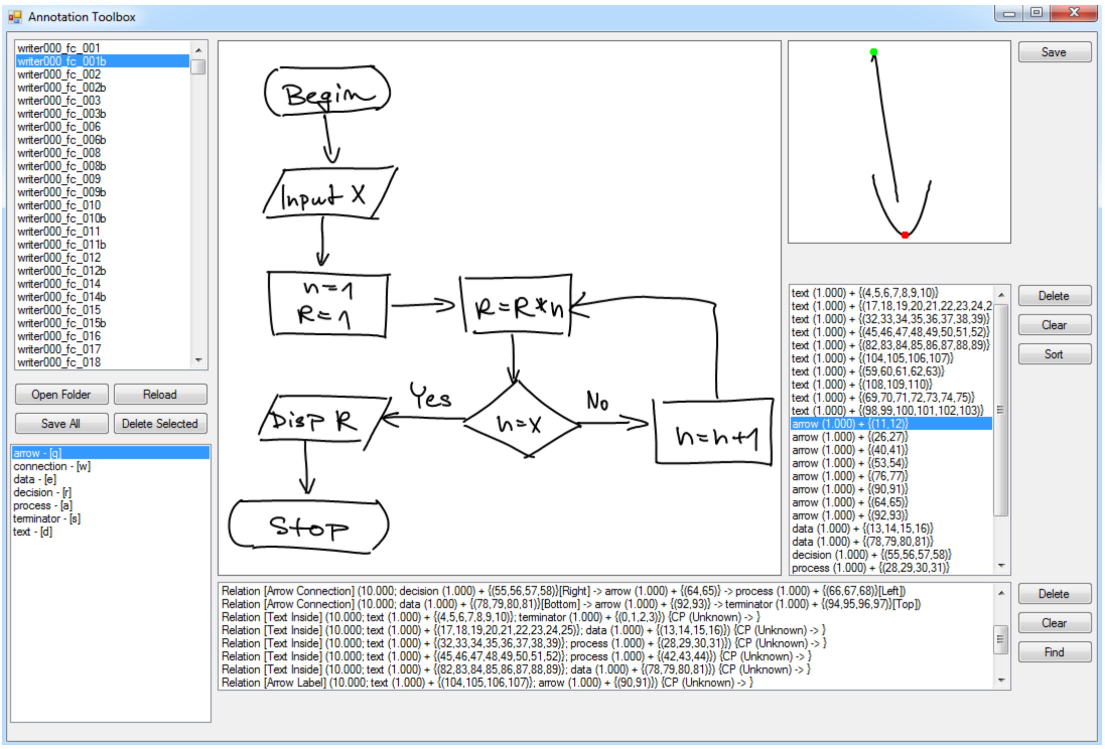


**Figure 2.5.** Screenshot of the window for annotation of arrow head. Selection is made using a lasso tool. Strokes comprising the head are red. Each stroke can be split by selecting a split point with the middle mouse button. It must be done to annotate a head drawn by one stroke along with the shaft. The index of the selected splitting point is shown in the grey box.

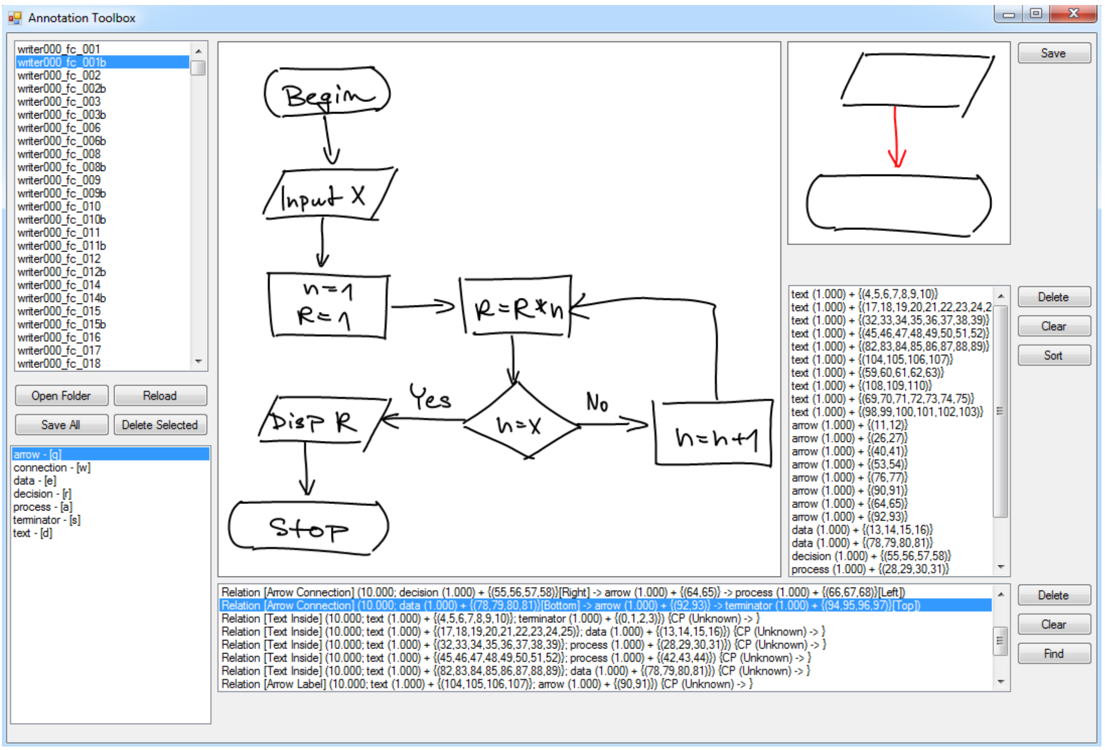
When all the entities are annotated and all information is provided, the user can run automatic detection of relation between the annotated entities. Text blocks are assigned to symbols they label and arrows are assigned two symbols they connect. This detection uses the annotated connection points of arrows. Therefore, the two closest symbols can be found easily. The whole detection of relations is very reliable. However, if there is an error, the user can remove wrong relations and create new ones manually by selecting participating symbols and the type of the relation. Before the relation detection is done, the application verifies if the user provided correct and complete annotation: it is checked that all strokes are part of some entity, each text block has its meaning assigned, each arrow has its connection points and the head annotated. Each annotated diagram is stored in a separate InkML file.



```
text (1.000) + {(4,5,6,7,8,9,10)}
text (1.000) + {(17,18,19,20,21,22,23,24,2
text (1.000) + {(32,33,34,35,36,37,38,39)}
text (1.000) + {(45,46,47,48,49,50,51,52)}
text (1.000) + {(82,83,84,85,86,87,88,89)}
text (1.000) + {(104,105,106,107)}
text (1.000) + {(59,60,61,62,63)}
text (1.000) + {(108,109,110)}
text (1.000) + {(69,70,71,72,73,74,75)}
text (1.000) + {(98,99,100,101,102,103)}
arrow (1.000) + {(11,12)}
arrow (1.000) + {(26,27)}
arrow (1.000) + {(40,41)}
arrow (1.000) + {(53,54)}
arrow (1.000) + {(76,77)}
arrow (1.000) + {(90,91)}
arrow (1.000) + {(64,65)}
arrow (1.000) + {(92,93)}
data (1.000) + {(13,14,15,16)}
data (1.000) + {(78,79,80,81)}
decision (1.000) + {(55,56,57,58)}
process (1.000) + {(28,29,30,31)}
```

```
Relation [Arrow Connection] (10.000; decision (1.000) + {(55,56,57,58)}[Right] -> arrow (1.000) + {(64,65)} -> process (1.000) + {(66,67,68)}[Left])
Relation [Arrow Connection] (10.000; data (1.000) + {(78,79,80,81)}[Bottom] -> arrow (1.000) + {(92,93)} -> terminator (1.000) + {(94,95,96,97)}[Top])
Relation [Text Inside] (10.000; text (1.000) + {(4,5,6,7,8,9,10)}; terminator (1.000) + {(0,1,2,3)}) {CP (Unknown) -> }
Relation [Text Inside] (10.000; text (1.000) + {(17,18,19,20,21,22,23,24,25)}; data (1.000) + {(13,14,15,16)}) {CP (Unknown) -> }
Relation [Text Inside] (10.000; text (1.000) + {(32,33,34,35,36,37,38,39)}; process (1.000) + {(28,29,30,31)}) {CP (Unknown) -> }
Relation [Text Inside] (10.000; text (1.000) + {(45,46,47,48,49,50,51,52)}; process (1.000) + {(42,43,44)}) {CP (Unknown) -> }
Relation [Text Inside] (10.000; text (1.000) + {(82,83,84,85,86,87,88,89)}; data (1.000) + {(78,79,80,81)}) {CP (Unknown) -> }
Relation [Arrow Label] (10.000; text (1.000) + {(104,105,106,107)}; arrow (1.000) + {(90,91)}) {CP (Unknown) -> }
```

**(a)** Annotation of arrow's connection points when an arrow is selected.



```
text (1.000) + {(4,5,6,7,8,9,10)}
text (1.000) + {(17,18,19,20,21,22,23,24,2
text (1.000) + {(32,33,34,35,36,37,38,39)}
text (1.000) + {(45,46,47,48,49,50,51,52)}
text (1.000) + {(82,83,84,85,86,87,88,89)}
text (1.000) + {(104,105,106,107)}
text (1.000) + {(59,60,61,62,63)}
text (1.000) + {(108,109,110)}
text (1.000) + {(69,70,71,72,73,74,75)}
text (1.000) + {(98,99,100,101,102,103)}
arrow (1.000) + {(11,12)}
arrow (1.000) + {(26,27)}
arrow (1.000) + {(40,41)}
arrow (1.000) + {(53,54)}
arrow (1.000) + {(76,77)}
arrow (1.000) + {(90,91)}
arrow (1.000) + {(64,65)}
arrow (1.000) + {(92,93)}
data (1.000) + {(13,14,15,16)}
data (1.000) + {(78,79,80,81)}
decision (1.000) + {(55,56,57,58)}
process (1.000) + {(28,29,30,31)}
```

```
Relation [Arrow Connection] (10.000; decision (1.000) + {(55,56,57,58)}[Right] -> arrow (1.000) + {(64,65)} -> process (1.000) + {(66,67,68)}[Left])
Relation [Arrow Connection] (10.000; data (1.000) + {(78,79,80,81)}[Bottom] -> arrow (1.000) + {(92,93)} -> terminator (1.000) + {(94,95,96,97)}[Top])
Relation [Text Inside] (10.000; text (1.000) + {(4,5,6,7,8,9,10)}; terminator (1.000) + {(0,1,2,3)}) {CP (Unknown) -> }
Relation [Text Inside] (10.000; text (1.000) + {(17,18,19,20,21,22,23,24,25)}; data (1.000) + {(13,14,15,16)}) {CP (Unknown) -> }
Relation [Text Inside] (10.000; text (1.000) + {(32,33,34,35,36,37,38,39)}; process (1.000) + {(28,29,30,31)}) {CP (Unknown) -> }
Relation [Text Inside] (10.000; text (1.000) + {(45,46,47,48,49,50,51,52)}; process (1.000) + {(42,43,44)}) {CP (Unknown) -> }
Relation [Text Inside] (10.000; text (1.000) + {(82,83,84,85,86,87,88,89)}; data (1.000) + {(78,79,80,81)}) {CP (Unknown) -> }
Relation [Arrow Label] (10.000; text (1.000) + {(104,105,106,107)}; arrow (1.000) + {(90,91)}) {CP (Unknown) -> }
```

**(b)** Visualization of a selected relation used for visual verification of the annotation.

**Figure 2.6.** Screenshots of the application for data annotation. The upper right canvas changes context according to the selected entity. When an arrow is selected (a) the user is expected to annotate the arrow's connection points by right clicking. The closest stroke point is selected. Start point (green) is selected first and end point (red) second. When a relation is selected (b) all interacting symbols are visualized. The arrow connecting two symbols is shown in red to visually separate the symbols.

# 3. Diagram Recognition

The goal is to find a diagram structure, which describes the sketched diagram syntactically. The system has to identify individual symbols and relations between them during the process of recognition. Additionally, text must be recognized in a way that it is divided into logical text blocks with known meaning. It is known to be a difficult chicken-egg problem to find the symbols and the structure. The diagram structure can be found by assembling symbols together. However, it is difficult to segment and recognize the symbols correctly without knowing the structure. Therefore, we find symbol candidates first by performing so called segmentation by classification using over-segmentation. Without doing hard decisions in this step, we let the structural analysis to choose the best combination of the symbol candidates able to assemble a valid diagram structure.

The described approach may be divided into three steps: symbol over-segmentation, symbol classification, and structural analysis. Before these can be performed it is wise to do several preprocessing steps including normalization of the input. It is also desired to divide the problem and separate text from other strokes. Finally, the text can be recognized after the diagram structure is known as post-processing.

This chapter first introduces our demo application illustrating the recognition process and possibilities with the formalized output in Section 3.1. The rest of the chapter then follows the recognition pipeline depicted in Figure 3.3 and illustrated by an example in Section 3.2. Specifically: Section 3.3 shows preprocessing steps, Section 3.4 describes our text/non-text separation, Section 3.5 explains the symbol segmentation through strokes clustering, Section 3.6 introduces our symbol detectors, Section 3.7 presents the structural analysis, and Section 3.8 explains detection and recognition of text blocks. Finally, a detailed description of the adaptation of the system from flowcharts to finite automata is provided in Section 3.9. It illustrates the adaptability of the system.

## Chapter Outline

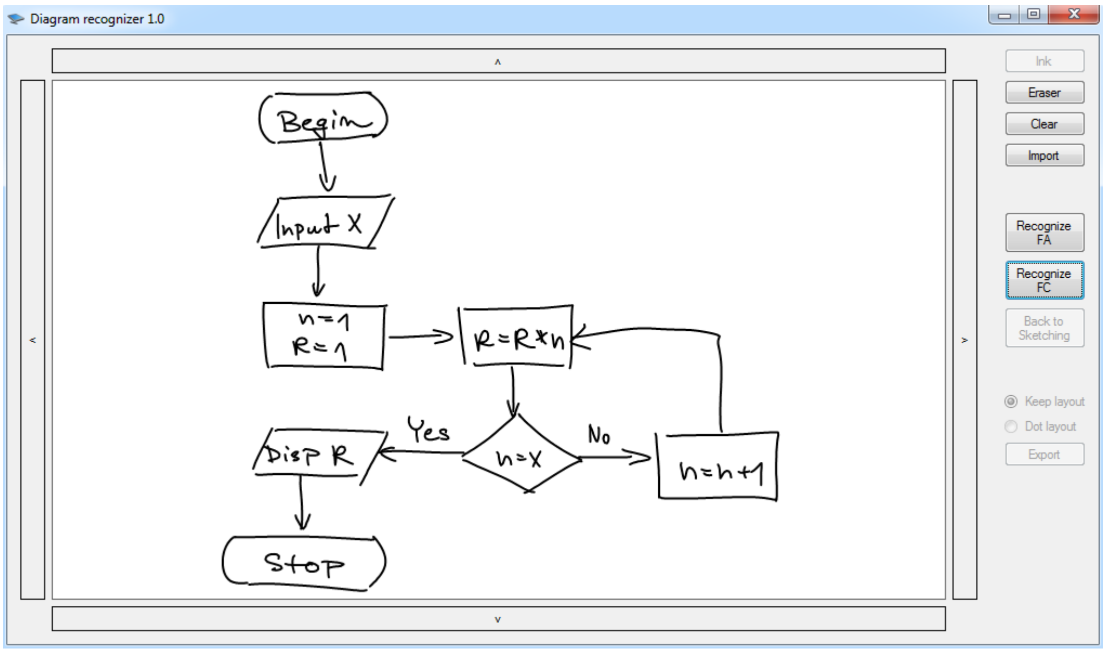
## 3.1. Demo Application

We created a demo application to show the capabilities of our diagram recognizer. It allows the user to draw or import a diagram from an InkML file (e.g. one of the diagrams from the benchmark databases). The diagram can be then recognized and its beautified version can be saved as an image. We encourage the reader to experiment with our demo application available at http://cmp.felk.cvut.cz/~breslmar/diagram_recognizer/. We show it here as a motivation for the remainder of this chapter.

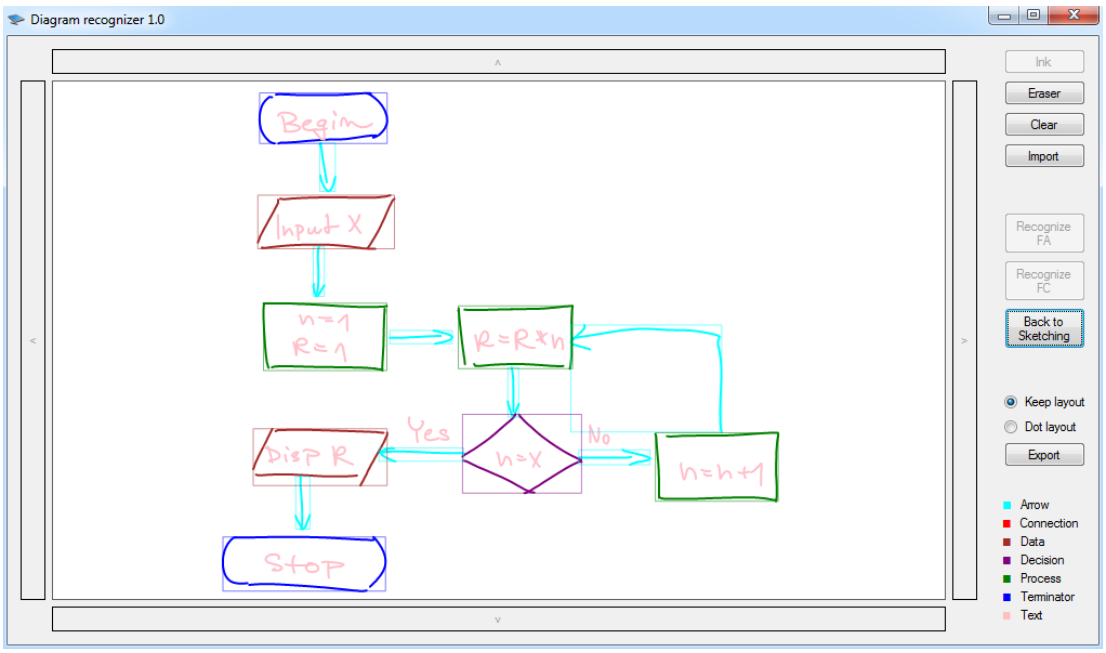
There are two recognizers running in the background: flowchart recognizer trained on the FC_B database and finite automata recognizer trained on the FA database. The user can thus draw or import a diagram from one of these two domains and then press the corresponding button to perform the recognition. The preview of the recognition result is displayed first: the strokes are coloured based on the symbol they were assigned to and individual symbols are additionally highlighted by their bounding boxes. The user can go back to the diagram editing mode, redraw the diagram, and perform the recognition again. Once he is satisfied with the result, the recognized and beautified diagram can be exported either to the PNG image or to the representation of DOT language. GraphViz library is used for the visualization and the user can choose whether the layout of symbols should correspond to the drawn layout or if the layout manager of GraphViz should be used. Figure 3.1 contains screenshots from the demo application with explanation of the user interface.

If MS Visio is installed on the computer it is also possible to export a recognized flowchart into it. In that case, MS Visio is automatically opened and the diagram is automatically created. This software for diagram design can be then used to further adjust the diagram – change graphical design or labels, reposition symbols or completely modify the diagram. Figure 3.2 compares two possible outputs: GraphViz visualization and MS Visio design. Finally, the DOT representation of the diagram used in this section as an example is as follows:

```
digraph flowchart {
graph [ dpi = 300 ]
layout = neato;
splines = ortho;
node [shape = ellipse]; S0 [ label = "Begin" pos="0.9,-0.2!" ];
node [shape = parallelogram]; S1 [ label = "Input x" pos="0.9,-1.1!" ];
node [shape = rectangle]; S2 [ label = "n=7 R=1" pos="0.9,-2.1!" ];
node [shape = parallelogram]; S3 [ label = "Disp R" pos="0.9,-3.1!" ];
node [shape = ellipse]; S4 [ label = "Stop" pos="0.7,-4!" ];
node [shape = rectangle]; S5 [ label = "R=R*n" pos="2.6,-2.1!" ];
node [shape = diamond]; S6 [ label = "h=X" pos="2.7,-3.1!" ];
node [shape = rectangle]; S7 [ label = "h=h+1" pos="4.5,-3.2!" ];
S7 -> S5 [ label = "" ];
S5 -> S6 [ label = "" ];
S6 -> S3 [ label = "yes" ];
S0 -> S1 [ label = "" ];
S1 -> S2 [ label = "" ];
S2 -> S5 [ label = "" ];
S3 -> S4 [ label = "" ];
S6 -> S7 [ label = "No" ]; }
```



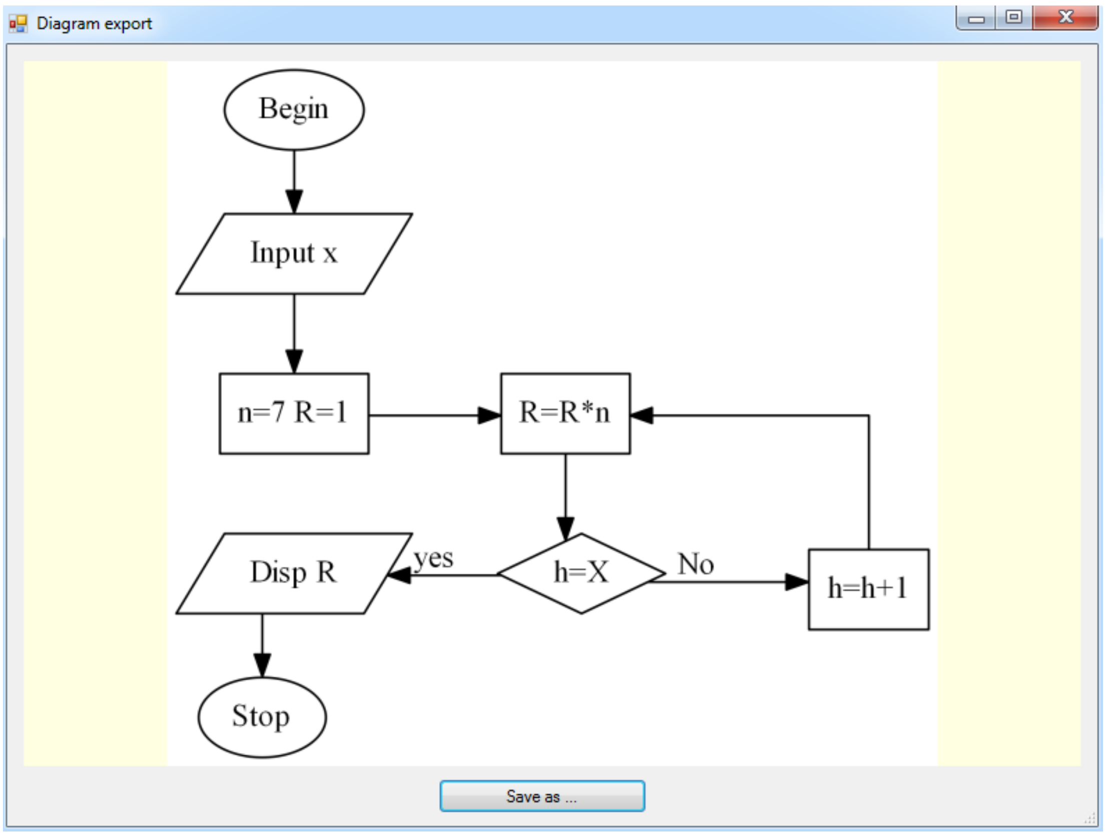
**(a)** Sketching mode – the user can draw a diagram using an ink input device. It is possible to scroll over the canvas using the the buttons on its edges. The recognition can be run by pressing a button of the corresponding domain.



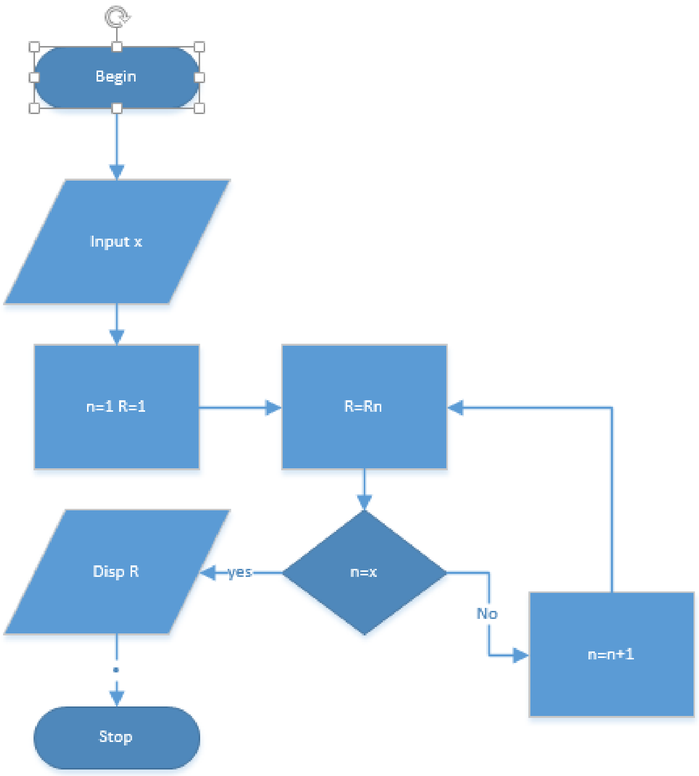
| | |
|---|---|
| 🟦 | Arrow |
| 🟥 | Connection |
| 🟥 | Data |
| 🟪 | Decision |
| 🟩 | Process |
| 🟦 | Terminator |
| 🟥 | Text |

**(b)** Recognition preview mode – recognized diagram entities are highlighted. The user can export the diagram using GraphViz visualization or can go back to the sketching mode.

**Figure 3.1.** Screenshots of the demo application: a) sketching mode where the diagram can be drawn or imported, b) recognition preview mode where the diagram can be exported.



**(a)** Window with GraphViz visualization. It can be directly saved as a PNG image or DOT representation.

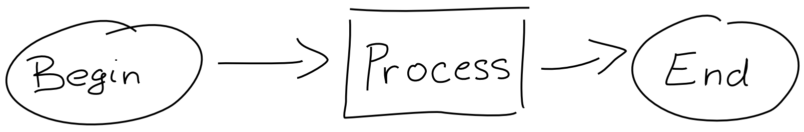


**(b)** Diagram exported into MS Visio.

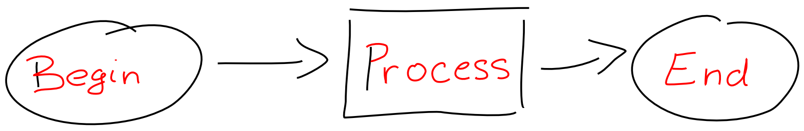
**Figure 3.2.** Screenshots of the two possible export options: a) visualization done by GraphViz, b) the diagram exported into MS Visio.

## 3.2. Pipeline Illustration by Example

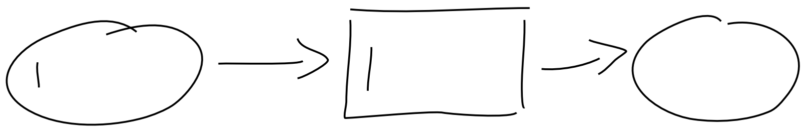
1. **Input**



2. **Text/non-text classification**

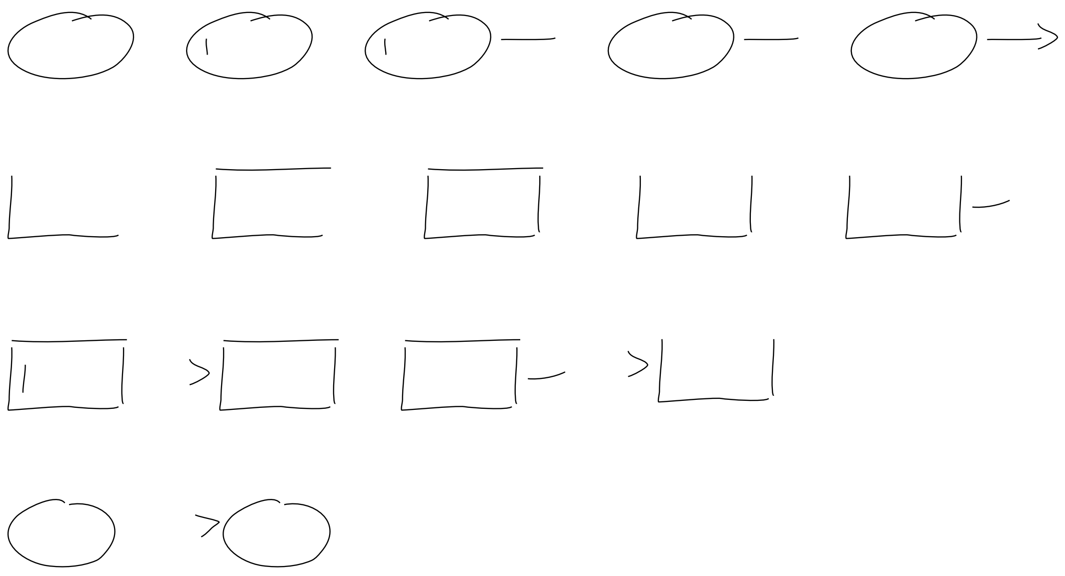


Strokes are classified into two classes: text and shapes. Strokes classified as text are coloured in red.



Text strokes were removed. Notice that some text strokes were misclassified and thus remained. It is a better situation than the opposite misclassification of shape strokes.

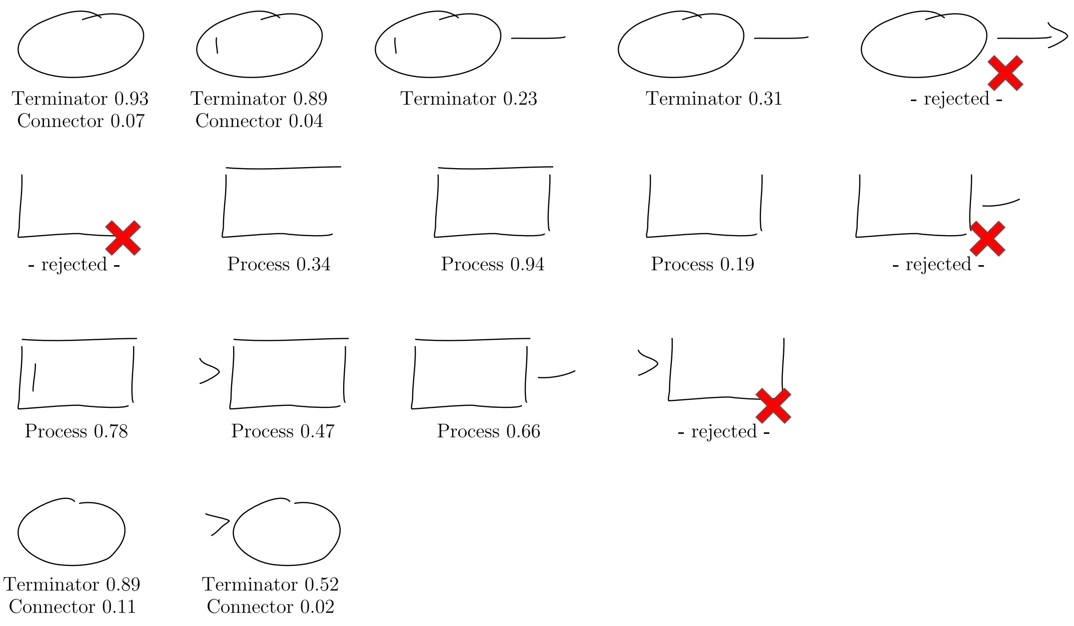
3. **Symbol over-segmentation**



Strokes are grouped/clustered together to form (possibly overlapping) subsets of strokes. Depending on the used technique, over-segmentation can generate a large number of subsets. Only a small portion of these subsets represent a symbol. The illustration shows only subsets very similar to symbols.

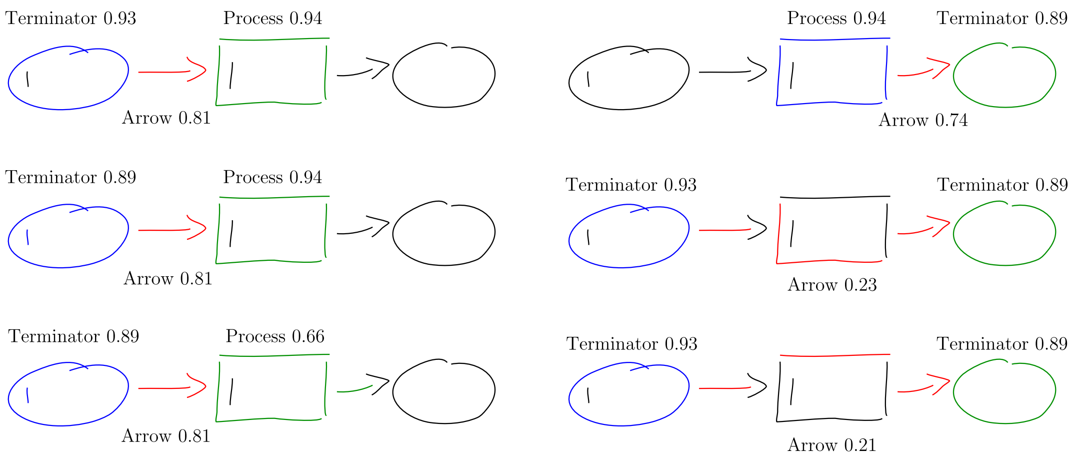
4. **Symbol classification**

### A) Uniform symbols



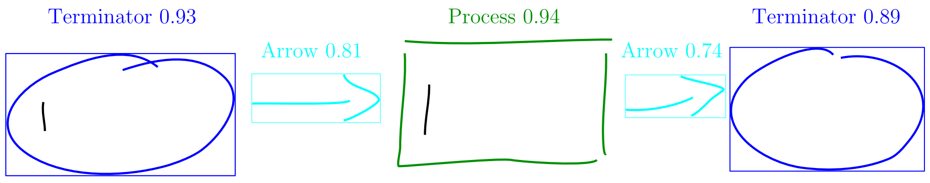
Generated subsets of strokes are classified by a symbol classifier with a rejection ability. Top two results are taken to create symbol candidates for further consideration. Each symbol candidate is shown with a score assigned by the classifier. If both of the two top results are valid symbol classes, they are both displayed. If none of them is a valid symbol, the candidate is rejected.

### B) Arrows



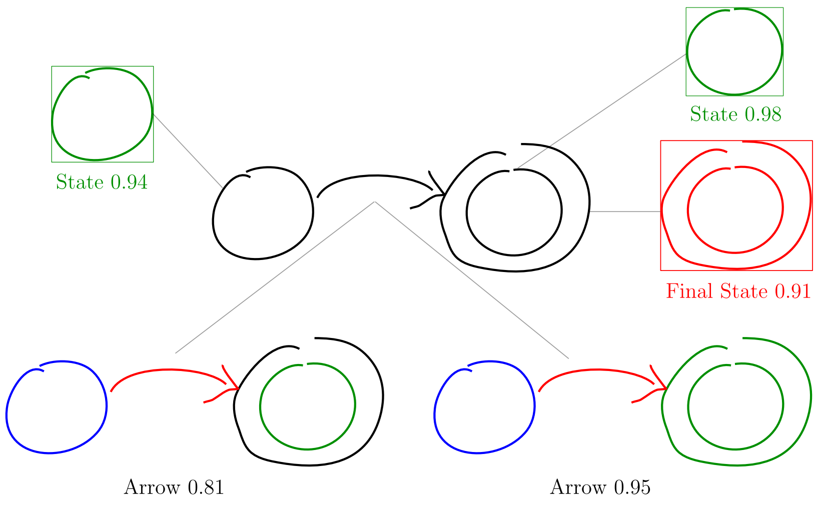
All pairs of symbol candidates are considered and the arrow detector tries to find an arrow connecting these two symbols. Detected arrows are candidates and can be in conflict because even the uniform symbols might be in conflict. This illustration shows only few interesting examples of symbol pairs and detected arrows candidates. The red arrow always leads from the blue symbol to the green one.

5. **Structural analysis**



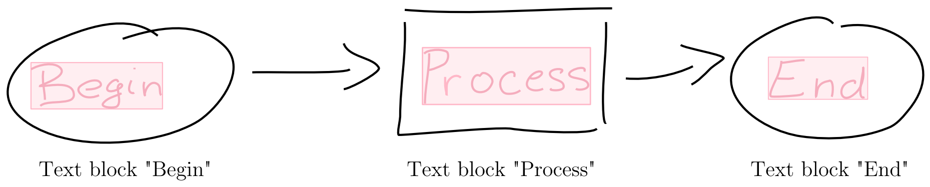
Structural analysis chooses the best combination of symbol candidates forming a valid diagram. There might remain unused strokes (ideally, misclassified text strokes).

Structural analysis exploits a broader context to select a globally optimal subset of symbol candidates. A bad candidate might have a higher score than a good candidate which is in direct conflict with it. However, the good candidate usually fits better into the whole structure of the diagram. Consider the following example commonly occurring in the finite automata domain.



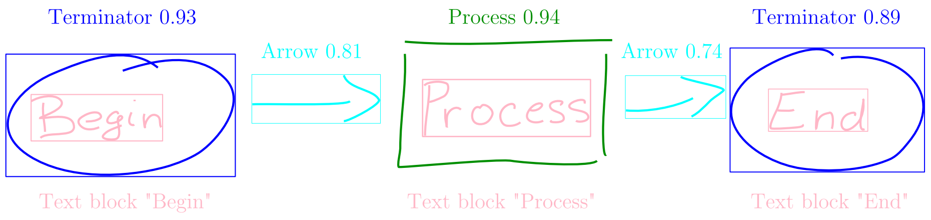
The inner circle of the final state is classified as a state. Its shape is more precise and thus it has higher score than the final state. However, the arrow is better connected (smaller distance between connection point of the arrow and the symbol) to the outer circle than the inner circle and thus the candidate for the arrow linked to the final state has much higher score. Therefore, the global score of the solution is higher when using the symbol of final state although it has lower score than the state.

6. **Text blocks detection and recognition**



Removed text and all unused strokes are divided into text blocks and recognized using a text recognizer.

7. **Output**



All symbols are recognized.

## 3.3. Preprocessing

Our preprocessing phase targets only filtering of insignificant points within strokes. This is necessary, since some of the used algorithms are not invariant to the distance between neighbouring points. On the other hand, we do not perform any global scaling of the input. The system is based on a data adaptive *distance threshold*, which will be explained further.

To filter out insignificant points, the distance between neighbours is measured. When the user draws very slowly, this distance decreases. There might even be duplicated points in the extreme case. A chosen criterion is $\frac{3}{4}$ of the median of distances between consecutive points. A point is removed when the distance to the following point is bellow this limit. The exception is every *corner* point (see details in Section 3.6.2), which is always preserved. The described filtering removes roughly 40 % of points and makes thus the whole recognition process faster.

To perform a stroke segmentation or stroke splitting is another common preprocessing step in many sketch recognition engines. Every stroke is divided into smaller segments at splitting points which are typically defined as corner points. It is required when the user draws multiple symbols by one stroke. However, we do not take this as an option in the case of our domains. The structure of flowcharts and finite automata makes it really unnatural for a user to do that. This decision is well justified empirically. During analysis of the datasets we did not find a single stroke being part of two different symbols. Therefore, we do not perform stroke segmentation generally on all strokes. Nevertheless, we need to perform it just locally during recognition of arrows. If the head and body of an arrow are drawn by one stroke, they must be split. More details follow in Section 3.6.2.

The already mentioned adaptive distance threshold expresses whether two points or strokes are mutually close. It depends on the used input device and a handwriting style of the user. Therefore, it must be computed based on data prior to recognition of each diagram. We denote it as *distThresh*. The threshold plays an important role in various stages of the pipeline. A description of the procedure determining a value of this parameter follows in Section 3.5.1. It describes strokes grouping where the distance threshold is used and thus we can measure what values give us satisfactory results.

## 3.4. Text/Non-text Separation

Separating text from other strokes is motivated by the observation that the text bears almost no information about the diagram structure. Ideally, all text strokes are removed and the diagram without text is recognized. This divide and conquer strategy reduces computational complexity significantly since the number of strokes is much lower. Text strokes are replaced after recognition. The diagram structure helps forming text blocks

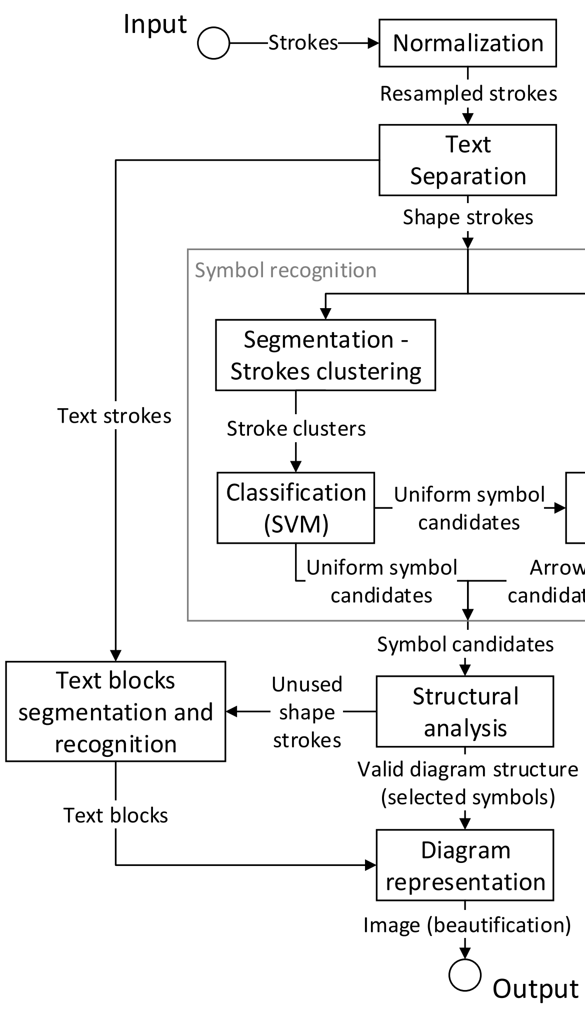


**Figure 3.3.** The proposed recognition pipeline.

and finding symbols, to which the blocks are assigned. Detection and recognition of the text blocks is described later in Section 3.8.

The text/non-text separation algorithm classifies single strokes into two classes – *text* and *shapes*. Although there exist quite precise algorithms for such separation [Delaye and Liu, 2014; Indermühle et al., 2012; Otte et al., 2012; Van Phan and Nakagawa, 2014], they are not able to separate all text strokes. Their accuracy achieved on the benchmark IAMonDo database[1] is between 97 % and 98 %. Moreover, these classifiers tend to have a higher error in class *shapes* due to using unbalanced training datasets. Our goal is the opposite one. We attempt achieving the minimal possible error rate for class *shapes* while a slightly higher error rate in class *text* is acceptable. The justification is that removing a shape stroke can easily cause a symbol not to be recognized, because it becomes incomplete. Some remaining text strokes are not a problem as symbol classifiers are robust enough to deal with noise.

We bias the classifier result and thus only strokes where the classifier is almost certain are marked as text. We implemented two classifiers performing best on the IAMonDo database [Otte et al., 2012; Van Phan and Nakagawa, 2014] and tested them on flowcharts and finite automata. The best performing was the classifier proposed

[1] http://www.iam.unibe.ch/fki/databases/iam-online-document-database

by Phan and Nakagawa [Van Phan and Nakagawa, 2014], which uses unary features to classify individual strokes into two classes: text and non-text. It also considers relationships between adjacent strokes in the writing order and uses binary features to classify transitions between strokes into three classes: text–text, text–non-text, non-text–non-text. Since it can be seen as a sequence labelling task, BLSTM RNN classifiers are used to capture the global context. The probabilistic outputs from the two BLSTM neural networks are combined together to obtain the final labeling probability. It achieves precisions 98.62 % (98.75 % in the *shapes* class and 98.53 % in the *text* class) for FC_A, 99.53 % (98.94 % in *shapes*, 99.74 % in *text*) for FC_B, and 98.84 % (99.85 % in *shapes*, 97.83 % in *text*) for FA in the unbiased case. We were able to reach a biased result in *shapes*/*text* class of 99.68 %/95.20 %, 99.41 %/98.75 %, and 100.00 %/93.31 % for FC_A, FC_B, and FA, respectively. See Table 3.1 for overview of the achieved results. Selection of the bias is a trade-off between the accuracy in both classes. Results for various bias values are shown in Figure 3.4. We chose 0.99 for FC_A, 0.8 for FC_B, and 0.7 for FA. Note that Van Phan and Nakagawa [Van Phan and Nakagawa, 2014] suggest to use sum rule when combining the probability outputs of individual classifiers. Therefore, the final labelling confidence is from interval $[0, 4)$, which explains the high values of the bias.

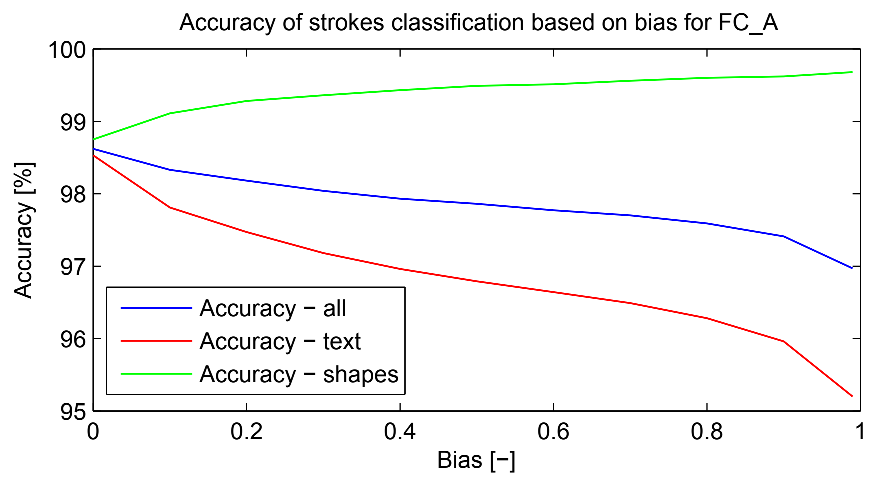
| | Unbiased | | | Biased | | | Otte et al. | | |
|---|---|---|---|---|---|---|---|---|---|
| | FC_A | FC_B | FA | FC_A | FC_B | FA | FC_A | FC_B | FA |
| Shapes | 98.75 | 98.94 | 99.85 | **99.68** | **99.41** | **100.00** | 94.22 | 94.39 | 95.21 |
| Text | 98.53 | 99.74 | 97.83 | 95.20 | 98.75 | 93.31 | 98.11 | 99.39 | 97.37 |
| All | 98.62 | 99.53 | 98.84 | 96.97 | 98.93 | 96.93 | 96.60 | 97.89 | 96.30 |

**Table 3.1.** Accuracy of stroke classification into text/non-text classes. We compare chosen classifier by Van Phan and Nakagawa in unbiased and biased form with unbiased classifier by Otte et al.

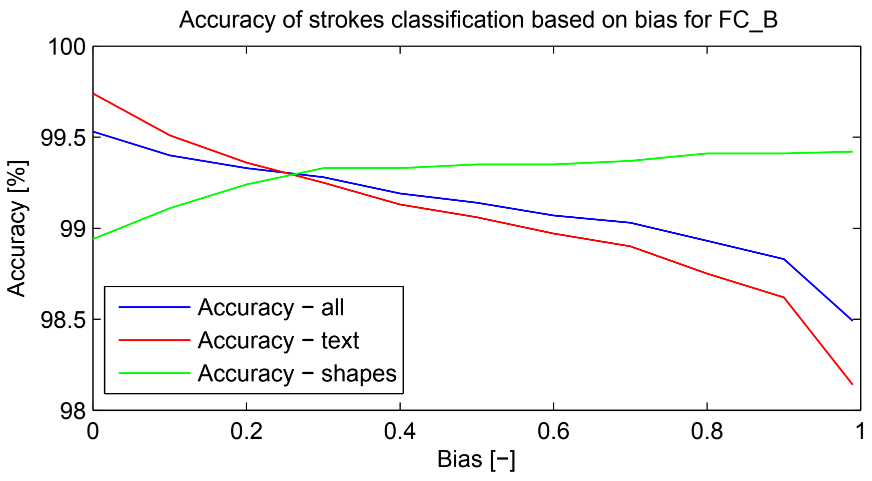
## 3.5. Symbol Segmentation

Segmentation is a process dividing strokes into subsets, each forming a particular symbol. Ideally, the subsets should be disjoint and cover all the strokes. However, it cannot reasonably be done without knowledge of the entire structure. It is unwise to make hard decisions at this early step, and so starting with *over-segmentation* is better. It supplies a larger number of subsets, which may share some strokes. The final decision on which subsets fit the structure of the input diagram best is left for the structural analysis performed later. Our system needs to segment uniform symbols only: arrows are detected after the initial segmentation using the knowledge of recognized uniform symbols. The text is recognized even later when the entire structure is known.

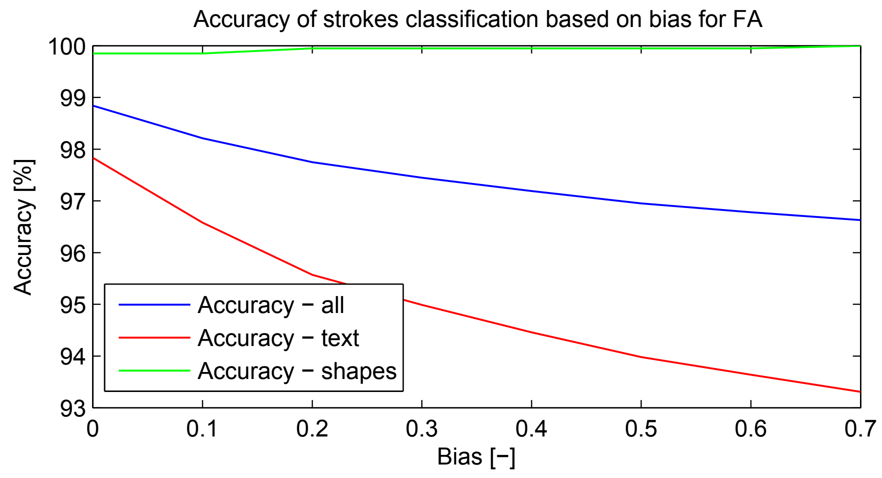
The most common approach to over-segmentation works under the assumption that symbols are formed of spatially and temporally close strokes. Stroke grouping is performed iteratively. Within the first iteration, every single stroke forms a subset of size 1. Next, subsets of size $k$ are created by adding a single spatially and temporally close stroke to subsets of size $k - 1$. The maximal size of the subsets is given by the domain and user's drawing conventions. The spatial proximity is determined simply by Euclidean distance between the two closest points. Temporal proximity is hinted at by stroke indices in the drawing sequence. The approach has been used in our earlier



**(a)** FC_A



**(b)** FC_B



**(c)** FA

**Figure 3.4.** Accuracy of stroke classification in relation to bias. Selection of the best bias is a trade-off between accuracy in *text* and *shapes* classes.

work [Bresler et al., 2014] as well as by others [Feng et al., 2009; Ouyang and Davis, 2011; Álvaro et al., 2014].

Delaye and Lee [2015] showed that symbols may be segmented using Single-Linkage Agglomerative Clustering (SLAC) using a properly trained distance function defined as a weighted sum of several simple features. In addition to Euclidean distance between strokes, the features express the difference between geometric and temporal characteristics of two strokes. The distance is defined as:

$$d(s, t; \mathbf{w}) = \sum_{i=1}^{k} w_i \, d_i(s, t), \tag{3.1}$$

where $s$ and $t$ are two given strokes and $w_i$ is the weight for the feature $d_i$ that needs to be learned. SLAC is a hierarchical bottom-up clustering technique, in which larger clusters are created by iteratively merging the two closest clusters based on the distance. The usage of a Single-Linkage clustering approach implies that the distance between two clusters is given by the distance between their two closest elements. This permits an efficient real-time implementation of complexity $\mathcal{O}(n^2)$, where $n$ is the number of strokes. The clustering gives the final segmentation, which reaches typically lower recall rates. We improved it by performing the over-segmentation by successive clusterings with varying thresholds [Bresler et al., 2015b]. In comparison to the grouping-based over-segmentation, this increased the precision (i.e., generated significantly fewer subsets), at the cost of only very slightly decreased recall. We achieved 95.1 % / 16.7 %, 98.4 % / 27.5 %, 99.8 % / 26.5 % recall / precision on FC_A, FC_B, FA databases, respectively. The high recall is the main objective directly affecting the precision of the whole system while the segmentation precision is a secondary objective, affecting mainly the speed of the system. The recall achieved allows overall high precision. The segmentation precision achieved shows that on average we do not generate more than 4-6 times more segments than is the true number of symbols, which is important for fast recognition.

## 3.5.1. Comparison of Naive Stroke Grouping with Trained SLAC

Here we would like to discuss the advantage of using SLAC-based over-segmentation as compared to naive stroke grouping. To do that, we first describe both approaches in more detail and then show the performance differences. The following notations are used in this section:

- $p_{s,a} = (x_{s,a}, y_{s,a}, t_{s,a}, p_{s,a}) \in s$: the $a$-th point on a stroke $s$ represented as a tuple consisting of $x$ and $y$ coordinates, a timestamp, and a pressure value
- $s = p_{s,1}\, p_{s,2} \, \ldots \, p_{s,n_s}$: a stroke with $n_s$ points
- $\|p_{s,a} - p_{t,b}\| = \sqrt{(x_{s,a} - x_{t,b})^2 + (y_{s,a} - y_{t,b})^2}$: Euclidean distance between two points

### Naive Stroke Grouping

As we already said, this approach iteratively creates new, larger groups/subsets of strokes by adding a single stroke to existing smaller groups. New stroke must be spatially and temporarily close to the subset. Three aspects must be considered: 1) the maximal size of a subset, 2) a criterion determining if two strokes or a stroke and a subset of strokes are spatially close, 3) a criterion for temporal connectivity. A discussion of these aspects follows:

1. The maximal number of strokes forming a symbol depends on the selected domain and the user's drawing style. A greater bound increases the number of subsets significantly. On the other hand, a too low bound might lead to misses of some symbols. Based on the empirical analysis of the datasets, whose result is summarized in the histogram in Figure 3.5, we set the bound to 5 for flowcharts and 3 for finite automata. Note that symbols consisting of a higher number of strokes are usually retraced and some strokes are not necessary for correct recognition (more in Section 4.1). Examples of symbols consisting of more strokes than expected are shown in Figure 3.6.

2. For finite automata, a distance between two strokes is defined as the Euclidean distance between their two closest points. Formally, the distance between stroke $s$ and $t$ in the finite automata domain is defined as follows:

$$d_{FA}(s,t) = \min_{a,b} \|p_{s,a} - p_{t,b}\| \tag{3.2}$$

   For flowcharts, it is enough to consider the distance between end-points, because symbols in flowcharts are drawn in a way when individual strokes are always linked together by their end-points. Formally, the distance between stroke $s$ and $t$ in the flowchart domain is defined as follows:

$$d_{FC}(s,t) = \min_{a,b} \|p_{s,a} - p_{t,b}\|; a \in \{1, n_s\}, b \in \{1, n_t\} \tag{3.3}$$

   A distance between a stroke and a subset of strokes is the distance between the stroke and a closest stroke of the subset. Therefore, it is enough if there is one spatially close stroke in the subset. In a diagram, we say that two strokes or a stroke and a subset of strokes are close if the distance between them is smaller than a threshold $distThresh = \alpha \cdot D_{\text{med}}$, where $D_{\text{med}}$ is a median of bounding boxes diagonal lengths of all single strokes present in the diagram. The constant $\alpha$ has been empirically chosen as 0.35 for both studied domains. The choice is based on the dependencies plotted in Figure 3.7. Such approach is necessary, because different drawing styles and used devices cause different scales. The threshold *distThresh* is applied in other steps of the pipeline as the strokes grouping is not the only one process where we need to determine the spatial proximity.

3. Each subset is required to be formed of at most two different groups of consecutively drawn strokes. This criterion is natural. It may happen that the user draws just half of a symbol, writes some different text, and finishes the symbol after that. However, there are no symbols drawn with two or more interruptions in any of the FC_A, FC_B, and FA databases.

We used all samples in the databases to establish values of the presented parameters. Note that, within each sample, strokes forming texts and arrows were excluded since strokes grouping is not intended to find text or arrows.

### SLAC with Trained Distance Function

The disadvantage of the stroke grouping is the fact that the method considers too many combinations of strokes, which are not important, because they can never form a symbol. This inefficiency led us into experimentation with other possibilities.

Single-linkage clustering based on weighted combination of several features with trainable parameters proposed by Delaye and Lee [2015] reaches a very high precision of

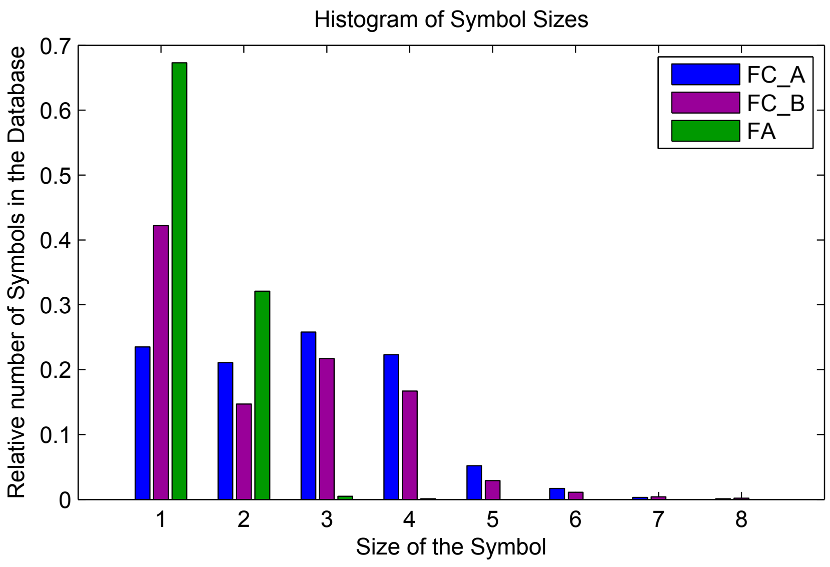
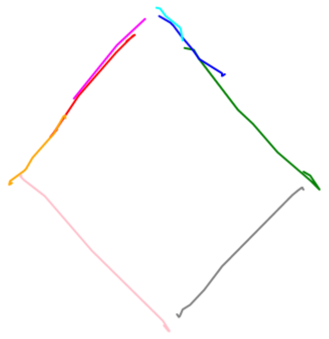


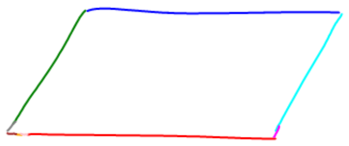
**Figure 3.5.** Histogram of symbol sizes helps to determine the maximal size of strokes subsets. The size is understood as the number strokes forming the symbol.

object segmentation. Its advantage is that it uses more features combined together and can express more complex relations between strokes than just the Euclidean distance. Another advantage is that single-linkage is a fast clustering algorithm. The time complexity is quadratic in the number of strokes. It is only needed to compute the distance between individual strokes once. The distance between two clusters is given by the distance of their two closest strokes. We reimplemented the method and trained the feature weights and the threshold as it is described in the work by Delaye and Lee. We achieved a bit worse precision (cca. 3 % less) on both, FC and FA, databases. We believe it is caused by slight differences in the input normalization. However, the method is powerful and the result is satisfactory for our purposes.

We perform the clustering with the trained parameters and several values of the threshold to get an over-segmentation to increase the recall. We obtained several values of the threshold by multiplication of the original threshold $h$ by a changing coefficient $c_i$: $h_i = h \cdot c_i$. We used increasing values of $c_i$ from the interval $[c_{min}, c_{max}]$ with the step 0.1, where the bounds $c_{min}$ and $c_{max}$ must be found in a validation procedure. Only the uniform symbols are our objects of interest, because our recognition system deals with text and arrows separately. We used the validation dataset of the FA database and training dataset of the FC database to find the bounds of the coefficient. We tried all combinations of $c_{min}$ from the range $[0.1, 1.0]$ and $c_{max}$ from the range $[1.0, 2.0]$. The best combination of the bounds is that one, which gives the highest recall. When more combinations give the same recall, the combination giving higher precision is taken. We found out that the best values are $c_{min} = 0.5$ and $c_{max} = 1.2$ for both domains.

A comparison of results achieved by both over-segmentation methods is shown in Table 3.2. Notice that the text separation step precedes the over-segmentation step and thus the most of the text strokes is removed. The text separator achieves the precision in *shapes/text* class of 99.68 %/95.20 %, 99.41 %/98.75 %, and 100.00 %/93.31 % for FC_A, FC_B, and FA, respectively. Since the over-segmentation is used to find uniform symbols only, we do not consider text blocks or arrows as relevant objects. Therefore


**(a)** Decision symbol from file writer1_1 of the FC_A database. Magenta and cyan strokes are redundant.


**(b)** Process symbol from file writer002_fc_010 of the FC_B database.

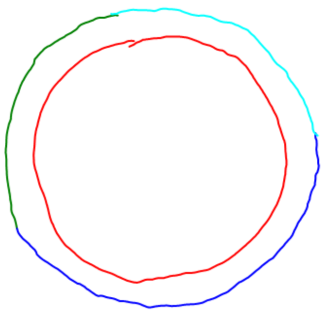

**(c)** State symbol form file writer002_fa_002 of the FA database.

**Figure 3.6.** Examples of symbols consisting of more strokes than expected from individual databases. Individual strokes are in different colours for better visualization.

there are 921 / 1 337 / 488 relevant objects in the test dataset of the FC_A / FC_B / FA databases. Notice that the clustering method achieved even higher recall than the naive grouping in the case of FA. Obviously, a few symbols in the test dataset violated one of the assumptions used in the process of strokes grouping. Specifically, they comprise of more strokes than allowed. The advantage of the clustering approach is that we do not need such assumption at all.

| Database – Method | Retrieved | Relevant | Matched | Recall | Precision | F-measure |
|---|---|---|---|---|---|---|
| FC_A – grouping | 19 714 | 921 | 878 | 95.33 % | 4.45 % | 0.085 |
| FC_A – clustering | 5 245 | 921 | 876 | 95.11 % | 16.70 % | 0.284 |
| FC_B – grouping | 18 618 | 1 337 | 1 315 | 98.35 % | 7.06 % | 0.132 |
| FC_B – clustering | 5 095 | 1 337 | 1 315 | 98.35 % | 25.81 % | 0.409 |
| FA – grouping | 6 095 | 488 | 485 | 99.39 % | 7.96 % | 0.147 |
| FA – clustering | 1 823 | 488 | 487 | 99.80 % | 26.71 % | 0.419 |

**Table 3.2.** The results of strokes grouping and clustering on the test datasets of the FC and the FA databases.

## 3.6. Symbol Recognition

Symbol recognition aims at classifying subsets of strokes (clusters) produced by segmentation. Each cluster is either assigned a symbol class or is rejected. We treat arrows in a special way since their form and shape varies, which is a difficulty for traditional classifiers. There are two stages. The first stage, uniform symbols are recognized using a standard classifier. The second stage, arrows are detected as connectors between symbol candidates found earlier. Both recognizers/detectors provide an ordered list of symbol candidates. The structural analysis selects the best symbols from these lists.

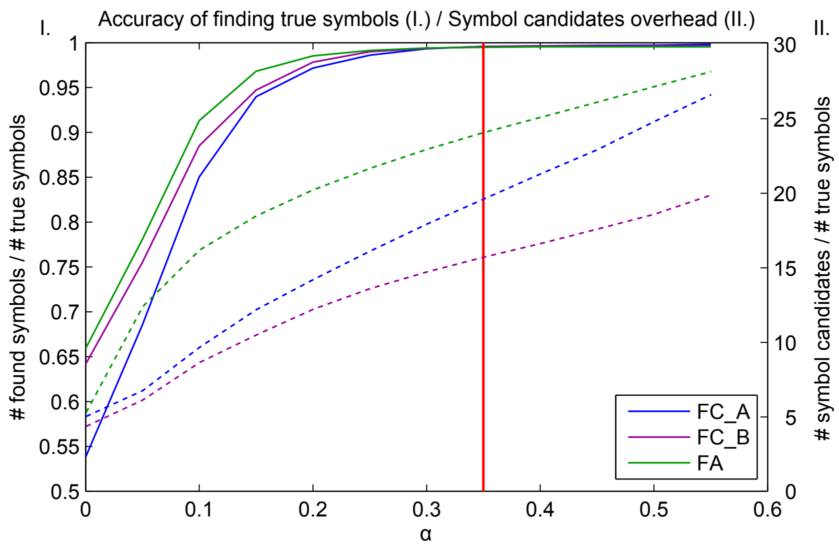


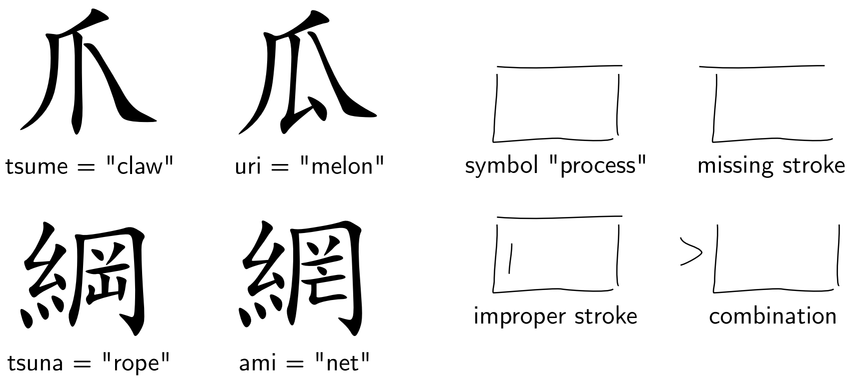
**Figure 3.7.** The result of experiments performed over FC_A, FC_B, and FA databases to obtain an optimal value of $\alpha$ coefficient. The graph shows the accuracy which improves as $\alpha$ grows (I., solid) and the growth of the number of strokes subsets (II., dashed) at the same time. The choice of $\alpha$ is thus a tradeoff. The chosen $\alpha = 0.35$ is indicated by the red line.

### 3.6.1. Uniform Symbol Classifier

The classifier has to fulfil three requirements: 1) It has to be fast since many stroke clusters need to be processed; 2) The rejection ability is mandatory as many stroke clusters do not represent anything meaningful; 3) Each classification produces a score (e.g., a posterior probability) to compare the candidates quality.

We used an off-the-shelf solution and combined the trajectory based normalization and direction features proposed by Liu and Zhou [2006] as a descriptor, which served as the input to the multiclass SVM classifier. The descriptor is based on hybrid features capturing dynamic information as well as the visual appearance of symbols. It consists of 512 features. It was primarily designed for recognition of Japanese characters and thus works well for a high number of visually similar symbol classes since individual Japanese symbols often differ in small details only. This property is desirable to enable rejection of incomplete symbols produced by the over-segmentation. The analogy with Japanese characters is illustrated in Fig. 3.8 and the achieved results confirm suitability of the selected descriptor. Although there are different descriptors available [Delaye and Anquetil, 2013], we did not experiment with them because we achieved satisfactory results with the chosen solution. We trained the classifier with negative examples to obtain the rejection ability. The dataset of symbols for training was obtained by applying the stroke clustering introduced in Section 3.5. If the cluster of strokes is annotated as a uniform symbol in the database, it is labelled by that symbol. Otherwise it is labelled as *no_match*, which denotes a negative example. Arrows as well as incomplete parts of symbols are labelled as negative examples. Because the FC_A database does not contain a validation set, we used a 5-fold cross-validation. Therefore, we merged the training and validation datasets in case of FC_B and FA databases to have the same conditions.

The number of negative examples is much higher than the number of uniform symbols. Moreover, they are very inhomogeneous. It is thus necessary to cluster them into subclasses. We employed $k$-means based on the descriptor to create $m$ *no_match*



**(a)** individual characters **(b)** *process* vs. *no_match*

**Figure 3.8.** Example of small differences between individual symbol classes in the case of Japanese characters (a) and uniform symbols in diagrams (b).

subclasses, where $m$ is domain dependent ($m = 30$ for flowcharts, $m = 20$ for finite automata). The appropriate values of $m$ were estimated from the data while pursuing clusters with desirable properties such as homogeneity and separability. The larger number of symbol classes in the flowchart domain naturally results in a greater $m$. This brings a need for a modified loss function, which gives the zero penalty when a negative example is classified into a different *no_match* subclass. Additionally, a greater penalty is required for misclassification of a uniform symbol as a negative example than in the opposite case. The ratio between these two penalties depends also on the ratio between the number of uniform symbols and negative examples. A properly chosen loss function can overcome the problem with an unbalanced database [Bresler et al., 2014]. However, our current implementation uses artificially synthesized samples to balance the database. The samples were synthesized using the approach of Martín-Albo et al. [2014]. It is based on Kinematic Theory and the distortion of the Sigma-Lognormal parameters in order to generate human-like synthetic samples. See Figure 3.9 for a better idea of how these samples look like. We generated up to 20 artificial samples from each uniform symbol taken from the training dataset. From all the synthesized samples of one class, we randomly chose a subset to get the desired number of symbols for training. The counts of original samples taken from the training and test datasets are shown in Table 3.3. We decided to supply each class of the uniform symbols with artificial samples to obtain 2000/4000 samples altogether in flowchart/ finite automata database. These numbers of samples ensure balanced datasets. This approach not only helps to balance the dataset, it also supplies additional information on handwriting and makes the classifier more robust. Therefore, we empirically set the smaller penalty to 1 and the bigger penalty to 2 just to increase recall at the cost of very small precision decrease. In the finite automata case, non-initial arrows and stroke subsets of final states have exactly the same appearance as initial arrows and states, respectively. To benefit from this knowledge of the domain, we excluded these two from negative examples, which increases the recall of the symbol classifier. Specifically, the recall increased from 96.23 % to 98.98 % reported later in this section. Unfortunately, the FC_A database does not contain any time information, which is crucial for the synthesis, thus artificial samples cannot be obtained for this database.

Without negative examples, the proposed classifier achieved the precision of 98.9 %, 97.5 %, and 100.0 % for FC_A, FC_B, and FA, respectively. If rejection is incorporated

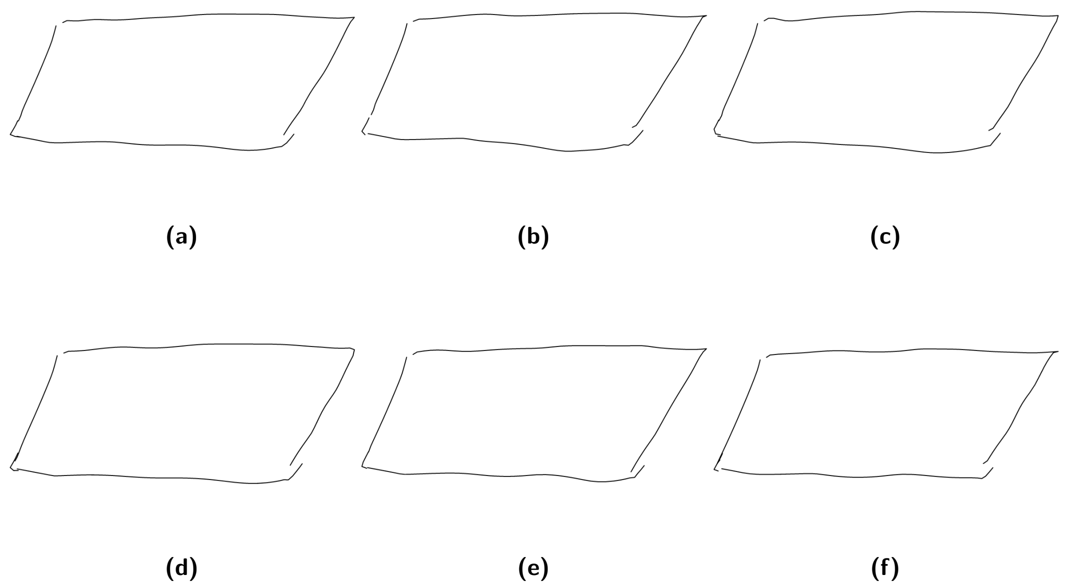


**Figure 3.9.** Example of the original data symbol from flowchart domain (a) and five synthetically generated samples from this symbol (b–f).

| | FC_A train. | FC_A test | FC_B train. | FC_B test | FA train. | FA test |
|---|---|---|---|---|---|---|
| Connection | 144 | 94 | 270 | 112 | - | - |
| Data | 337 | 214 | 865 | 356 | - | - |
| Decision | 247 | 158 | 544 | 224 | - | - |
| Process | 478 | 304 | 920 | 380 | - | - |
| Terminator | 241 | 151 | 643 | 265 | - | - |
| Initial arrow | - | - | - | - | 197 | 75 |
| Final state | - | - | - | - | 342 | 127 |
| State | - | - | - | - | 720 | 286 |
| No-match | 42 319 | 31 600 | 84 225 | 28 357 | 12 326 | 4 672 |

**Table 3.3.** The number of original samples in training and test datasets.

through negative examples, we keep the two topmost results of classification for each stroke cluster to make the symbol candidate detection more robust. It might happen that both results are *no_match*. Then, the corresponding cluster is rejected. This yields the recall/precision of 94.13 %/43.13 %, 96.63 %/45.33 %, and 98.98 %/37.15 % for FC_A, FC_B, and FA, respectively. The average recognition time per sample is 0.7 ms (tested on a tablet PC Lenovo X230 – Intel Core i5 2.6 GHz, 8 GB RAM).

### 3.6.2. Arrow Detector

As stated earlier, it is difficult to perform recognition of an arrow based on its appearance, even if several arrow subclasses are considered. This was our initial approach [Bresler et al., 2013], which did not lead to satisfactory results. Some generic recognizers for arrows are also known [Kara and Stahovich, 2004; Stoffel et al., 2009]. However, they expect a fixed form of arrows – the number of strokes as well as the range of angles between them are restricted. This conforms to some domain specific solutions, but is impractical in general. The requirements on the user's drawing style are unnatural, a new model for each arrow form has to be defined, etc.

Therefore we created a new specialized arrow detector [Bresler et al., 2015a]. It exploits the special property of arrows (they connect two symbols) and detects candidates after uniform symbol candidates have been recognized. We consider each pair of detected uniform symbol candidates and try to find an arrow as an arbitrarily shaped connector linking them. Each arrow consists of a shaft and a head, hence the arrow candidates detector works in two sequential steps:

1. Find the shaft of an arrow connecting two given symbols. The shaft is a sequence of strokes leading from the vicinity of the first symbol to the vicinity of the second symbol and is undirected.

2. Find the head located around one of the end-points of the shaft. It defines the arrow orientation.

The detection of the arrow shaft can be done iteratively by simply adding strokes to a sequence such that the first stroke starts in a vicinity of the first symbol and the last stroke ends in a vicinity of the second symbol. A new stroke is added to the sequence only if the distance between the end-point of the last stroke and the end-point of the new stroke is smaller than a threshold. The algorithm must consider all possible combinations of strokes creating a valid connection between the given two symbols. The search space can be reasonably reduced by setting a maximal number of strokes in the sequence. This number depends on the domain and the fact, how many strokes the users use to draw arrow shafts. Typically, it is four and two for flowcharts and finite automata, respectively. We can immediately remove some shafts, which are in a conflict with another shafts, and keep those with the smallest sum of the following distances: a) distance between the first symbol and the first stroke of the shaft, b) distance between the second symbol and the last stroke of the shaft, c) distances between individual strokes of the shaft.

Since we do not know the orientation of the arrow yet and the shaft is undirected, we have to consider both end-points of the shaft and try to find two heads (one in the vicinity of each end-point). Ideally we will be able to find just one head. In practice, it can happen that we find two heads and we have to decide which one is better. The detection of an arrow head is not a trivial task, because there might be a lot of interfering strokes around the end-points of the shaft: heads of another arrows or some remaining text. The decision which strokes represent the true arrow head we are looking for and which are not, is a task, where the stroke positioning might be used beneficially. First, we define a reference stroke (a sub-stroke of the shaft) and a reference point (end-point of the shaft), which are used to express a relative position of query strokes (details follow later in this section). Second, this information about relative position is given to a classifier making the decision. The query strokes are all strokes in a vicinity of a given end-point of the shaft, which are not a part of the shaft itself nor the two given symbols. We make a classification into two classes: *head* and *not-head*. The explanation for the evaluation of the relative position of strokes and the classification is given later in this section. Let us just note that the classifier returns a class, into which the query stroke is classified along with a potential[2]. We use this

[2]The potential is given by a neural network which we use as the classifier. Its detailed description will be provided later in this section, in subsection *Evaluation of Relative Position of Strokes*. Note that this potential is not the final score of the arrow.

potential to decide which head is of better quality in the case we find two heads. We just compute the sum of potentials of all strokes in each head and decide for the head with the greatest value. This slightly favours heads consisting of higher number of strokes, which is desirable in the most cases. A pseudocode for the described algorithm that is divided into two procedures and presented as Algorithm 1 and Algorithm 2. The arrow recognition pipeline is depicted in Figure 3.10.

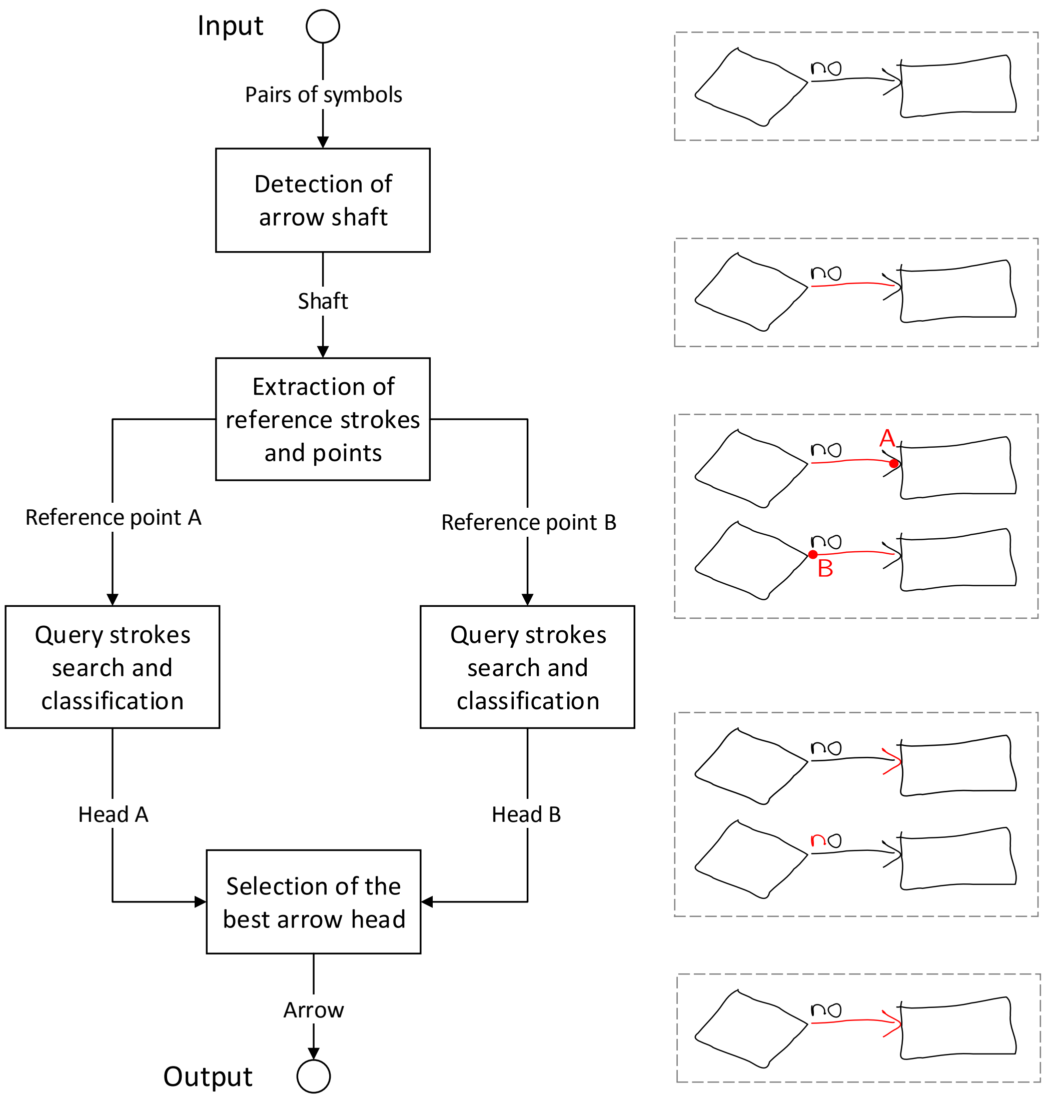

Input
Pairs of symbols
Detection of arrow shaft
Shaft
Extraction of reference strokes and points
Reference point A
Reference point B
Query strokes search and classification
Query strokes search and classification
Head A
Head B
Selection of the best arrow head
Arrow
Output
no
A
B

**Figure 3.10.** Arrow recognition pipeline. The recognition process is illustrated on a simple example of two symbols from flowchart domain. The input is a pair for non-arrow symbols and the output is an arrow connecting the given symbols.

It happens quite often that the user draws a shaft and a head of an arrow by one stroke. Our algorithm would fail in that case. Therefore, we make one additional step before we try to find the arrow head. We split the last stroke of the shaft into smaller sub-strokes in such a way that the head is split from the shaft. Created sub-strokes are divided into two groups. One group is used to finish the shaft again such that it reaches the symbol again. Sub-strokes of the second group are put into the set of query strokes possibly forming the head. Our splitting algorithm is described later in this section. If the shaft and the head are not drawn by one stroke, the algorithm will ideally perform no segmentation and this step can be skipped.

The rest of this section describes extraction of the reference stroke and the reference point, splitting algorithm to split the arrow head form the arrow shaft, and the evaluation of the relative position of the head strokes with respect to the reference point of the arrow shaft.

```
Input: symbolA, symbolB, strokes
Output: List of arrow candidates arrows
shaftCandidates = DetectShaftCandidates(symbolA, symbolB, strokes);
RemoveShaftsInConflict(shaftCandidates);
foreach shaft in shaftCandidates do
    /* Segment strokes of the shaft to split the head in the case it
    was drawn by one stroke together with the shaft. We keep only a
    minimal number of strokes coming from the first given symbol to a
    vicinity of the second given symbol.                            */
    [shaftA, residualA] = Segment(shaft, symbolA, symbolB);
    [shaftB, residualB] = Segment(shaft, symbolB, symbolA);
    endpointA = shaftA.lastPoint;
    endpointB = shaftB.firstPoint;
    queryStrksA = StrokesInVicinity(endpointA, strokes \ shaftA.strokes);
    queryStrksB = StrokesInVicinity(endpointB, strokes \ shaftB.strokes);
    /* Classify the query strokes into two classes. Keep only stroke
    classified as arrow's head. Function returns a sum of potentials
    of these strokes given by the classifier as well.               */
    [headA, potentialA] = Classify(queryStrksA ∪ residualA, shaftA, endpointA);
    [headB, potentialB] = Classify(queryStrksB ∪ residualB, shaftB, endpointB);
    if headA == null && headB == null then continue;
    if potentialA > potentialB then arrows.Add(new Arrow(shaft, headA));
    else arrows.Add(new Arrow(shaft, headB));
end
```

**Algorithm 1:** Algorithm searching for arrows connecting two given symbols.

### Reference Stroke and Reference Point

It is necessary to define a reference stroke. Position of all query strokes will be evaluated relatively to it. Naturally, it seems that the arrow shaft should be the reference stroke. However, it is better to use just a sub-stroke of the shaft for this purpose. The reason is that the shaft might be arbitrarily curved or refracted, the whole arrow might be arbitrarily rotated, and we want to normalize the input in such a way that the reference stroke has always more or less the same appearance and the query strokes have always more or less the same relative position. Therefore, we created a sub-stroke beginning at the end-point of the shaft with the shape of a line segment. It is done iteratively by adding points to the newly created stroke until the value of a criterion, which expresses how similar the stroke is to a line, is greater than a threshold. The criterion is a ratio of the distance between the end-points of the stroke and the path length of the stroke (sum of distances between neighbouring points). We set the threshold empirically to 0.95. Another condition is that the distance between end-points of the stroke must be

```
Input: symbolA, symbolB, strokes
Output: List of shaft candidates shafts
openShafts = stack of unfinished shafts;
foreach stroke in strokes do
    if stroke is not in a vicinity of symbolA then continue;
    shaft = new Shaft(stroke);          // create a shaft candidate of size 1
    if Distance(symbolA, stroke.firstPoint) < Distance(symbolA, stroke.lastPoint)
    then
        shaft.firstPoint = stroke.firstPoint;
        shaft.lastPoint = stroke.lastPoint;
    else
        shaft.firstPoint = stroke.lastPoint;
        shaft.lastPoint = stroke.firstPoint;
    end
    openShafts.Push(shaft);
end
while openShafts is not empty do
    shaft = openShafts.Pop();
    if shaft.lastStroke is in a vicinity of symbolB then shafts.Add(shaft);
    else if shaft.size <= maxSize then      // maxSize = 1 for the FA domain
        foreach stroke in strokes do
            if shaft.Contains(stroke) then continue;
            if Distance(stroke.firstPoint, shaft.lastPoint) < distThresh then
                newShaft = clone of shaft;
                newShaft.Append(stroke);
                newShaft.lastPoint = stroke.lastPoint;
                openShafts.Push(newShaft);
            else if Distance(stroke.lastPoint, shaft.lastPoint) < distThresh then
                newShaft = clone of shaft;
                newShaft.Append(stroke);
                newShaft.lastPoint = stroke.firstPoint;
                openShafts.Push(newShaft);
            end
        end
    end
end
```

**Algorithm 2:** Algorithm searching for arrow shaft candidates (sequences of strokes) connecting two given symbols – DetectShaftCandidates.

bigger than a threshold derived empirically from the average length of strokes, because the possible presence of so called hooks at ends of strokes would cause a small value of the criterion for short strokes. Figure 3.11 illustrates how the reference stroke is determined as a sub-stroke of the shaft. Then we rotate the reference stroke and all query strokes by such an angle that the vector given by the end-points of the reference stroke will be pointing in the direction of the $x$-axis. In another words, it will ensure that the true arrow heads should point from the left to the right. We have to define a reference point for the purposes of our method evaluating the relative position of strokes described further in this section. Obviously, it is the end-point of the shaft.

Because we still do not know the orientation of the arrow, we have to consider both options: the arrow is heading to the first symbol or the second symbol. Therefore, we define two reference points, end-points of the shaft. A reference (sub)stroke is associated to each of these two points then. Figure 3.11 shows the whole process of the reference stroke extraction and rotation.

## Stroke Splitting

Stroke splitting (often referenced as segmentation) is very active field of research, because it is the frequently used preprocessing step. Therefore, there exist various papers dealing with this task. The segmentation is performed by defining a set of splitting points. The substantial information is the curvature and the speed defined at each point, and geometric properties of stroke segments. The common approach is to find tentative splitting points with high curvature and low speed. The best subset of these points is selected according to the error function fitting points of each segment into selected primitives. The most common primitives are line segments and arcs [El Meseery et al., 2009; Wolin et al., 2009]. It is also possible to use machine learning to train a classifier detecting the splitting points [Feng and Viard-Gaudin, 2008; Herold and Stahovich, 2011].

The presented algorithms are sophisticated and allow to find segments fitting predefined primitives. However, using any of these methods seems to be an overkill for our task. We neither require splitting a stroke at any precisely defined point nor creating segments with particular geometrical properties (line segments or arcs). All we need is splitting the arrow head from its body. It is not important if both, the body and the head, will be further split into several segments. We suggest to use a much simpler algorithm for stroke segmentation. Its description follows.

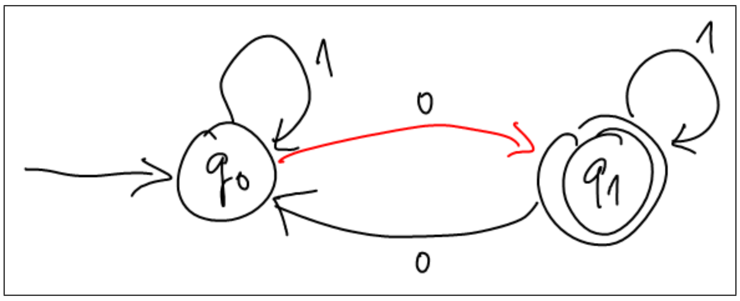
We compute a value $AA$, which we call “accumulated angle”, associated to each point of the stroke $S = \{p_1, p_2, \dots, p_n\}$ according to the following equation:

$$AA_i = \text{mean}(\text{Rank3}\{\text{A}(i,1), \dots, \text{A}(i, \min(i-1, n-i, R))\}), \tag{3.4}$$

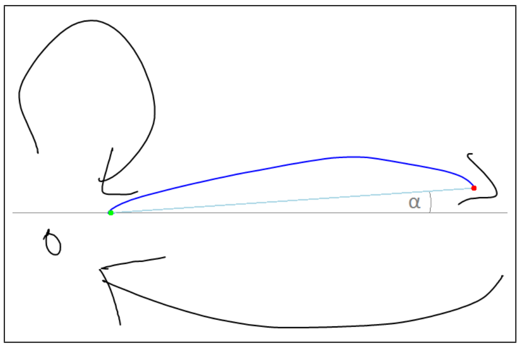
where $i$ is the index of the point in the sequence, Rank3 is an operator choosing up to the three smallest values of a given set, $R$ is the maximal radius, and A is a function computing an angle between two vectors defined by the index of the given reference point and its two neighbouring points chosen by the size of the radius. The function $A$ is defined as follows:

$$\text{A}(i,r) = \arccos \frac{\overrightarrow{p_i p_{i-r}} \cdot \overrightarrow{p_i p_{i+r}}}{\|p_i p_{i-r}\| \cdot \|p_i p_{i+r}\|}. \tag{3.5}$$

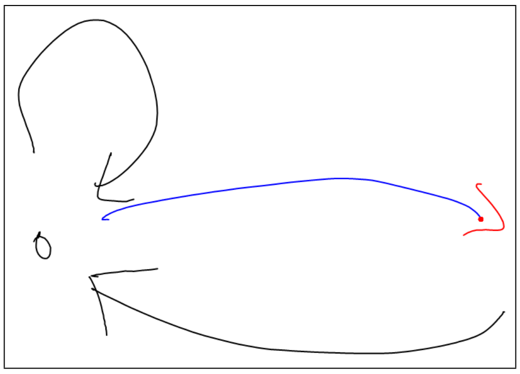
Let us formally define the operator Rank3 in the following way. Let $X = \{X_1, \dots, X_n\}$



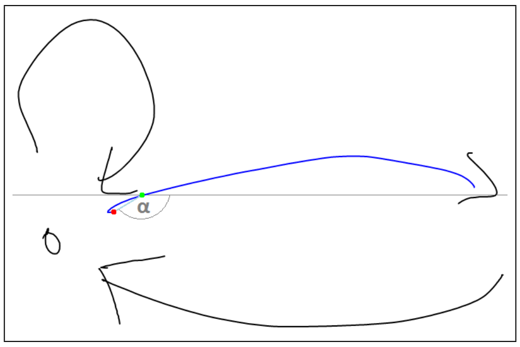
(a)



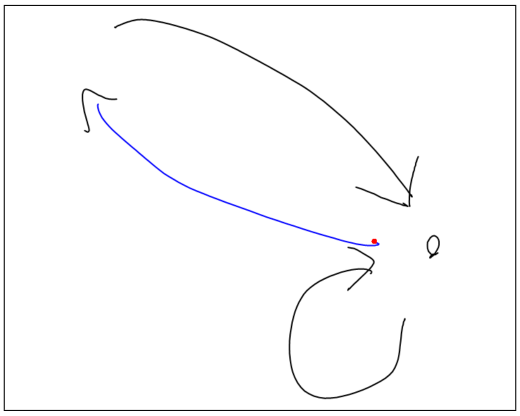
(b)



(c)



(d)



(e)

**Figure 3.11.** Example showing a diagram and the way of choosing the reference point, the reference stroke, and the rotation. Individual pictures illustrate the following: (a) whole diagram with a highlighted (red) arrow to be detected, (b) detected arrow shaft is blue and right end-point is considered to be the reference point, second point is green, the angle $\alpha$ used to rotate query strokes is marked, (c) rotation is done, the reference point is red as well as strokes of the real arrow's head, (d) analogously to (b) with the other end-point considered, (e) analogously to (c) with exception that there is no real head, because the arrow orientation is wrong.

be a set of elements, such that $n \geq 3$. Now we define the following ordering permutation:

$$\sigma : \{1, \dots, n\} \to \{1, \dots, n\} \tag{3.6a}$$
$$X_{\sigma(1)} \leq X_{\sigma(2)} \leq \dots \leq X_{\sigma(n)} \tag{3.6b}$$

Then we can define Rank3 as follows:

$$\text{Rank3}\{X\} = \{X_{\sigma(1)}, X_{\sigma(2)}, X_{\sigma(3)}\}. \tag{3.7}$$

Let us note that $AA_i$ is computed according to Equation (3.4) only for $i \in \{2, \dots, n-1\}$ and $AA_1 = AA_n = 0$. We define the initial set of splitting points by taking points where $AA$ reaches a local minima and the value is smaller than $mCoeff \cdot \text{mean}\{AA_1, \dots, AA_n\}$. If that there are two splitting points too close to each other ($\text{dist}(p_i, p_j) < distThresh$), we remove the one with the smaller $AA$ value. We set $mCoeff = 0.5$ and $distThresh = 200$ empirically. After this removal, the segmentation is finished.

We tested the described algorithm on arrows from the validation datasets of FC_B and FA databases, which were drawn by one stroke. It turned out that the algorithm split the head from the body in 100% of cases. Let us emphasize that parameters *mCoeff* and *distThresh* are tunable. It makes it easy to adjust for demands of a given task.

## Evaluation of Relative Position of Strokes

Let assume we are given a reference stroke (represented by its reference point $R$) and a query stroke $S = \{p_1, p_2, \dots, p_n\}$. To describe the relative position of $S$ with respect to $R$, the relative position of each point $p_i$ is expressed in polar coordinates as angle $\alpha_i = \overrightarrow{Rp_i} \angle \overrightarrow{x}$ and distance $d_i = \|Rp_i\|$, see Figure 3.12. For each pair of a reference and a query stroke, this induces a sample formed of feature vector $\{[\alpha_1, d_1], [\alpha_2, d_2], \dots, [\alpha_n, d_n]\}$ and a label indicating the class of the query stroke. We use Long Short Term Memory (LSTM) Recurrent Neural Network (RNN) as a classifier, because it reaches the best results in many related applications [Graves and Schmidhuber, 2005; Otte et al., 2012; Indermühle et al., 2012]. It is beneficial to normalize inputs when dealing with neural networks. We set

$$\hat{v_k} = \frac{v_k - m_k}{\sigma_k}, \tag{3.8}$$

where $v_k$ is an input value, $\hat{v_k}$ is the normalized value, $m_k$ and $\sigma_k$ are the mean and the standard deviation of all values of the same features from the training database, respectively. This is applied to normalize the distance. The proposed features express relative position of the query stroke with respect to the reference point as well as the shape of the query stroke. It is possible to reconstruct the trajectory of the query stroke from the sequence of features.

## Experiments with the Arrow Detector

To train the proposed arrow head classifier, we extracted a reference point and stroke for each annotated arrow as described in this section earlier. The only difference is that the shaft is known. We created a set of query strokes and rotated these strokes according to the reference stroke. We extracted features with respect to the reference point for each query stroke and assigned a label to it based on the annotation from

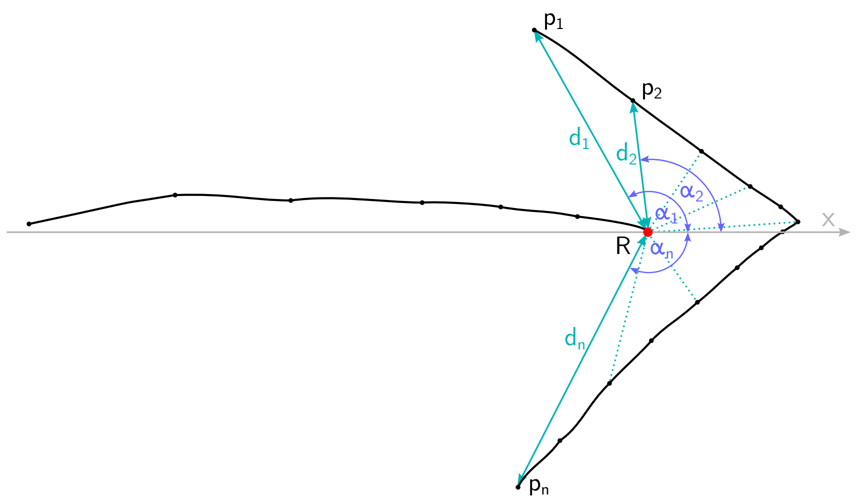


**Figure 3.12.** A pair of reference and query strokes and extracted sequences of features (angles and distances). The reference point R is marked red. Both, the reference and the query strokes, are already rotated. The query stroke is the sequence of points $\{p_1, p_2, \ldots, p_n\}$.

the database. Positive samples are those samples assigned by label *head*, negative are those assigned by label *not_head*. In total, we extracted 1731/1407, 3686/3491, and 1480/1242, positive/negative samples from training datasets of FC_A, FC_B, and FA, respectively.

We used LSTM RNN implemented in the library JANNLab [Otte et al., 2013]. We also experimented with the Bidirectional LSTM and tested different amounts of nodes in the hidden layer to get the best performance [Bresler et al., 2015a]. The optimal compromise between the accuracy and the speed was achieved by LSTM with 8 nodes in the hidden layer. The network was trained in 200 epochs with learning rate 0.001 and momentum 0.9. One classification takes 0.44 ms in average. The achieved precision is 98.1 %, 99.8 %, and 99.6 % for FC_A, FC_B, and FA, respectively.

As the second experiment, we took all annotated uniform symbols and tried to find arrows with the proposed arrow detector to test its precision. The detected arrows were compared with the annotated arrows. Notice that the text strokes were removed by our text/non-text separator introduced in Section 3.4 and all pairs of symbols were considered. Conflicting arrow shafts were removed immediately. However, adding arrow heads may cause another conflicts. The result of the arrow detector is thus a list of arrow candidates, the remaining conflicts are intended to be solved by the structural analysis described later in Section 3.7. We achieved the recall/precision of 92.4 %/63.1 %, 94.7 %/75.0 %, and 95.1 %/44.6 % for FC_A, FC_B, and FA, respectively. Our arrow detector performs 88 stroke classifications in average per diagram when searching for arrow heads while there are 10 arrows in average per diagram.

## 3.7. Structural Analysis

The input to the structural analysis comprises of symbol candidates assigned by score. Candidates for arrows also identify which two symbols they connect. The task is to detect a subset of the candidates forming a valid diagram. The score of the individual candidates itself does not suffice (even a bad candidate might have a high score). Relations between the candidates have to be examined. Each relation is assigned its own score. The score of the entire diagram is calculated as the sum of scores of all selected symbol candidates and relations among them. The highest score solution is sought in an optimization task. Having the candidates selected, the diagram structure can be easily reconstructed, because all the necessary information is carried by arrows.

We define three types of relations between symbol candidates: 1) Conflict – two candidates share one or more strokes or two arrows are connected to the same connection points of uniform symbols (this forbids parallel arrows in flowcharts); 2) Overlap – two uniform symbol candidates have overlapping bounding boxes; 3) Endpoint – each arrow requires existence of both uniform symbols it connects. All possible pairs of candidates are examined to find first two relations. Potential conflicts are detected first, and if none occurs, relations of type 2) are evaluated. Relations of type 3) are explicitly given by the definition of arrows. Relations of type 1) get the score $s_\mathrm{c} = -\infty$. Each relation of type 2) gets the score

$$s_\mathrm{o} = -S_{A \cap B} / \min(S_A, S_B) \,, \tag{3.9}$$

where $A$ and $B$ are bounding boxes of the first and the second symbol. $S_A$, $S_B$ and $S_{A \cap B}$ are areas of $A$, $B$ and of their intersection. Finally, the relations of type 3) get the score $s_\mathrm{e} = -\infty$. The first two relations are effective if both symbol candidates are selected in the solution, the third one is effective when the arrow is selected and one of the connected uniform symbols is not. Negative scores of the relations express that they are unwanted.

### 3.7.1. Max-Sum Formulation

The pairwise max-sum labeling problem[3] [Werner, 2007] (a.k.a. computing the MAP configuration of a Markov random field, discrete energy minimization, or valued constraint satisfaction) is defined as maximizing the sum of unary and binary cost functions (potentials of discrete variables)

$$\max_{\boldsymbol{k} \in K^V} \Big[ \sum_{u \in V} g_u(k_u) + \sum_{\{u,v\} \in E} g_{uv}(k_u, k_v) \Big] \,, \tag{3.10}$$

where an undirected graph $G = (V, E)$, a finite set of labels $K$, and costs $g_u(k_u), g_{uv}(k_u, k_v) \in \mathbb{R} \cup \{-\infty\}$ are given. We maximize over assignments of labels from $K$ to nodes of $G$. Each node $u$ and edge $\{u, v\}$ is then evaluated by the cost given by $g_u$ and $g_{uv}$.

In our model, each symbol candidate defines a single node of the graph $G$. The edge is defined for each pair of interacting nodes (i.e., two symbol candidates in a relation). Two labels are used, $K = \{0, 1\}$, where 0 means the candidate is not selected as a part of the solution while 1 means it is. Values $g_u(k_u), g_{uv}(k_u, k_v)$ are set to express scores of symbol candidates and relations, and to model natural restrictions as follows: $g_u(0) = 0$ and $g_u(1) = s$ for each symbol candidate $u$ with the score $s$. Further, for all pairs of objects $\{u, v\} \in E$

1. $g_{uv}(1, 1) = s_c = -\infty$ if $u$ and $v$ are in conflict.
2. $g_{uv}(0, 1) = s_e = -\infty$ if $u$ is a symbol candidate and $v$ is an arrow connected to that symbol.

[3] We call it this way because it is our convention at CMP where we build on the research of Prof. Michail I. Schlesinger from National Technical University of Ukraine.

3. $g_{uv}(1,1) = s_o$ if $u$, $v$ are two non-arrow symbol candidates with overlapping boxes ($s_o$ is given by (3.9)).
4. $g_{uv}(k,\ell) = 0$ otherwise.

A good commensurability of various scores in the model is confirmed by experiments. We also tested a set up of the unary and binary potentials based on logarithms of scores, which did not lead to better results.

### 3.7.2. Example

We illustrate the structural analysis by a simple example. Figure 3.13a contains an input flowchart with labelled strokes. We assume that the following symbol candidates were detected in this diagram:

1: process $\{t_1\}$ with score $s_1$
2: connection $\{t_4\}$ with score $s_2$
3: connection $\{t_8\}$ with score $s_3$
4: terminator $\{t_8\}$ with score $s_4$
5: arrow $\{t_2, t_3\}$ $[1 \rightarrow 2]$ with score $s_5$
6: arrow $\{t_5, t_6, t_7\}$ $[1 \rightarrow 3]$ with score $s_6$
7: arrow $\{t_5, t_6, t_7\}$ $[1 \rightarrow 4]$ with score $s_7$

The strokes forming each symbol candidate are in curly brackets. For arrow candidates, the values in the square brackets say which symbols are connected by the arrow. The resulting max-sum model is depicted in Figure 3.13b.

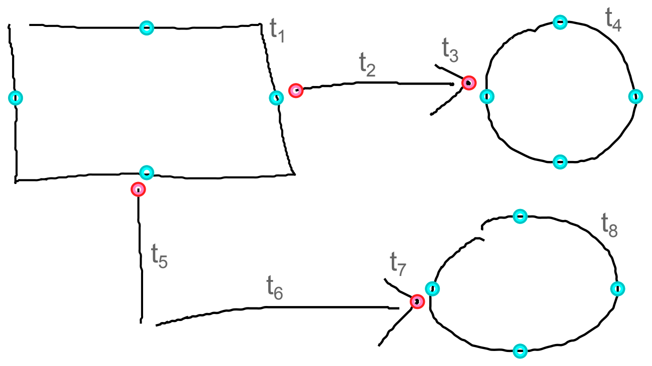
### 3.7.3. Solving the Optimization Task

The max-sum problem is NP-hard, although some of its special forms, such as the submodular max-sum problem, can be solved in polynomial time [Werner, 2007]. Unfortunately, this is not our case. However, the size of generated graphs is not so big (72/50/84 nodes and 673/305/721 edges in average for FC_A/FC_B/FA). Therefore, general branch and bound solvers are able to solve them quickly (see Section 4.1). We tested the max-sum solver Toulbar2[4]. Its minor disadvantage is that it supports only non-negative integer costs and thus it is necessary to transform the score values. Another option is to formulate the max-sum problem as an integer linear program (ILP) and solve it using a general ILP solver. We tested IBM ILOG CPLEX library[5]. The conversion is based on the linear programming relaxation of the problem [Werner, 2007] and the ILP formulation is as follows:

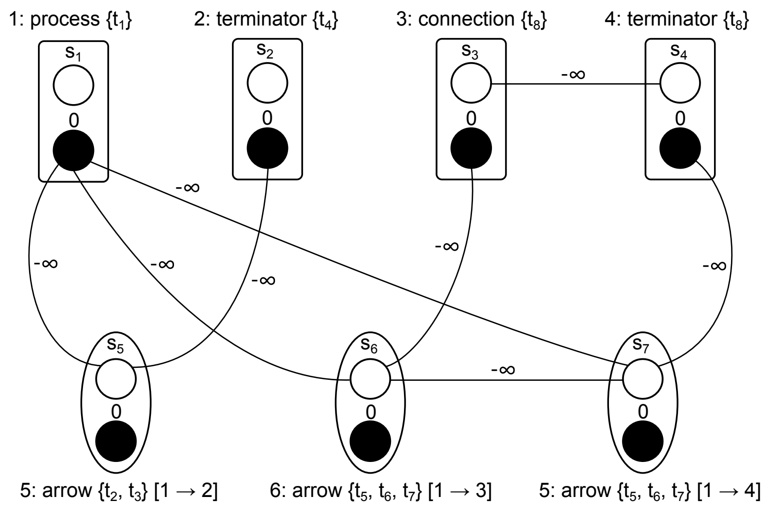
$$\max_{\boldsymbol{\mu}} \Big[\sum_{k\in K}\sum_{u\in V}\mu_u(k)g_u(k)+\sum_{k,\ell\in K}\sum_{\{u,v\}\in E}\mu_{uv}(k,\ell)g_{uv}(k,\ell)\Big] \tag{3.11}$$

$$\begin{aligned} \text{s.t.} \quad & \sum_{k\in K}\mu_u(k) = 1, && \forall u \in V, \\ & \sum_{\ell\in K}\mu_{uv}(k,\ell) = \mu_u(k), && \forall\{u,v\} \in E \;\wedge\; \forall k \in K, \\ & \mu_{uv}(k,\ell) = \mu_{vu}(k,\ell), && \forall\{u,v\} \in E \;\wedge\; \forall k,\ell \in K, \\ & \boldsymbol{\mu} \geq \mathbf{0}. \end{aligned}$$

[4] http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro
[5] http://www.ibm.com/software/integration/optimization/cplex-optimizer/



**(a)**



**(b)**

**Figure 3.13.** Input flowchart (a) consisting of 8 strokes $t_1, \ldots, t_8$ forming 3 uniform symbols and 2 arrows. Connection points are marked in blue and red for uniform symbols and arrows, respectively. There are 4 uniform symbol candidates since stroke $t_8$ is classified as *connection* (with score $s_3$) and *terminator* (with score $s_4$). The structural analysis is cast as a max-sum problem (b) where rectangular nodes represent uniform symbol candidates while elliptic nodes represent arrow candidates (formally, the nodes are of the same kind). Both possibilities for labelling a particular node are represented inside the node (white circle – label 1, black circle – label 0). Each label $k$ in a node $u$ is assigned by value $g_u(k)$. An edge connecting a label $k$ in a node $u$ and a label $\ell$ in a node $v$ is assigned by value $g_{uv}(k, \ell)$. Edges with zero costs are not shown.

If components of $\boldsymbol{\mu}$ are restricted to integral values only (i.e., they are 0 or 1), we obtain the desired ILP whose optimal value equals the optimal value of (3.10). The label $k_u \in K$ chosen in a node $u \in V$ is the only label satisfying $\mu_u(k) = 1$.

We made experiments with both formulations and compared the performance of Toulbar2 and CPLEX. Both solvers find the exact solution and the recognition result was the same. Toulbar2 was approximately 60 % faster on average. It appeared slower for the smallest diagrams; this was caused by the data exchange trough files, which was the only possibility supported by the binary distribution of Toulbar2. Some minimal time was thus needed for the solver initialization. This minor technical limitation may be resolved in the future. Details on runtimes are provided in Section 4.1.

## 3.8. Text Recognition

The text recognition is the last step of the diagram recognition pipeline. It is performed when the diagram structure is already known. Strokes removed during the text/non-text separation step together with strokes which were not identified as a symbol are processed. The diagram structure provides enough information to group the unused strokes into text blocks. Two types of text blocks are distinguished – those labeling a uniform symbol and those labeling an arrow. The grouping is accomplished in two stages accordingly. Text blocks inside symbols are found first, text blocks labeling arrows are found subsequently.

Each text block labeling a uniform symbol $U$ is determined by the area of the symbol. It includes all unused strokes whose bounding box centroid is located inside the bounding box of $U$.

Text blocks labeling arrows can be found easily by a grouping based on the spatio-temporal proximity. Only strokes unused in the first stage are left. Text blocks are determined by split points found in the sequence of the remaining strokes. A split point is defined as a point in the stroke sequence where two consecutive strokes are not linked. The measure is given as a weighted sum of six following features:

1. Distance between centroids of both bounding boxes.
2. Distance between the right edge of the first bounding box and the left edge of the second bounding box.
3. Distance between top edges of both bounding boxes.
4. Distance between bottom edges of both bounding boxes.
5. Distance between the last point of the first stroke and the first point of the second stroke.
6. Difference in length of diagonals of both bounding boxes.

All the features are normalized by the general distance threshold *distThresh*. There must be set a threshold to decide which points are the split points. We found suitable values of the weights and the threshold empirically. The text blocks are salient objects in remaining unused strokes. It is illustrated by an example in Figure 3.14 where the remaining text strokes labeling arrows are shown in red. Therefore, it is a relatively easy task to find suitable values. The text blocks are attached to the closest arrows afterwards.

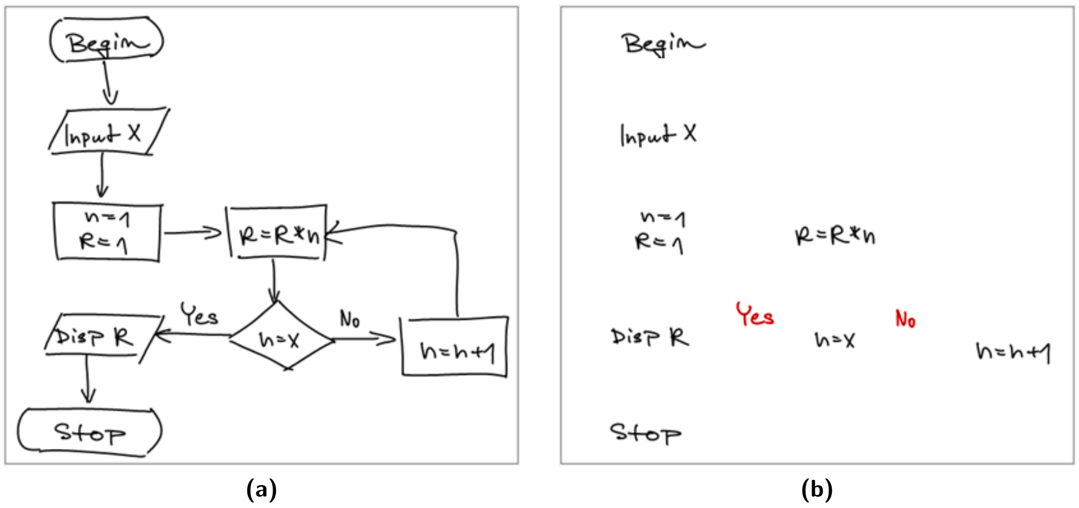


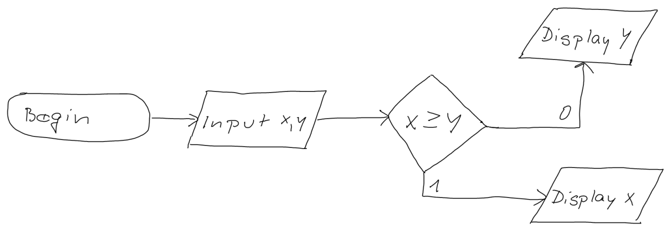
**Figure 3.14.** (a) Input flowchart and (b) text strokes. Text blocks are salient objects. Blocks labeling arrows are highlighted in red.

In the end, all detected text blocks are recognized. In our demo application, we use the MS Text Recognizer, which is a tool of the .NET framework. However, this recognition is out of the scope of this thesis. The FC_A database does not even provide meaning of the text and thus we did not measure the accuracy of the text recognizer. We considered text blocks as another symbols and checked their correct segmentation only making the text blocks segmentation the goal. The MS Text Recognizer is designed for plain English text and exploits a vocabulary. Therefore, it fails inevitably when recognizing text blocks containing math expressions in the case of flowcharts or state labels containing subscripts in the case of finite automata.

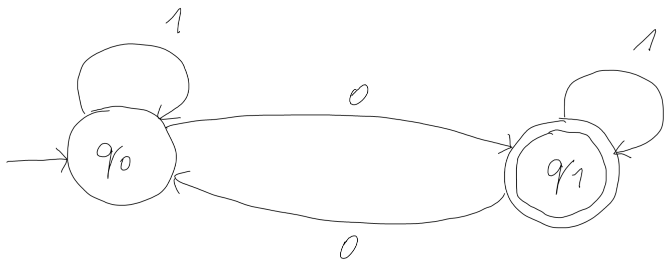
## 3.9. Adaptation of the Recognizer for New Domains

The proposed diagram recognizer was designed for flowcharts. We decided later to adapt it for finite automata. Here we will describe the necessary effort this adaptation required. The reader can get an idea what it requires to adapt it for additional domains. As we stated earlier, the adaptation mainly requires to retrain all the classifiers. However, this process is highly automated since the descriptors and all methods remain the same. Although the model used in the structural analysis remain the same, it is advantageous to exploit the knowledge of the domain to increase the accuracy. Most of the differences between the domains were mentioned earlier. This section serves as a recapitulation. We will describe it in more detail and provide illustrative examples. We will point out differences in the domains, which need our attention one by one.

- The ration between the number of text and non-text strokes is different in both domains. The text in finite automata is very limited. It is used for very short labels only. Therefore, a different bias needs to be used in the text/non-text separation step. However, the classifier used for the separation must be retrained anyway and the estimation of the bias is done empirically based on the data. The example showing the different character of text in both domains is shown in Figure 3.15.
- Symbols consist of different number of strokes. Flowcharts contain more complex symbols while finite automata contain only states, final states, and arrows, which



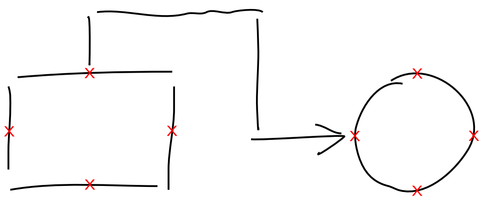
**(a)**



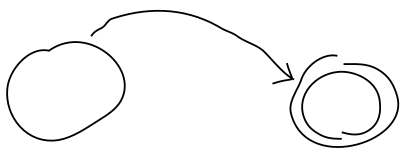
**(b)**

**Figure 3.15.** Illustrative example showing the different character of text in a) flowchart and b) finite automata domain. The ratio of text/non-text strokes is usually much higher in the case of flowcharts.

are usually formed from one or two strokes only. It leads to a smaller number of combinations and recognition might be faster. On the other hand, strokes are always linked together by their endpoints and arrows are connected to symbols at certain connection points only. It is not the case for finite automata. Therefore, each pair of points must be considered when we measure distance between strokes or between an arrow and a symbol. These differences are illustrated in Figure 3.16.



**(a)**



**(b)**

**Figure 3.16.** Illustrative examples showing differences in the construction of symbols and in connection of arrows with symbols. a) shows a flowchart where arrows can be (intermittent) polylines. However, the strokes are always linked by their endpoints. Moreover, arrows are expected to be connected to symbols at one of its connection points (red) only. b) shows a finite automaton where arrows are often curved and thus consist of less strokes. However, it can be connected to a symbol at any point on its surface. Additionally, strokes of one symbol may not be linked by their endpoints (e.g. the final state).

- Finite automata domain has a smaller number of symbol classes and thus the classification should be theoretically more precise. There are two particular difficulties. The final state symbol consist of two concentric circles and the state symbol is represented by one circle only. Therefore, the inner or outer circle of a final state is often classified as a state. Because it is easier to draw the first circle precisely, the score of state symbols is usually higher than the score of final states, because they are not drawn so precisely. Although this misclassification can be often solved by the structural analysis, it is advantageous to artificially increase the score of the final states (e.g. the square root of the score which is from interval $[0, 1]$). The second difficulty is that there are initial arrows entering the first state. These arrows cannot be detected by our arrow detector because they do not link two symbols. Therefore, they must be detected using an appearance based symbol classifier. The problem is that they look exactly the same as other arrows, which are among the negative examples. Our solution is to remove other arrows from the negative examples. It can happen then that an arrow will be classified as an initial arrow and will have higher score. We use here the same trick as we used for the final states. We artificially increased the score of arrows. The justification for this unnatural intervention is the following. In both cases, only one symbol can be misclassified as the other one and it cannot happen vice versa. An initial arrow cannot be classified as an arrow because the other symbol to link it is missing. Similarly, the state cannot be classified as a final state because the second circle is missing. Both cases are illustrated by an example in Figure 3.16

# 4. Evaluation

A complete description of the proposed diagram recognizer for arrow-connected diagrams from domains of flowcharts and finite automata has been provided in previous chapter of this thesis. Individual steps of the pipeline dealing with various recognition subtasks have been presented along with the intermediate results. Here we report the performance evaluation of the entire diagram recognizer.

We also encourage the reader to experiment with our demo application available at `http://cmp.felk.cvut.cz/~breslmar/diagram_recognizer/`. I allows to draw a diagram directly on the canvas or to import a previously captured data in the InkML format (e.g., diagrams form the benchmark databases). The application performs the recognition and provides a visualization of the beautified diagram using Graphviz. The formal representation of the recognized diagram can be also saved in the DOT language.

This chapter is divided into two sections: Section 4.1 presents the achieved results on the FC_A, FC_B, and FA benchmark databases. It shows the precision and the time complexity of the proposed recognition system. It provides comparison with alternative state-of-the-art approaches. Section 4.2 brings extensive analysis of the system. It analyses failure cases and explains the impact of individual steps of the pipeline to the overall precision. It helps to find and understand the limits and weak points of the proposed diagram recognizer.

## Chapter Outline

## 4.1. Experiments

We present an overall performance evaluation of the proposed diagram recognition system (abbreviated $\gamma$). We made a comparison with two state-of-the-art methods: the grammar based method (abbreviated $\alpha$) by Carton et al. [Carton et al., 2013] and the purely statistical method (abbreviated $\beta$) by Delaye [Delaye, 2014], and compared our performance to their published results. The former approach was evaluated on the FC_A database only, while the second one is more recent and was evaluated on the FA database as well. FC_B is new (introduced in [Bresler et al., 2016]) and results achieved on this database are thus reported without a comparison to others. However, it shows that the higher quality data match well to capabilities of current devices. A significantly higher accuracy is achieved.

We use two basic metrics to measure the recognition precision. The first metric (SL) assesses the correct stroke labelling. We assign each stroke a label of the symbol class it is classified to, and this assignment is checked against the ground truth in the database. The second metric (SR1) assesses the correct symbol segmentation and classification. It is more informative and provides a better insight into the recognition result quality.

The most direct way to decide if a symbol was correctly recognized is to check whether it comprises exactly the same strokes and has the same label as the annotation. We call this criterion the strict one. The metrics SL and SR1 are common and were used by authors of both systems $\alpha$ and $\beta$ and thus they allow fair comparison of individual systems. However, the exact stroke matching is not necessarily required for the correct diagram structure recognition. Users sometimes draw symbols by multiple strokes when correcting or beautifying symbols. Some strokes are redundant in some way. See Figure 3.6 for an example. It might happen that although the symbol is recognized correctly, it is not formed of exactly the same strokes as its annotated pattern in the database. For more insight, we define an additional more relaxed criterion (SR2) based on matching each annotated symbol with one of the recognized symbols. Two symbols match if they are of the same class and their bounding boxes overlap by 80 %. This value was estimated empirically. It must be high enough to forbid matching of different symbols. At the same time, it must be low enough to allow matching of symbols with shrunk bounding boxes due to missing redundant strokes. In the case of arrows, we also require that they connect the same symbols and have the same direction.

The error rate is expressed as the number of unmatched annotated symbols. This criterion is more meaningful. It was used to assemble the histogram in Figure 4.1 showing how many diagrams were recognized with a particular number of errors. The best results were achieved for the FA database, where nearly 80 % of diagrams were recognized correctly. The worst results were achieved for the FC_A database, probably because of its low quality. Inputs are noisy and there is no temporal information, thus it has not been possible to synthesize additional samples to train our classifiers. Even so, we have still achieved state-of-the-art precision, see details in Tables 4.1, 4.2 and 4.3. Differences in symbol segmentation and recognition results given by the two criteria are shown in Table 4.4.

The system has been implemented in C#, and tests were performed on a standard tablet PC Lenovo X230 (Intel Core i5 2.6 GHz, 8 GB RAM) with 64-bit Windows 7. It is a typical device, which can utilize our recognizer. Using this computer for the experiments shows that the recognizer allows real time performance on average machines. In fact, the target on tablet PCs and tablets with similar The average runtime needed for recognition was 0.78 s, 0.89 s, and 0.69 s for diagrams from FC_A, FC_B, and FA, respectively. This means that our system is faster than the grammar based system pro-

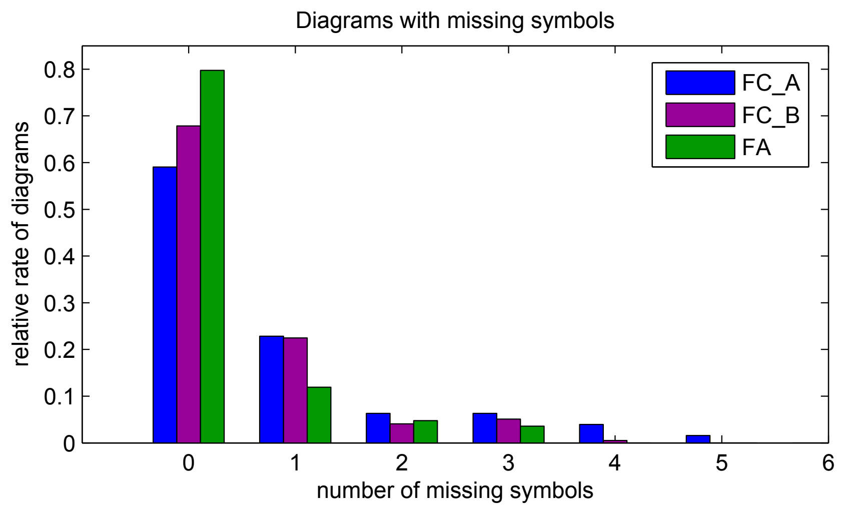


**Figure 4.1.** Counts of diagrams by the number of missing symbols in the result.

| Class | SL [%] | | | SR1 [%] | | |
|---|---|---|---|---|---|---|
| | ($\alpha$) | ($\beta$) | ($\gamma$) | ($\alpha$) | ($\beta$) | ($\gamma$) |
| Arrow | 83.8 | – | 87.5 | 70.2 | – | 76.6 |
| Connection | 80.3 | – | 94.1 | 82.4 | – | 95.1 |
| Data | 84.3 | – | 95.3 | 80.5 | – | 90.5 |
| Decision | 90.9 | – | 88.2 | 80.6 | – | 72.9 |
| Process | 90.4 | – | 96.3 | 85.2 | – | 88.6 |
| Terminator | 69.8 | – | 90.7 | 72.4 | – | 89.0 |
| Text | 97.2 | – | 99.2 | 74.1 | – | 89.7 |
| **Total** | **92.4** | **93.2** | **96.3** | **75.0** | **75.5** | **84.2** |

**Table 4.1.** Recognition results for FC_A database. Comparison of the proposed recognizer ($\gamma$) to the grammar based method ($\alpha$) and to the purely statistical method ($\beta$). We list correct stroke labelling (SL) and symbol segmentation and recognition measured with the strict (SR1) method.

posed by Carton et al., which has an average recognition time 1.94 s and slower than the purely statistical approach by Delaye and Lee with an average recognition time 80 and 52 ms for FC_A and FA, respectively. Table 4.5 lists the minimal, maximal, average, and median time needed to solve the max-sum problem and to perform the entire recognition. The values confirm that the optimization is solved relatively fast as it consumes only a small proportion of the whole processing time. The speed of the solver Toulbar2 is compared with CPLEX in Table 4.6.

## 4.2. System Analysis

Here we report on additional experiments performed to analyse the impact of the individual steps of the pipeline on the overall precision. We investigated which steps of the recognition pipeline are responsible for misrecognition of symbols from individual classes. Some of the recognition failures are illustrated by examples and commented. We also tested the system having the advanced pipeline steps disabled one by one.

| Class | SL [%] | SR1 [%] |
|---|---|---|
| Arrow | 93.8 | 93.2 |
| Connection | 88.4 | 88.4 |
| Data | 96.1 | 93.8 |
| Decision | 90.3 | 92.0 |
| Process | 98.4 | 97.6 |
| Terminator | 99.7 | 98.9 |
| Text | 99.6 | 97.1 |
| **Total** | **98.4** | **95.3** |

**Table 4.2.** Recognition results for FC_B database. We list the correct stroke labelling (SL) and symbol segmentation and recognition measured with strict (SR1) method.

| Class | SL [%] | | SR1 [%] | |
|---|---|---|---|---|
| | ($\beta$) | ($\gamma$) | ($\beta$) | ($\gamma$) |
| Arrow | – | 98.0 | – | 97.5 |
| Initial arrow | – | 98.6 | – | 97.3 |
| Final state | – | 99.2 | – | 99.2 |
| State | – | 98.3 | – | 98.2 |
| Label | – | 99.7 | – | 99.2 |
| **Total** | **98.4** | **99.0** | **97.1** | **98.5** |

**Table 4.3.** Recognition results for FA database. We compared the proposed system ($\gamma$) with the purely statistical method by Delaye ($\beta$). We list the correct stroke labelling (SL) and symbol segmentation and recognition measured with strict (SR1) method.

### 4.2.1. Failure analysis

The system fails to recognize some diagrams even when the best approaches are used in each step of the pipeline. There are four possible reasons why a symbol may not be recognized properly: a) some of its strokes were misclassified as text, b) the symbol was not properly segmented, c) the symbol was rejected by the classifier, d) the structural analyser did not choose the symbol. The special case is the arrow, which is not segmented as a uniform symbol. We say that wrong segmentation is the reason for its misrecognition if the symbols it connects were not segmented correctly. Another exception is the text block, misrecognition of which is always caused by the misrecognition of a symbol it labels and thus it is not a part of the analysis. The rate of each reason for symbol misrecognition with respect to the symbol class is shown in Table 4.7. The average rates are shown for individual datasets in Figure 4.2. It turned out that the most frequent reason for failure was the symbol misclassification in the case of flowcharts and the wrong selection of symbol candidates by the structural analyser in the case of finite automata. Diagrams with a high number of misclassified symbols showing different kinds of errors in individual databases are analysed in Figures 4.3, 4.4, 4.5.

### 4.2.2. Advanced techniques analysis

We replaced the most advanced techniques for text/non-text separation, symbol segmentation, and symbol classification by our previous or naive approaches. The results are summarized in Table 4.8.

Our recognition system is robust enough to handle some remaining text in a diagram. On the other hand, it can barely recover when some shape strokes are removed.

| Class | SR1 [%] / SR2 [%] | | |
|---|---|---|---|
| | FC_A | FC_B | FA |
| Arrow | 76.6 / 78.3 | 93.2 / 94.3 | 97.5 / 98.1 |
| Connection | 95.1 / 96.0 | 88.4 / 89.3 | – |
| Data | 90.5 / 91.4 | 93.8 / 94.7 | – |
| Decision | 72.9 / 76.1 | 92.0 / 95.1 | – |
| Process | 88.6 / 89.9 | 97.6 / 98.4 | – |
| Terminator | 89.0 / 89.7 | 98.9 / 99.6 | – |
| Text / Label | 89.5 / 91.6 | 97.1 / 98.7 | 99.2 / 99.4 |
| Initial arrow | – | – | 97.3 / 97.3 |
| Final state | – | – | 98.2 / 98.6 |
| State | – | – | 99.2 / 99.2 |
| Total | **84.2 / 85.4** | **95.3 / 96.6** | **98.5 / 98.8** |

**Table 4.4.** Comparison of the symbol segmentation and recognition rates using strict (SR1) and structure based (SR2) method.

| Dtb. | Running time [s] | | | |
|---|---|---|---|---|
| | minimal | maximal | average | median |
| FC_A | 0.11 / 0.19 | 0.66 / 4.61 | 0.14 / 0.78 | 0.12 / 0.71 |
| FC_B | 0.11 / 0.46 | 0.22 / 3.56 | 0.13 / 0.89 | 0.12 / 0.83 |
| FA | 0.12 / 0.27 | 0.37 / 1.43 | 0.14 / 0.69 | 0.13 / 0.62 |

**Table 4.5.** Optimization/total running time.

| Database | Running time [ms] | | | |
|---|---|---|---|---|
| | minimal | maximal | average | median |
| FC_A | 114 / 2 | 655 / 917 | 136 / 230 | 123 / 218 |
| FC_B | 114 / 3 | 216 / 598 | 126 / 129 | 122 / 48 |
| FA | 116 / 19 | 370 / 720 | 137 / 235 | 129 / 229 |

**Table 4.6.** Running time consumed to solve the optimization by Toulbar2/CPLEX.

However, if there is a lot of remaining text, the system needs significantly more time for recognition and can eventually get confused. To demonstrate how the system is susceptible to the result of text/non-text separation, we evaluated it with three different text/non-text separation settings: a) using unbiased text / non-text classifier, b) using the perfect text removal based on the annotation, c) performing no text/non-text separation.

We used strokes grouping instead of the clustering to perform over-segmentation. The iterative strokes grouping is a naive method achieving high recall values at the cost of lower precision. SLAC is a more sophisticated method with a significantly increased precision and only slightly worse recall, which guarantees a speed-up of the system.

We evaluated the system with uniform symbols classifiers trained without artificial samples. These classifiers achieve a lower precision reflected in the lower accuracy of the whole system. Moreover, it has reduced the ability to reject clusters, which represent no symbols and thus the recognition time slightly increases. It is obvious that the importance of classifiers trained with artificial samples increases in the flowchart domain with a higher number of symbol classes. Unfortunately, the FC_A database does not contain the time information, which is necessary for the synthesis of the artificial samples, thus we could not do this analysis on the database.

### 4.2.3. Analysis findings

Based on the performed analysis, we come to the following conclusions:

**Text/non-text separation** is very precise. It achieved 100 % precision in the shapes class on the FA database and thus it was not responsible for a single error there. Even in the case of the flowchart domain, it was responsible for symbol misrecognition only in a few cases. Further analysis showed the importance of the text/non-text separation step. Without text being separated, the time complexity increased and the precision dropped significantly. On the other hand, it turned out that the results achieved with unbiased classifiers or ground truth based separation are comparable to the baseline. Naturally, the use of the ground truth led to the faster recognition, because all text strokes were removed.

**Symbol segmentation** is the main culprit in symbol misrecognition for both flowchart databases. This is caused by the fact that symbols consist of more strokes and users sometimes retrace them. Moreover, when a symbol is not segmented correctly, it inherently causes misrecognition of arrows connected to it. Further analysis showed that the naive strokes grouping can increase the precision. However, it can not compensate the increase of processing time.

**Symbol classification** is the second most responsible step for symbol misrecognition. Its failure means that the classifier rejected a symbol candidate. The use of artificial samples to train the classifiers is more important in the case of the flowchart domain, due to the higher number of symbol classes and the fact that some of them might be of a very similar appearance.

**Structural analysis** is the last step of the recognition pipeline, in which the symbols are selected from the symbol candidates. An error occurs in this step when the classifier does not reject a symbol, but gives more alternatives for classification and the structural analyser picks the wrong one. It typically happens in the case of two similarly looking symbols like state / final state or connection / terminator.

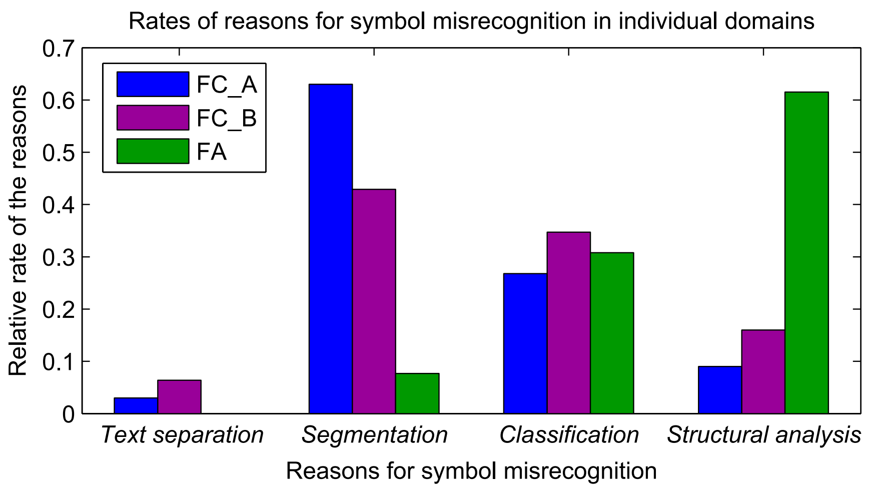


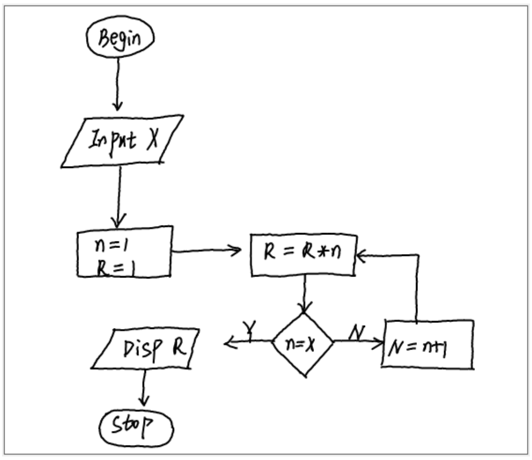
**Figure 4.2.** Average rates of causes of symbol misrecognition in individual domains.

| Class | # of misrecognized symbols [-] | Text separation [%] | Segmentation [%] | Classification [%] | Structural analysis [%] |
|---|---|---|---|---|---|
| Arrow | 220 / 91 / 18 | 0.3 / 7.7 / 0.0 | **58.5** / **51.7** / 5.6 | 35.2 / 37.4 / 33.3 | 6.0 / 3.3 / **61.1** |
| Connection | 5 / 13 / – | 0.0 / 0.0 / – | 17.7 / 0.0 / – | **82.3** / 38.5 / – | 0.0 / **61.5** / – |
| Data | 20 / 22 / – | 2.3 / 4.5 / – | **79.1** / 13.6 / – | 7.0 / 31.8 / – | 11.6 / **50.0** / – |
| Decision | 35 / 18 / – | 0.0 / 5.5 / – | **75.0** / **77.7** / – | 4.2 / 16.7 / – | 20.8 / 0.0 / – |
| Process | 35 / 9 / – | 2.7 / 11.1 / – | **67.6** / 33.3 / – | 13.5 / **55.6** / – | 16.2 / 0.0 / – |
| Terminator | 17 / 3 / – | 5.9 / 0.0 / – | **76.5** / 0.0 / – | 5.9 / 0.0 / – | 11.8 / **100.0** / – |
| Initial arrow | – / – / 2 | – / – / 0.0 | – / – / 0.0 | – / – / **100.0** | – / – / 0.0 |
| Final state | – / – / 1 | – / – / 0.0 | – / – / 0.0 | – / – / 0.0 | – / – / **100.0** |
| State | – / – / 5 | – / – / 0.0 | – / – / 20.0 | – / – / 0.0 | – / – / **80.0** |

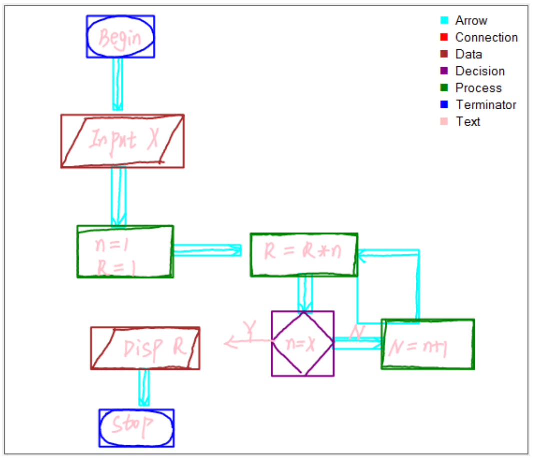
**Table 4.7.** Numbers of misrecognized symbols from the individual symbol classes with the rates of the reasons for their misrecognition (the dominant reasons are shown bolded). The displayed result correspond to FC_A / FC_B / FA databases.

| | | Analysis | | | | | |
|---|---|---|---|---|---|---|---|
| | | Baseline | Text separation | | | Symbol segmentation | Symbol classification |
| | | | unbiased | ground truth | no separation | grouping | no artificial samples |
| FA | SL [%] | 99.0 | 99.0 | 99.3 | 83.7 | 98.8 | 98.7 |
| | SR1 [%] | 98.5 | 98.5 | 98.9 | 82.0 | 98.2 | 98.1 |
| | SR2 [%] | 98.8 | 98.8 | 99.2 | 82.7 | 98.7 | 98.5 |
| | AT [ms] | 690 | 704 | 650 | 962 | 1 220 | 705 |
| FC_A | SL [%] | 96.3 | 96.2 | 96.6 | 94.1 | 96.5 | - |
| | SR1 [%] | 84.2 | 84.1 | 84.6 | 79.4 | 84.4 | - |
| | SR2 [%] | 86.4 | 86.2 | 86.6 | 82.7 | 86.7 | - |
| | AT [ms] | 780 | 770 | 762 | 2 273 | 1 060 | - |
| FC_B | SL [%] | 98.5 | 98.5 | 98.7 | 96.5 | 98.6 | 94.3 |
| | SR1 [%] | 95.6 | 95.5 | 96.0 | 90.5 | 95.2 | 87.7 |
| | SR2 [%] | 97.1 | 96.9 | 97.3 | 93.2 | 97.8 | 89.2 |
| | AT [ms] | 891 | 892 | 815 | 3 429 | 1 205 | 930 |

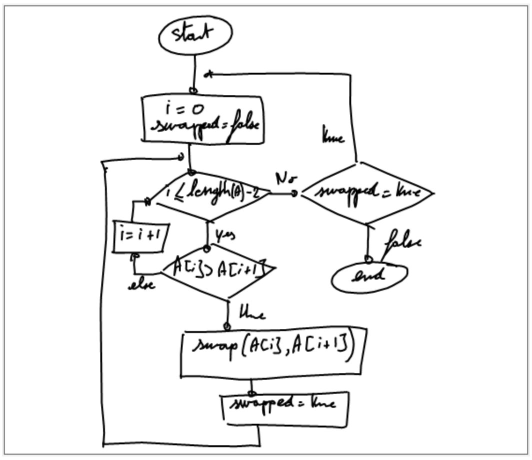
**Table 4.8.** Results of various analyses showing the effect of individual solutions to the steps of the pipeline. It is compared to the baseline, which was used to obtain the results in Section 4.1. We measured the correct rate of stroke labelling (SL), symbol segmentation and recognition measured with strict (SR1) and structure based (SR2) method, and average recognition time (AT).



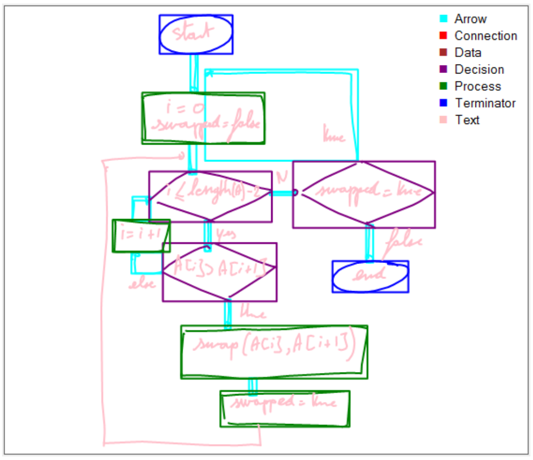
**(a)** input diagram writer10_1b



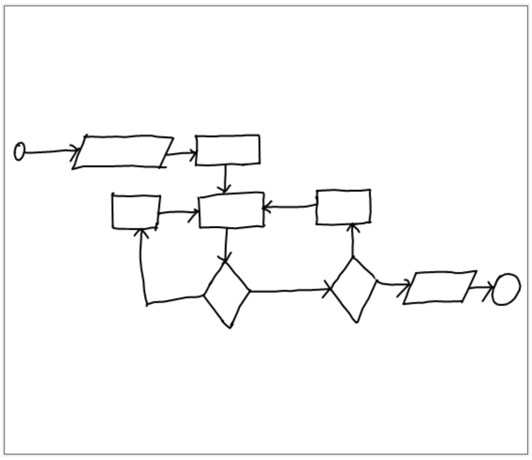
**(b)** unrecognized arrow – not properly connected to the symbol



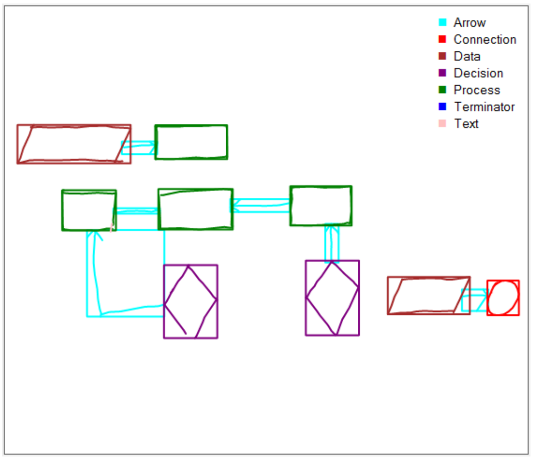
**(c)** input diagram writer12_7



**(d)** unrecognized arrow – connected to another arrow instead of the symbol

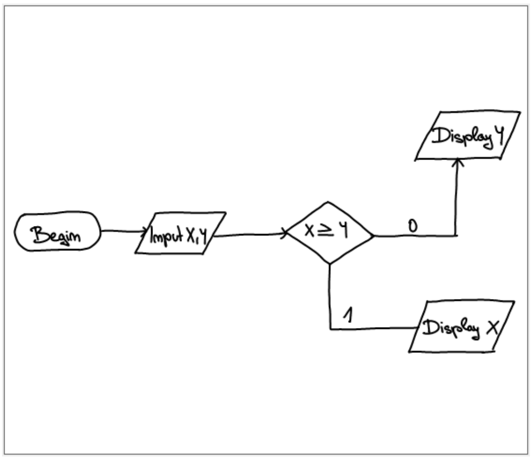


**(e)** input diagram writer06_13

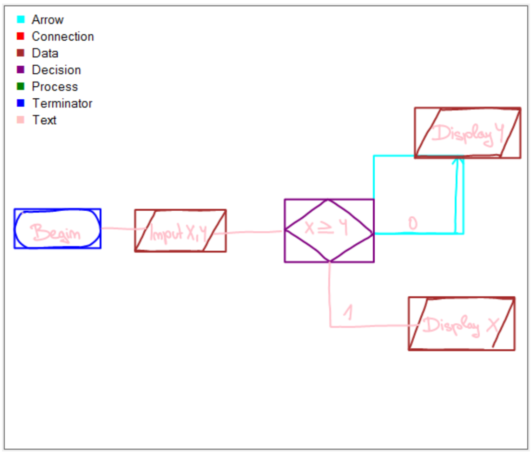


**(f)** missing text in the input biased text/non-text classification – symbols missing because their strokes were labeled as text

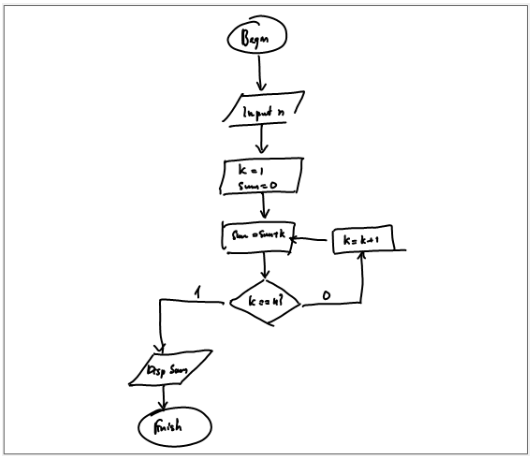
**Figure 4.3.** Examples of misrecognized diagrams from the FC_A database. The input diagrams are shown in the left column, recognition results are in the right column. Symbol colouring is explained in the legends, which are parts of the images. Each recognized symbol is surrounded by its bounding box.



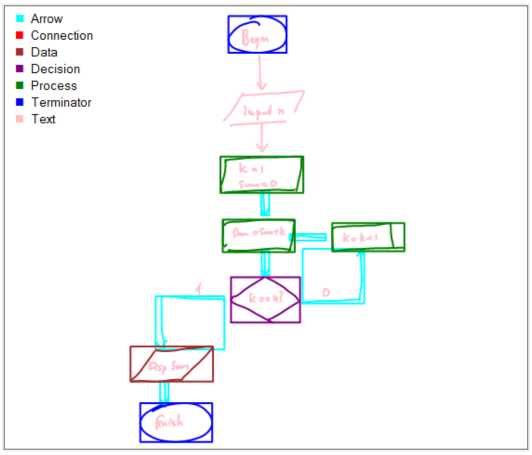
**(a)** input diagram writer018_fc_003b



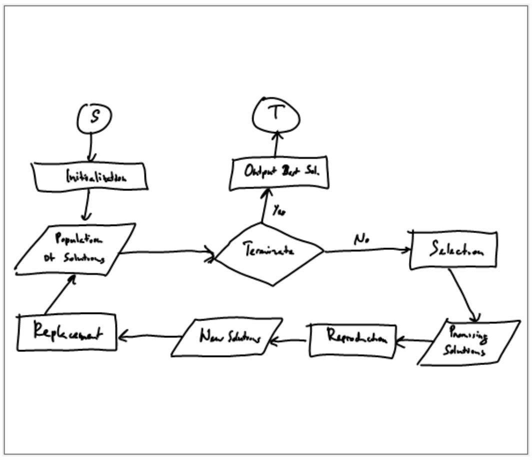
**(b)** unrecognized arrows – missing heads in the input



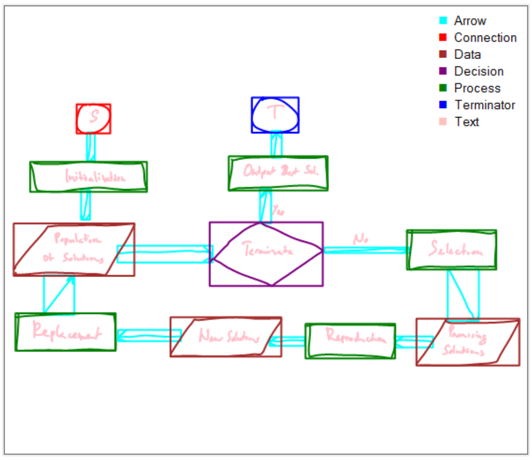
**(c)** input diagram writer019_fa_006b



**(d)** unrecognized data – interstroke gaps

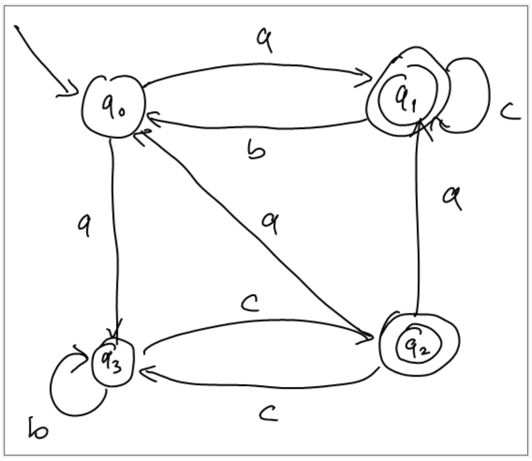


**(e)** input diagram writer020_fc_021

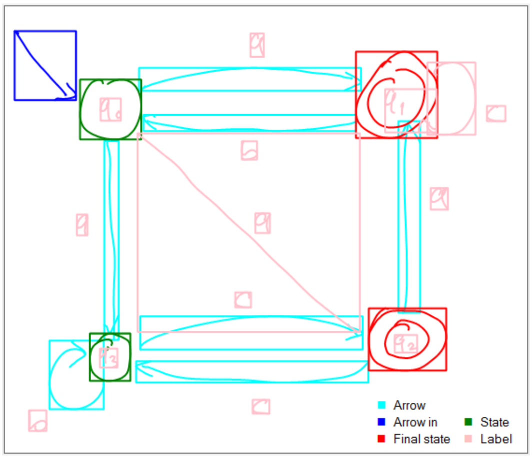


**(f)** misrecognized connector – visually similar to terminator

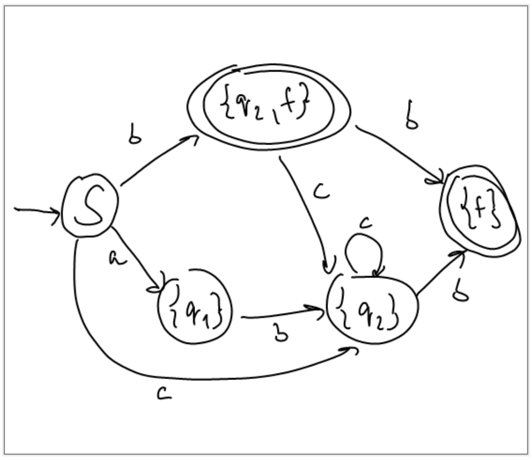
**Figure 4.4.** Examples of misrecognized diagrams from the FC_B database. The input diagrams are shown in the left column, recognition results are in the right column. Symbol colouring is explained in the legends, which are parts of the images. Each recognized symbol is surrounded by its bounding box.



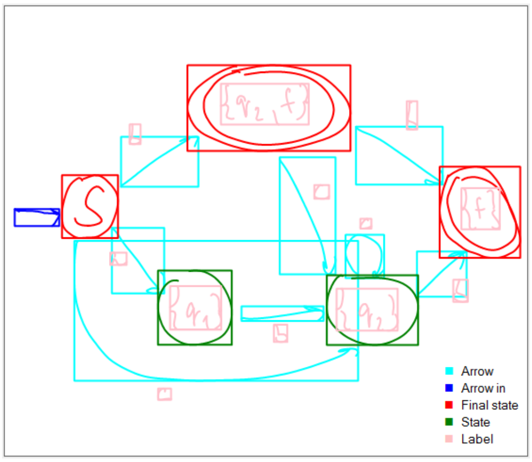
**(a)** input diagram writer020_fa_002



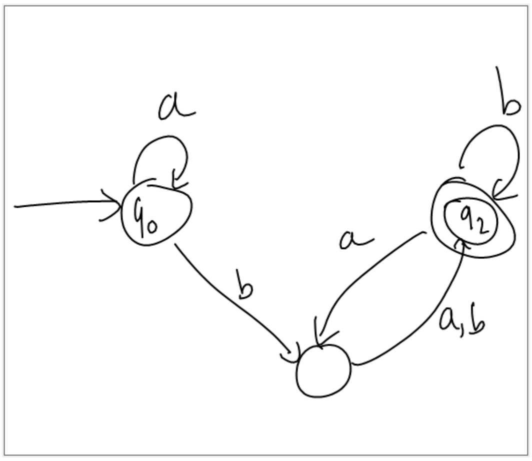
**(b)** unrecognized arrow – colliding heads



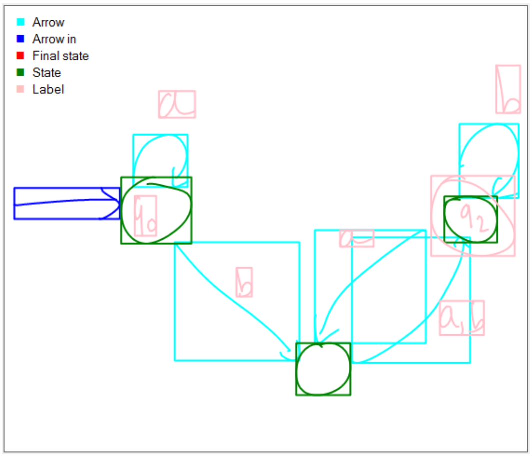
**(c)** input diagram writer022_fa_006



**(d)** a label recognized as a part of a symbol



**(e)** input diagram writer005_fa_001



**(f)** misrecognized final state – arrow is connected to the inner circle and thus the hypothesis of state gets higher a score

**Figure 4.5.** Examples of misrecognized diagrams from the FA database. The input diagrams are shown in the left column, recognition results are in the right column. Symbol colouring is explained in the legends, which are parts of the images. Each recognized symbol is surrounded by its bounding box.

# 5. Adaptive Tool for Object Segmentation

In this chapter, we investigate possibilities for structuring free-form sketches to facilitate fast and easy selection and rearrangement. Doing this, we aim for a domain-independent approach that works for all forms of free-form sketching. As shown in previous chapters, a lot of research in the field of symbol segmentation within well structured domains [Álvaro et al., 2014; Bresler et al., 2013; Feng et al., 2009; Ouyang and Davis, 2011; Stahovich et al., 2014] has exploited the knowledge of the domain to perform *segmentation by classification*. However, in the case of free-form sketching, such an approach cannot be used. Therefore, we use a data mining method called *cluster analysis*. This divides elements (in our case, strokes) into disjoint sets, where strokes within a cluster share similar characteristics. As we mentioned in Section 3.5, Delaye and Lee [2015] presented a machine learning approach based on Single-Linkage Agglomerative Clustering (SLAC) that successfully segmented sketches in several domains.

We contribute to the field of free-form sketching by introducing a novel smart selection tool called cLuster [Perteneder et al., 2015], which builds on the clustering approach by Delaye and Lee [2015]. Below, we first introduce the problem of free-form sketching and survey existing smart tools in Section 5.1. Section 5.2 brings a study we conducted to reveal different understanding of sketches. Our tool is described in Section 5.3. Evaluation and discussion are provided in Sections 5.4 and 5.7, respectively.

## Chapter Outline

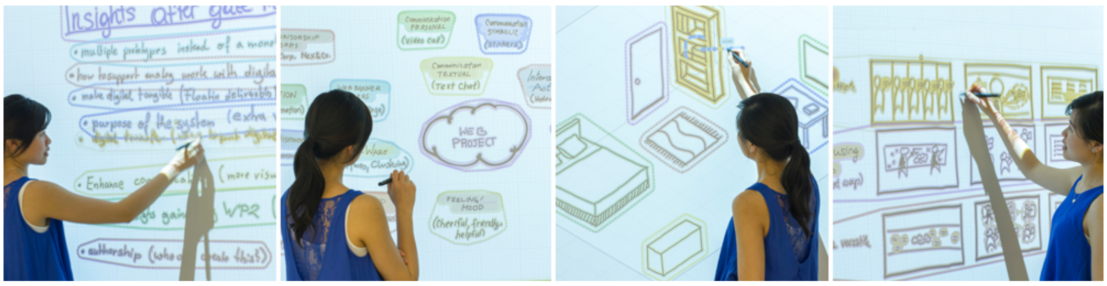


**Figure 5.1.** Users must often make large movements when working on large interactive surfaces. This quickly becomes cumbersome, especially when tools are not user-friendly and ergonomically designed.

## 5.1. Introduction

In the case of free-form sketching, the user can draw practically anything from abstract pictures to structured diagrams. Moreover, pictures can be combined altogether with diagrams, text, formulas, etc. Therefore, it is extremely difficult to understand the sketch and often impossible to formalize it. The mostly desired and expected feature is the seamless selection and manipulation with parts of the sketch. Our research was focused on large interactive surfaces – smart boards or digital whiteboards. Their large size can cause strain and fatigue with extended periods of use. Their applications should therefore be designed to mitigate these effects. Nevertheless, sketching tools that mimic the use of traditional whiteboards [Haller et al., 2010; Mynatt et al., 1999] are generally simplistic and lack the means to interpret sketched content. This results in physically demanding and time-consuming selection tasks when users wish to structure or rearrange their sketched content. To address this, many previous works have made attempts to improve selection tools [Grossman et al., 2009; Lank and Saund, 2005; Lindlbauer et al., 2013; Xu et al., 2012]. However, for structuring content, their solutions still necessitate a high degree of effort from the user. Either users must make repeated selections to rearrange content, or they must actively organize their content into layers. In contrast to this, others have taken the approach of introducing sticky note-like objects [Geyer et al., 2011; Guimbretière et al., 2001; Hilliges et al., 2007] to contain a single idea. This enables content to be structured or rearranged without the need for users to make explicit selections, but comes with the cost of a reduced degree of flexibility and freedom for content creation. Being able to efficiently cluster strokes into groups that are sensible to users, in order to easily structure and rearrange content in free form sketches is our main motivation.

Selection tasks are common in sketching tools. Using perceptual cues of *Gestalt theory*, Dehmeshki and Stürzlinger [2009, 2010] improved selection speed and results of selection tools. Igarashi et al. [1995] also used proximity and regularity to find structures in card stacks. Similarily, in Suggero [Lindlbauer et al., 2013], the authors present a selection tool that is based on the Gestalt Laws. Both cLuster and Suggero use the initial selection of a cluster to interpret the user's intention; however, they differ in their interaction possibilities. Furthermore, cLuster uses a more general feature set and superimposes a concept to handle different perspectives.

The work from Shilman et al. [2002, 2004, 2003] shares some similarities with cLuster, as it also uses a variety of spatial features. However, it primarily aims to distinguish handwritten text/symbols from sketches, as well as interpret an overall spatial structure, in order to convert them into type. This leads to a different way for dealing with ambiguity in grouping: while Shilman et al. [2002] uses context, cLuster uses an initial grouping example provided by the user to resolve this. ScanScribe [Saund et al., 2003; Saund and Mahoney, 2004] also includes mechanisms to form sensible groupings, however, the underlying algorithms are fundamentally different to our approach as the tool is pixel-based.

Smart Scribbles [Noris et al., 2012] provides the user-guided segmentation of drawings. In a semi-automated workflow, users mark parts of drawings using rough scribbles. The scribbles explicitly determine the number of segments and therefore can be seen as very rough initial selections. In contrast, cLuster uses one precise initial selection and determines the number of clusters automatically. Lazy Select [Xu et al., 2012] and Sloppy Selection [Lank and Saund, 2005] are also examples of tools that interpret the users' intention even with an imprecise input. The authors analyse the performed gesture as well as the structure of the existing content to increase certainty about the users' intention when only imprecise input is available. Handle Flags [Grossman et al., 2009] is a localized selection technique to select single strokes or their clusters of ink strokes that are clustered by proximity. While some of these selection tools perform analysis for all objects in the background, the goal is usually to support the user in creating one selection. In contrast to this, we aim to cluster all strokes and enable ways to interact with them as objects that respond.

Flatland [Mynatt et al., 1999] provides a feature called Auto-Segmenting that combines spatially close strokes for easy management of the spatial layout. It also introduces the idea of manipulating content indirectly when other objects are moved, which we extend in our work. Moran et al. [1997] extend the work done in Tivoli [Pedersen et al., 1993] by including implicit recognition of regions that allow for organizing content spatially. Despite these early efforts in large surface computing, improving the spatial arrangement of objects is a complex topic. Many systems leave this task up to the user [Guimbretière et al., 2001; Hailpern et al., 2007; Hilliges et al., 2007] or provide only minimal support [Geyer et al., 2011].

## 5.2. Background Study

To gain a better understanding of how users sketch on interactive whiteboards, what these sketches look like, and how users would cluster strokes into objects, we conducted a background study. We presented sketches to participants and asked them how they would cluster them. Then we tried to find common themes amongst the participants' clusters, resulting in a number of perspectives. These were then taken as the base for training the weights of our clustering algorithm.

### 5.2.1. Methodology

We gathered more than 300 different real snapshots made on interactive whiteboards, where $\approx 70\,\%$ were produced by MIL research group at the University of Upper Austria and $\approx 30\,\%$ by an external company. The sketches were created over a period of two years in real work scenarios by more than 15 individuals, and were the output of team meetings, workshops, brainstorming as well as note-taking sessions during presentations.

To keep the length of the study within reasonable time constraints but nevertheless achieve a sufficient variety in our selected sketches, we categorized them into seven different categories: *Sentences and Paragraphs, Word Lists, Structured Words with Annotation, Words in Visual Diagram Constructions, Diagrams/Image Sketches with Labels, Pure Diagrams, Pure Image Sketches.* It is a slightly modified set of the categories

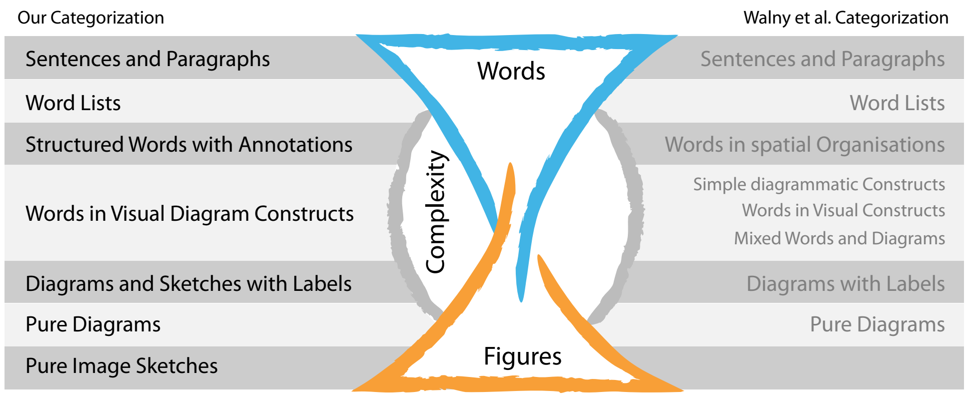
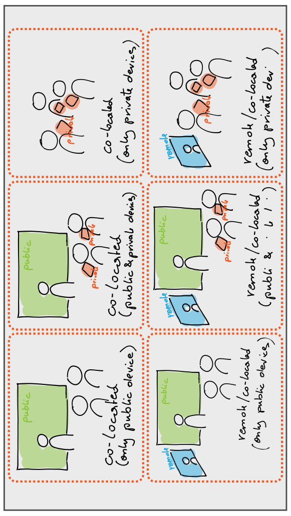

Our Categorization
Walny et al. Categorization
Sentences and Paragraphs
Word Lists
Structured Words with Annotations
Words in Visual Diagram Constructs
Diagrams and Sketches with Labels
Pure Diagrams
Pure Image Sketches
Words
Complexity
Figures
Sentences and Paragraphs
Word Lists
Words in spatial Organisations
Simple diagrammatic Constructs
Words in Visual Constructs
Mixed Words and Diagrams
Diagrams with Labels
Pure Diagrams

**Figure 5.2.** Sketch Categorization based on Walny et al.

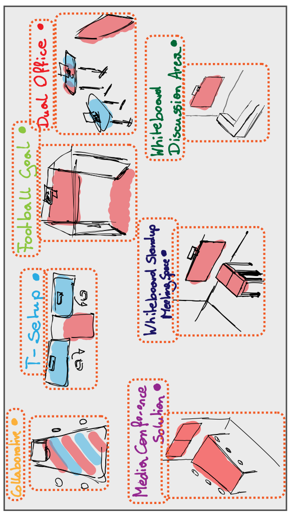
defined by Walny et al. [2011], which covers the range between primarily word-based constructs and pure diagrams as depicted in Figure 5.2.

In contrast to the data from traditional whiteboards presented by Walny et al., we could see that participants made use of the additional virtual space and produced less cluttered pages. While both types are used for quick communication, we feel that traditional whiteboards often house permanent information, whereas interactive whiteboards are used much more for the specific purpose of sensemaking and/or presentation.

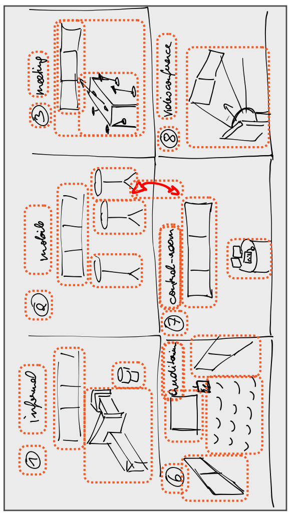
From the categorized sketches, we selected two exemplary pages for each category (three from *Words in Visual Diagram Constructions*) of sufficient complexity (i.e. non-trivial). Moreover, we tried to incorporate various drawing styles by using examples from different authors. The selected sketches were presented to 14 participants (6 females) in a counterbalanced order. The group of participants ($M = 30.6$, $SD = 8.0$) was from the local research campus and included novice users, who had never used an interactive whiteboard, as well as experts. We asked them to find and highlight potential clusters that they could identify as meaningful and important. In addition, for each clustered page, we asked them to name the type of the clusters they used. We repeated this process for each sketch until no additional clustering possibility could be identified.


public
co-located (only public device)
co-located (public & private device)
co-located (only private device)
remote
remote/co-located (only public device)
remote/co-located (public & private)
remote/co-located (only private dev.)

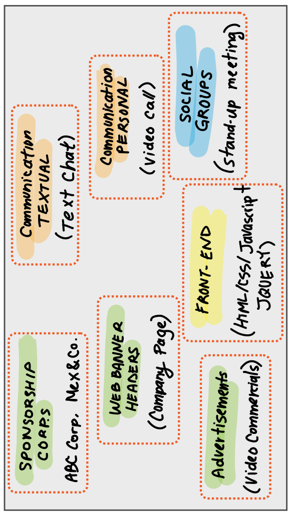
*Visual Subgroups:* Visual, non-atomic units that belong together, e.g. due to proximity


Collaboration
Media Conference Solution
T-Setup
Whiteboard Standup Meeting Space
Football Goal
Whiteboard Discussion Area
Dual Office

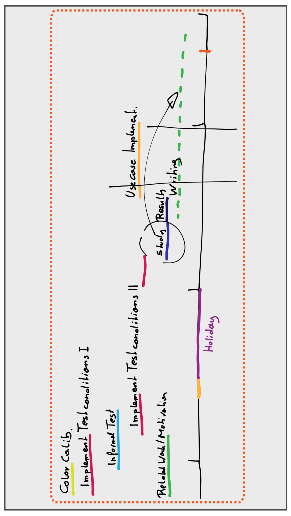
*Structural Similarity:* Units of similar structure (text vs. figures)



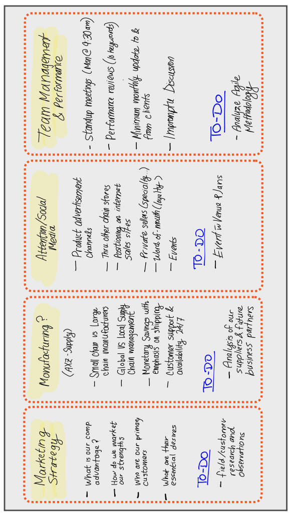
*Objects:* Atomic Units that cannot be split into meaningful subsets



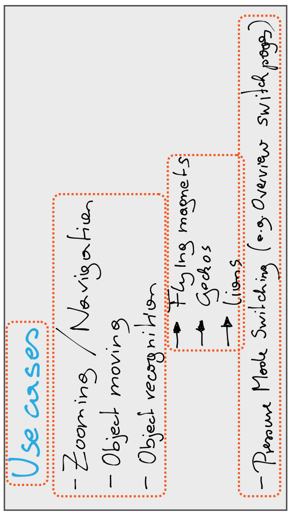
*Visual Text Units:* Any text that can be split visually (e.g. bullet points, text blocks etc.)



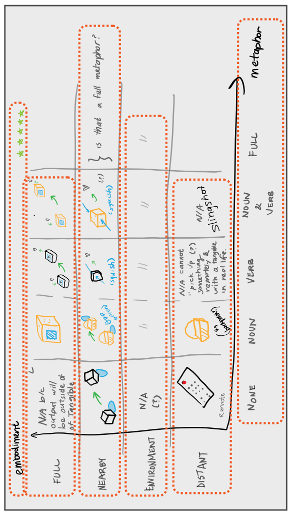
*Global:* Entire content on the page, incl. or excl. headline



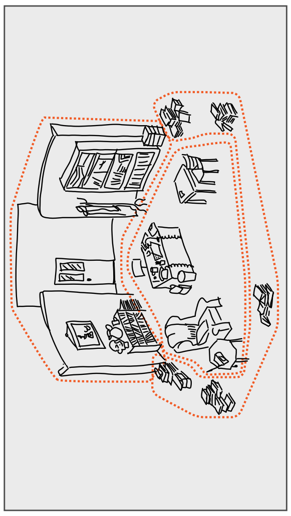
*Columns:* Vertically aligned units like table columns or item lists



*Hierarchies:* Hierarchical structures (e.g. multi-level headlines)



*Rows:* Horizontally aligned sketches (e.g. table rows or text lines)



*Categories:* Contextual categories, object categories, and sentences

**Figure 5.3.** List of perspectives used to annotate sketches and learn SLAC parameters.

### 5.2.2. Results

Overall, each participant had to find clusters for 15 different sketches, which took them approximately 45 min. each. On average, participants assigned 2.31 ($SD = 0.35$) alternatives to each sketch. It became apparent that persons with a high level of expertise in whiteboard usage identified more possibilities than persons with a low level of expertise. This suggests that demands on flexible interpretation rise with increasing experience. To avoid confusion with the sketch categories (cf. Figure 5.2), we will call a way of clustering a single sketch *perspective*. We categorized the alternative perspectives for each individual sketch and then summarized them for all sketches, using the Grounded Theory approach [Glaser and Strauss, 1967]. Finally, we identified nine main perspectives that are shown in Figure 5.3.

The bar chart presented in Figure 5.4 shows different perspectives in the order of their occurrence and their mapping to the initial categories they have been identified in. The occurrence is provided in a percentage number, which can be interpreted as the chance that a particular perspective was identified in a sketch. This means that the chance of being able to segment a sketch into Visual Subgroups is almost 40 %, while segmentation following Categoric Units is applicable with only a 7 % probability. On average, a perspective occurs in 5.33 ($SD = 1.05$) different categories. This indicates that they are quite universal and work over a large spectrum of sketches. Thus, we are confident that the identified perspectives work as a basis for an automatic clustering algorithm, to provide training datasets for different alternatives of segmentation.

## 5.3. Smart Clustering Tool

When aiming to provide users with a tool that allows for high-level interaction with free-form sketches, the biggest challenge is to cluster existing strokes into sensible groupings. Since users may write or draw anything, such sketches inherently lack an expected structure. Even worse, as we have learned in our background study, one specific sketch can be interpreted in multiple ways. Also, basic objects and their meaning might differ between users, meaning that classifiers used to detect or recognize objects (e.g. symbols) cannot be used. Therefore, we make use of data mining methods - specifically cluster analysis. This is commonly used to find useful information in unknown data. In our case, it is the information about which strokes belong together and represent an object.

### 5.3.1. Single Linkage Agglomerative Clustering

As we already mentioned in Section 3.5, Delaye and Lee [2015] showed that objects of interest in handwritten documents could be found using Single-Linkage Agglomerative Clustering (SLAC), equipped with a properly trained distance function. It is a hierarchical bottom-up clustering technique. First singleton clusters consisting of a single stroke are initialized. Larger clusters are created by iteratively merging the two closest clusters based on the distance. Using a Single-Linkage clustering approach implies that the distance between two clusters is given by the distance between their two closest elements and not a cluster average. This permits an efficient real-time implementation $\mathcal{O}(n^2)$, where $n$ is the number of strokes. The distances between strokes is only calculated once and not iteratively. A link is created at each merging step, which contains information about the two clusters it links and the distance between them. This results in a tree structure, a so-called *dendrogram*, that can be cut by using a suitable distance threshold (see Figure 5.5). The distance between two strokes $s$ and $t$ is given by a

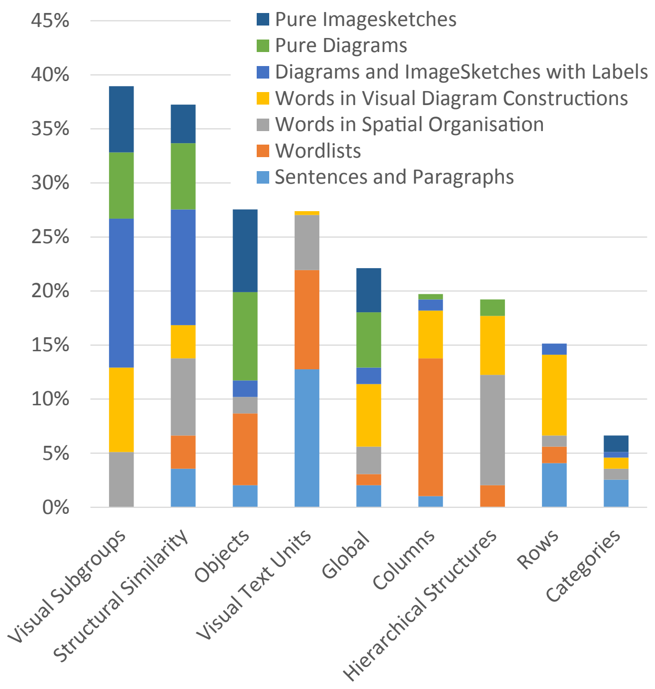


**Figure 5.4.** The occurrence of the different perspectives in percent of the sketches where they have been identified in. The colours show the mapping to the sketch categories to indicate the relatively even distribution of the perspectives.

weighted sum of their features:

$$d(s,t) = \sum_{i=1}^{k} w_i \, d_i(s,t), \tag{5.1}$$

where $w_i$ is the weight for the feature $i$ that needs to be learned and $d_i$ is the value of the feature. Previously annotated data is used to find suitable weights and the threshold to cut the dendrogram.

For training purposes, two types of links are defined. $C$-links, which connect two different clusters and $M$-links, which connect the two most distant sub-clusters of one object (see Figure 5.5). To allow a cluster containing an object, the distance of $C$-links must to be bigger, while the distance of $M$-links must be smaller than the threshold. Otherwise the two clusters are merged. For details of the algorithm, see the original paper [Delaye and Lee, 2015].

## 5.3.2. Our SLAC Adaptation

Delaye and Lee [2015] achieved promising results with segmenting benchmark datasets in domains like flowcharts, finite automata diagrams, mathematical expressions, loosely defined text blocks, and figures in free-hand sketches. However, they modified their weights and thresholds to suit each domain. They also used different subsets of features for different domains. Unfortunately, we are not dealing with any particular domain and we do not know in advance what the user's perspective is. Learning a new set of parameters requires a lot of data (at least ten annotated sketches) and takes a considerable amount of time (from minutes to hours depending on the complexity and the amount of data). Thus, it cannot be done in real-time for each individual input.

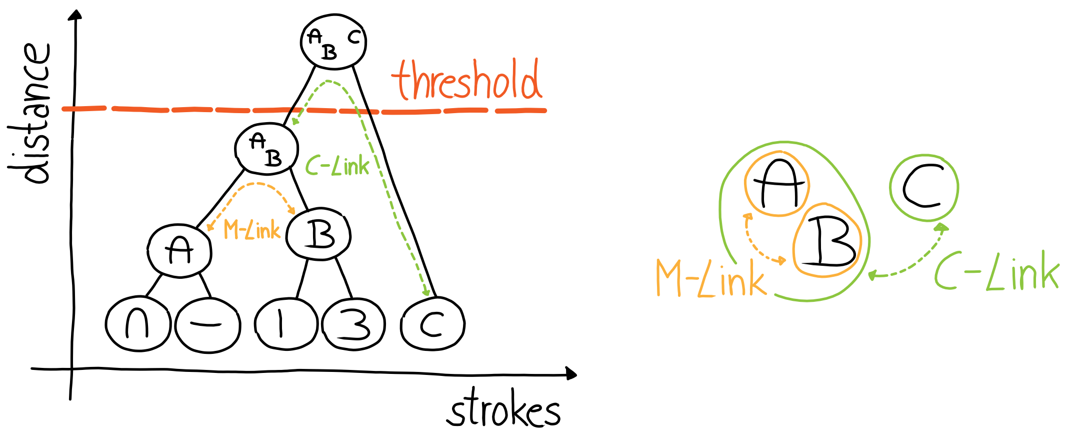


**Figure 5.5.** Illustrative example of a dendrogram and its cutting. The $M$-link is the link between the most distant sub-clusters within a cluster. The $C$-link is the link to the next cluster that will be merged.

Instead of aiming to learn the SLAC parameters for a specific domain, we used the perspectives found in our background study to define characteristics for the objects of our interest. This is why we performed our background study on a broad range of sketches. We skipped the perspective *Global*, where the entire sketch was grouped into one cluster and the perspective *Categories* that seemed of minor importance (see Figure 5.4). Summarizing, we ended up using seven perspectives from our background study: *Visual Subgroups, Structural Similarity, Objects, Visual Text Units, Columns, Hierarchies, Rows*. We used these perspectives to annotate training data and to train our system. This requires an additional layer to be built on top of the clustering method, which combines several sets of weights (from each perspective) learned from different annotated data to make the clustering tool suitable for an individual sketch viewed under a particular perspective (see Figure 5.6). Based on this clustering system, we can build tools that provide a quick adaptation according to the user's intentions.

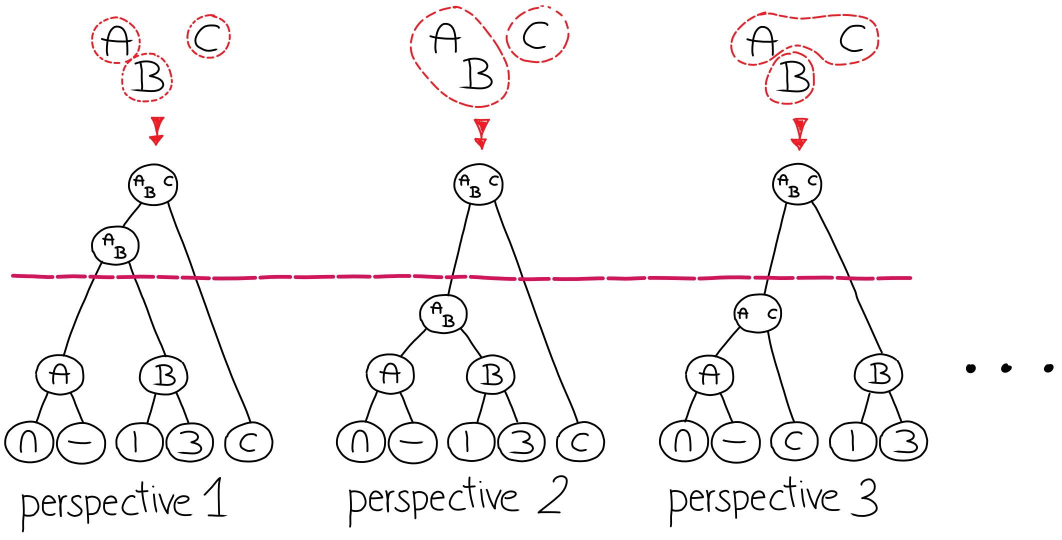


**Figure 5.6.** The different perspectives can be imagined as multiple dendrograms that provide different opportunities for clustering.

### 5.3.3. Implementation

Our implementation is based on the learning algorithm from [Delaye and Lee, 2015]. Finding the right suitable measure for defining the distance between two strokes is a challenge. Typically, features contain geometric, spatial, and temporal characteristics. Delaye and Lee used a Greedy approach in which a new set of features was added to improve the result iteratively until the improvement was smaller than a certain threshold.

We wanted to use as many features as possible to deal with the significant variety of input in free-hand sketches. However, it turned out that too specific features can cause a loss of generality (details in *Discussion* in Section 5.7). Therefore, we did not use features based on the speed of drawing, which could be misleading in the multi-user scenario. Individual users may draw the same things with very different speeds. Additionally, we omit computationally expensive features like the minimal, maximal, and average distance between all pairs of points of two strokes, as well as features that use stroke intersection information. Eventually, we used the same features that Delaye and Lee used for segmentation of symbols in flowcharts and finite automata. These features and their descriptions are presented in Table 5.1.

| Features | Characteristics |
|---|---|
| F1 | Minimum and maximum distance from any point of one stroke to the center of the other stroke's bounding box |
| F2 | Minimum and maximum distance from any point of one stroke to an extremity (first or last) points of the other stroke |
| F3 | Minimum distance between extremity points of the two strokes |
| F4 | Left, right, top, and bottom offset of the bounding box coordinates |
| F5 | Ration of horizontal, vertical, and overall overlapping of strokes bounding boxes |
| F6 | Absolute difference in strokes bounding boxes width, height, and surfaces |
| F7 | Number of strokes between the two strokes in the original drawing sequence |
| F8 | Difference in intrinsic parameters like arc length, closure, and ink intensity |

**Table 5.1.** List of the features we used.

We trained a set of parameters (the feature weights and the threshold) for each perspective using associated annotated sketches (see Figure 5.6). To combine these sets of pre-trained perspectives and find a new suitable distance threshold, we use a linear combination of the original feature weights. This linear combination can be understood as a definition of a new distance function:

$$d_c(s,t) = \sum_{j=1}^{p} \alpha_j d_j(s,t), \tag{5.2}$$

where $d_c$ is the new distance function defined as a linear combination of the original distance functions $d_j$ of individual perspectives defined by Equation (5.1). Finally, $\alpha_j$ are coefficients of the linear combination, the new tunable parameters, and $p$ is the number of perspectives. Hence, two kinds of parameters can be adjusted in our system to adapt the clustering. The coefficients $\alpha_j$ define the distance function and thus they affect the structure of the dendrogram. A change of these parameters cause a need to rebuild the dendrogram. On the other hand, the distance threshold is just used to cut the dendrogram and thus changing it is a cheap operation. The threshold adapts the granularity of the clustering and hence it is also referred to as the *granularity* threshold.

### 5.3.4. Cluster Creation and Refinement

As we learned from our background study, the expectations of how a sketch should be clustered can be quite diverse. Furthermore, even if expectations were to match between users, one cannot expect even a very well-trained algorithm to be consistently accurate. For this reason, we took advantage of the parameters that allow us to configure the granularity and the type of clustering. To achieve the best performance, we implemented a two-step approach, where users can firstly influence the process of clustering by selecting an initial cluster and then change the proposed clusters manually if required.

#### Selection of the initial cluster

While the distance threshold directly affecting a general cluster size is easy to understand, the coefficients that describe the influence of the different perspectives are hard to comprehend for users. Therefore, we have to think of ways for users to implicitly rather than explicitly adjust these parameters.

A very quick solution is to use the equal coefficients for the linear combination (i.e. $\alpha_j = 1/n, \forall j = 1, \ldots, n$, where $n = 7$ is the number of the trained perspectives). This option may be sufficient when the objects of interest are salient and easily separable. The user can then directly change the granularity threshold, e.g. by using a slider.

However, in most of the cases, it is more tricky to separate the objects and it is necessary to adjust the coefficients of the linear combination. As it is not ideal to set the coefficients of the linear combination manually, we provide the user with the possibility to make an initial selection indicating one cluster, cf. Figure 5.7. Any favourite selection tool may be used to do this. If the clustering is invoked afterwards, our implementation does not only aim to find the required threshold that would create desired clusters but also finds adequate coefficients of the linear combination providing the right way of clustering. As we mentioned earlier, the learning algorithm uses two types of links – $M$-links and $C$-links. The selected cluster is used to extract one $M$-cluster (or none if the cluster consists of one stroke only). However, we have no other clusters to extract $C$-links. Therefore, we search for a single stroke closest to the selected cluster to create a new $C$-link.

Obviously, the initial cluster gives very limited information: 0 or 1 $M$-link and 1 $C$-link. It is woefully little for learning parameters of the original clustering method. However, it turned out to be enough to find proper coefficients of the linear combination of parameters from previously trained perspectives. To do this, we use exactly the same learning algorithm as described before. However, this time we use the distance function $d_c$ and learn linear combination coefficients $\alpha_j$ instead of the feature weights $w_i$. The algorithm still learns the threshold as well. We set the initial values of coefficients $\alpha_j = 1/n$ and of the threshold $h = \sum_{j=1}^{n} \alpha_j \cdot h_j$. Since we use a limited set of data, the algorithm works in real-time. In our experiments, the algorithm needed 340 ms on average using the standard tablet PC Lenovo X230 (Intel Core i5 2.6 GHz, 8 GB RAM).

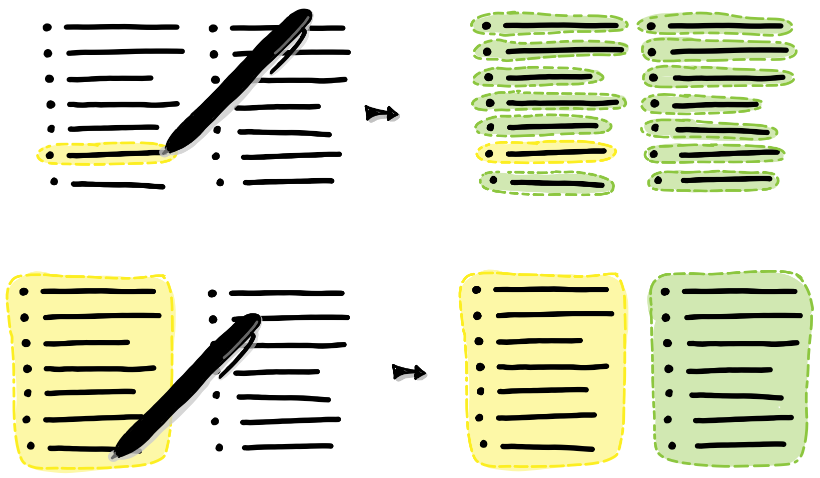


**Figure 5.7.** The initial selection serves as a template for the algorithm to cluster the rest of the page. If a bullet point is selected, the page is clustered into bullet points. If a column is selected, it is clustered into columns.

### Manual cluster refinement

Although the initial interaction for selecting the right cluster performs relatively well, it is necessary to provide error correction techniques [Mankoff et al., 2000a,b]. Especially due to ambiguous expectations in our scenario, it might be necessary to adjust some of the wrong clusters. Therefore, we implemented a very easy way to re-group some of the wrong clusters. First of all, users can adapt the system threshold with a simple slider to manipulate the cluster granularity. Figure 5.8 shows that clusters can additionally be split by cutting them with a simple stroke gesture. By drawing a line gesture that intersects two or more clusters, multiple clusters can be merged into one larger cluster. All changes that will happen are shown in a real-time preview manner; this provides the user with the ability to undo a split or merge action by tracing back along the gesture-stroke. Moreover, it is possible to cancel an entire adaption by performing a zig-zag gesture.

## 5.4. Evaluation

We evaluated our clustering method based on the seven trained perspectives of the background study. To train these perspectives, we used an annotated *training dataset*. This was created from the initial sketches we gathered, excluding the 15 sketches we had used in the background study. These 15 sketches were used as a *test dataset* for evaluation. The two sets were distinct and disjoint to ensure there was no bias from a machine learning point of view.

Each sketch from the training dataset was annotated under all of the applicable perspectives. As a result, our training set consisted of 84 sketches for *Visual Subgroups*, 40 for *Structural Similarity*, 57 for *Objects*, 30 for *Visual Text Units*, 30 for *Columns*, 49 for *Hierarchies*, and 24 for *Rows*. The test dataset was then used to compare the results of our algorithm to the perspectives indicated by the participants in the study. Summarizing, we asked the users about their expectations, trained the system on different data, and then checked if it then met the expectations. Note that for each sketch in the test dataset, we picked the two most commonly occurring user perspectives from the study to use for ground truth annotation. In two cases, the most common

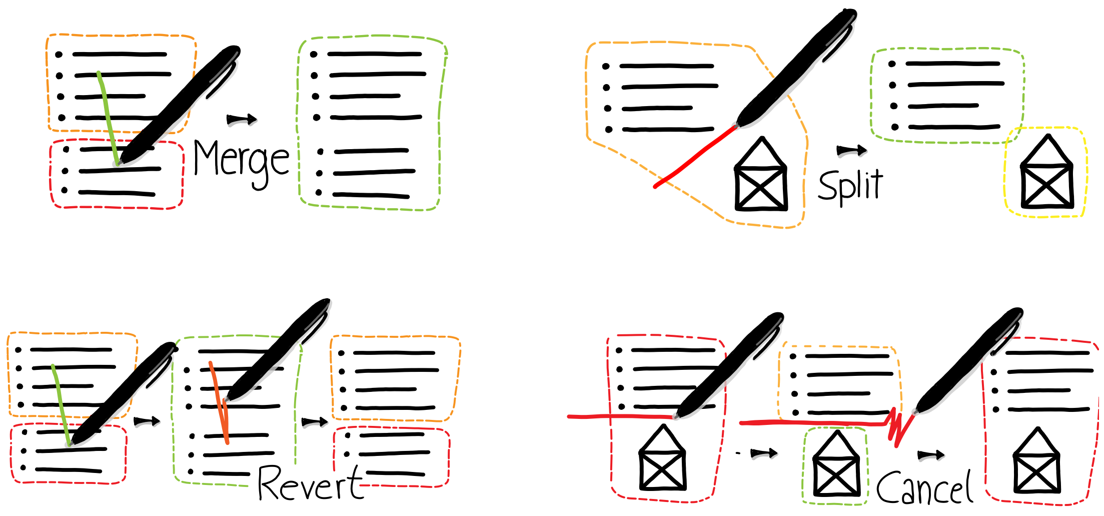


**Figure 5.8.** Connecting two or more clusters with a single stroke merges them. Cutting through a cluster with a stroke splits the cluster along the line.

expectations included just one cluster containing all strokes. We did not consider these trivial cases. Thus, we obtained a test dataset containing 28 differently annotated sketches. Since the result of the clustering strongly depends on the initial cluster, we let an expert user perform three different initial selections, which he considered to be reasonable. The initial selection always matched one of the annotated objects. In general, a good initial selection should have a reasonable complexity and should be close to other expected clusters to define informative $C$- and $M$-links for parameter adaptation. In a few cases, there were not enough annotated objects to create three different initial selections and thus only two were performed. In total, we have analysed our method using 82 clustering trials.

Once the best linear combination of the perspectives was learnt from the initial selection and the clustering was done, we counted how many corrective steps were necessary to obtain the expected result. This is our measure for the quality of the result. In an ideal case, the result matches the expectation and no further steps are required. Figure 5.9 (top) shows an ideal example, where the expert initially selected one column and the system recognized the remaining four columns. In the bottom case, where one bullet entry was selected, the user interaction was required to achieve the expected result. Three possibilities exist to improve an imperfect result: a) change the distance threshold, b) split a cluster, and c) merge two clusters. A low number of required steps (e.g. one or two) is considered to be a success since the user can achieve the expected result with minimal effort.

Figure 5.10 shows the number of required steps that were necessary to get the final result. In 44 % of all trials the desired result was achieved with zero or max. one modification. In 22 % of all trials the desired result was achieved with two or three modifications and in 24 % of all trials more than 7 modifications were required. On average, the clustered sketches needed 2.76 ($SD = 2.70$) steps to achieve the expected result, where the sketches contained on average 13.71 ($SD = 10.02$) annotated objects.

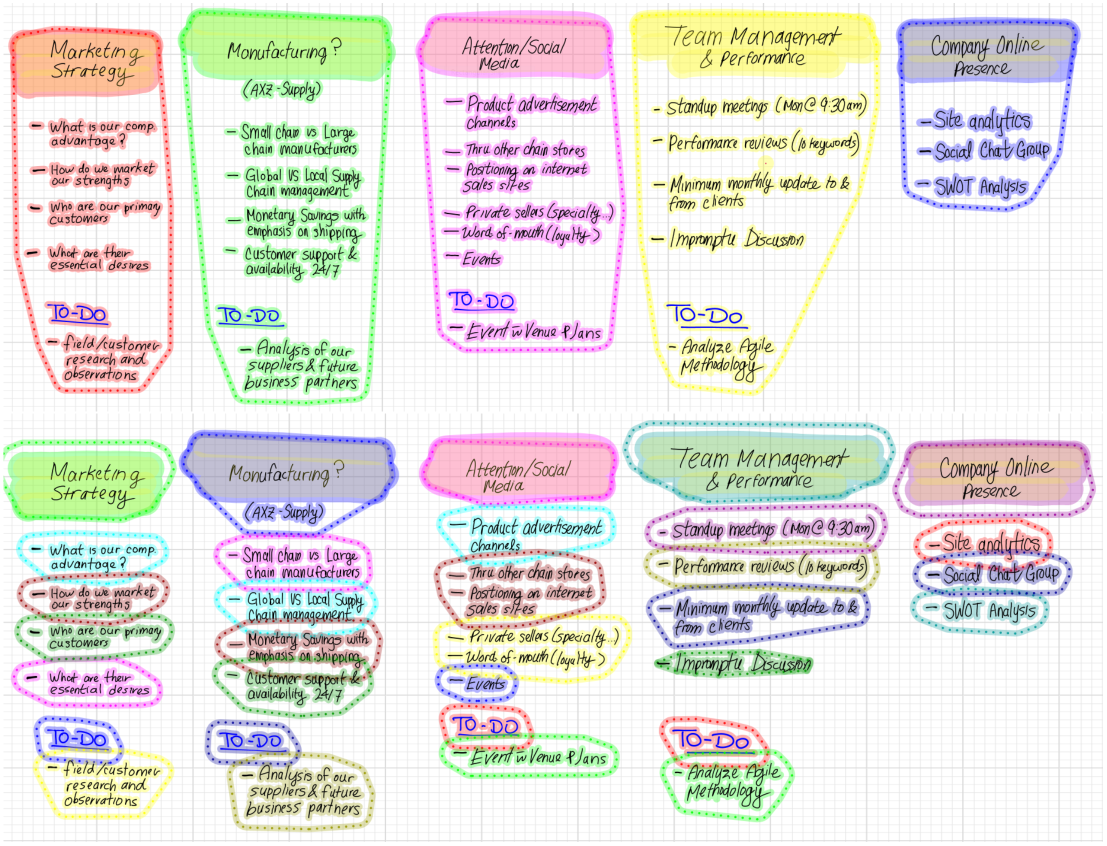

Marketing Strategy
- What is our comp advantage?
- How do we market our strengths
- Who are our primary customers
- What are their essential desires
TO-DO
- field/customer research and observations
Manufacturing?
(AXZ-Supply)
- Small chain vs Large chain manufacturers
- Global vs Local Supply Chain management
- Monetary Savings with emphasis on shipping
- Customer support & availability 24/7
TO-DO
- Analysis of our suppliers & future business partners
Attention/Social Media
- Product advertisement channels
- Thru other chain stores
- Positioning on internet sales sites
- Private sellers (specialty...)
- Word-of-mouth (loyalty)
- Events
TO-DO
- Event w Venue Plans
Team Management & Performance
- Standup meetings (Mon @ 9:30am)
- Performance reviews (to keywords)
- Minimum monthly update to & from clients
- Impromptu Discussion
TO-DO
- Analyze Agile Methodology
Company Online Presence
- Site analytics
- Social Chat Group
- SWOT Analysis

**Figure 5.9.** Top: An example that did not require any user correction. Bottom: Most entries were correctly recognized, the few remaining ones can be fixed using the Split tool.

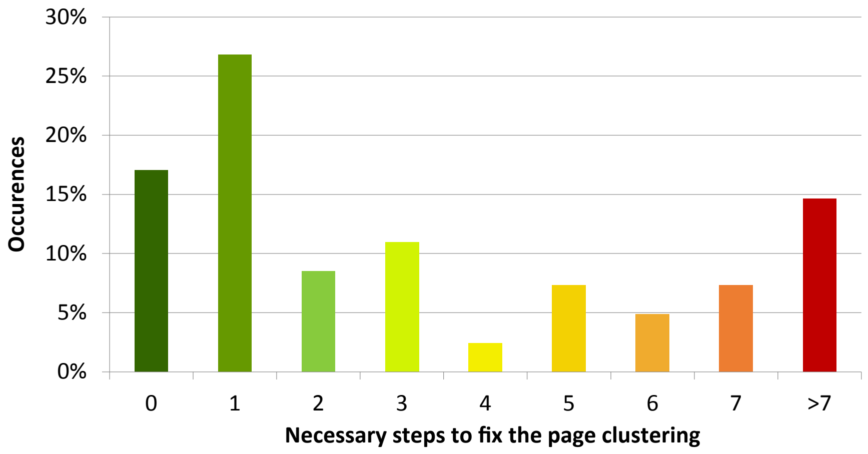


**Figure 5.10.** Number of steps that are required to achieve the expected result.

## 5.5. Preliminary Field Test

In the course of an overarching research project, cLuster was deployed within a tablet sketching application and was tested by two industrial design companies and one academic design institute. The designers considered the clustering feature to be particularly useful for visual note-taking and conceptual sketching tasks. For visual note-taking during meetings, they found it helpful for quickly rearranging sketched objects and words. This made note taking a less linear activity and enabled the creation of more concise notes. We were surprised by how they used the tool to retroactively disassemble sketches during conceptual sketching tasks. They found it helpful since they no longer needed to consider *layers* while creating content.

## 5.6. Applications and Interaction Techniques

In this section, we cover some of the interactive affordances of cLuster and present several application scenarios that show the utility and the performance of our algorithm in practice.

### 5.6.1. Advanced Crossing Selection

Obviously, our approach performs best once the user wants to cluster an entire page with an initial cluster selection as a template. However, it can also be used to improve the performance of crossing-based selection tools [Accot and Zhai, 2002] by suggesting possible selections. Figure 5.11 depicts this approach in action; while crossing a stroke, the system invokes a clustering process and suggests a selection to the user. The predicted selection is then visualized as a surrounding hull. A small icon that follows the users' pen path can be used to trigger the suggested selection.

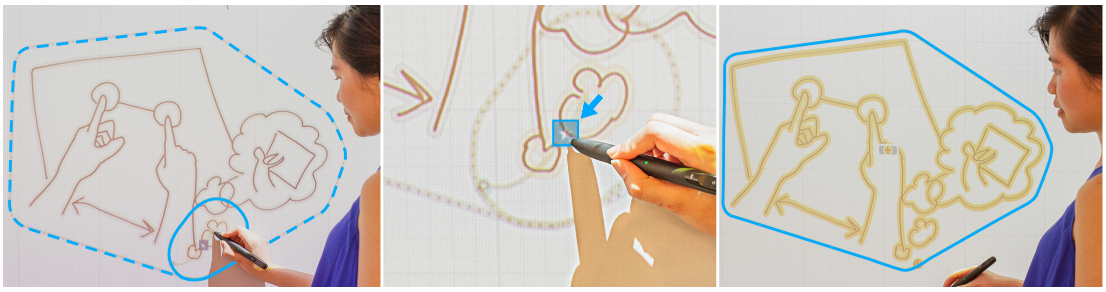


**Figure 5.11.** Advanced Crossing Selection: The first few selected strokes are used to cluster the rest of the page and anticipate a likely selection. The user can confirm the proposed cluster by lifting the pen up on the icon following the crossing path behind the pen.

### 5.6.2. Advanced Tapping Selection

While tapping is well suited for single targets, it is cumbersome for large groups [Mizobuchi and Yasumura, 2004]. Nowadays, many applications (e.g. text editors) provide the feature that enables users to quickly select a word by performing a double-click, while three clicks in a row selects the entire paragraph. With cLuster, we can achieve similar results in a sketching scenario. A single tap selects one cluster, while a second tap adds the closest cluster to the selection (see Figure 5.12). After merging two clusters this way, the resulting cluster is used to recalibrate the clustering algorithm. By performing a directional flick gesture, the user can also choose the direction in which the next potential cluster should be found. After merging multiple clusters into one, this selection can be further used.

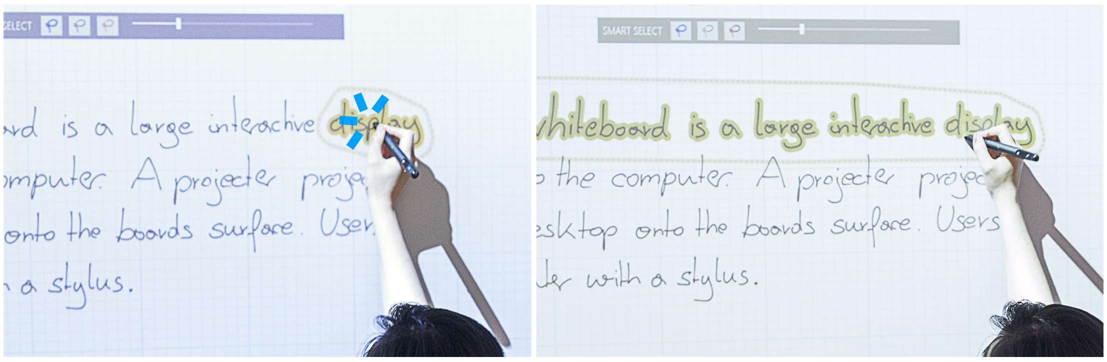

SMART SELECT
whiteboard is a large interactive display
the computer. A projector projec
esktop onto the boards surface. Users
der with a stylus.

**Figure 5.12.** A tap selects a small unit, such as a single word. Directional flicks add nearby clusters to the selection.

### 5.6.3. Advanced Rearrangement

Extensive re-arrangement is a domain where our approach really shines. Clustering the entire page at once eliminates the need to repetitively select strokes when translating groups.

All clusters are at hand and can be directly moved or altered. This reduces the effort needed for structuring and rearrangement tasks immensely. However, in many cases users want to move clusters to a position that is already occupied by existing content. Knowing the placement of all the clusters, they can be moved out of the way to make space, so that no temporary space is required. Initially, we used a coalition based approach, similar to [Mynatt et al., 1999]. However, we soon realized that the behavior was not ideal in many cases. Therefore, we investigated several options of how other identified clusters could make space. Of course, each strategy has its pros and cons and is very dependent on the type of sketches being manipulated. In the following, we present two possible strategies:

**Spring Strategy** One basic arrangement strategy simulates a spring's behaviour. The general rule is that clusters try to remain as close as possible to their original position. This is a very standard approach that can be used for a wide variety of applications.

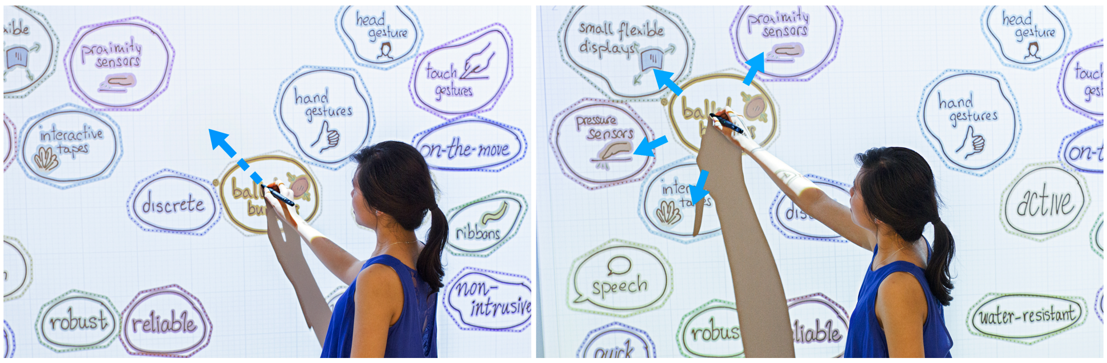


**Figure 5.13.** With the Spring Strategy, inactive clusters *make way* for incoming active clusters.

**Bubble Strategy** Working with well organized sketches, such as lists or storyboards, the Bubble Strategy is very effective. If a cluster is moved through a list, the other clusters swap their positions with the current selection being moved. This creates a bubble effect that enables easy exchange of clusters without changing the overall list and without needing to rearrange a large number of list items when an entry is relocated within the list.

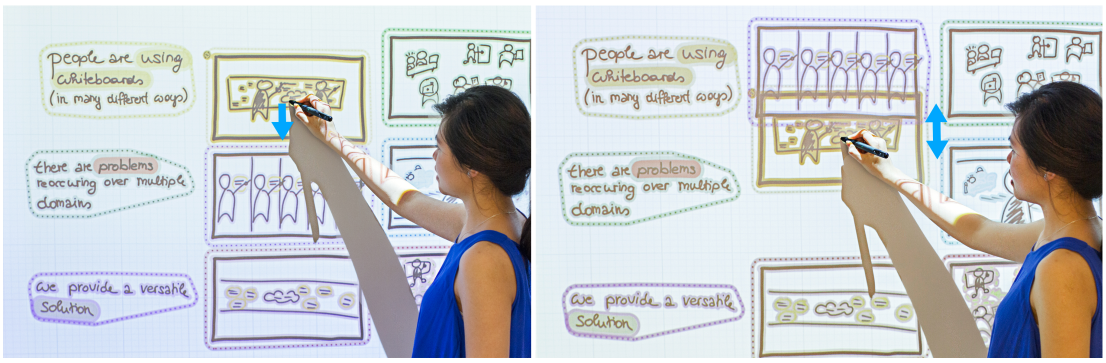

People are using whiteboards (in many different ways)
there are problems reoccuring over multiple domains
we provide a versatile solution

**Figure 5.14.** The Bubble Strategy makes clusters exchange place as soon as their hulls intersect.

### 5.6.4. Advanced Copy and Paste Across Applications

As highlighted in the introduction, there are two major types of collaborative content creation applications on large interactive surfaces. Besides the sketching applications that mimic traditional whiteboards and serve as a non-restrictive and flexible tool for sketching and writing, there are other applications based on the concept of sticky notes that provide a means to quickly rearrange, structure, and cluster ideas. Based on cLuster, we can now quickly cluster objects in freehand sketches and transfer these with one click to another application (see Figure 5.15).

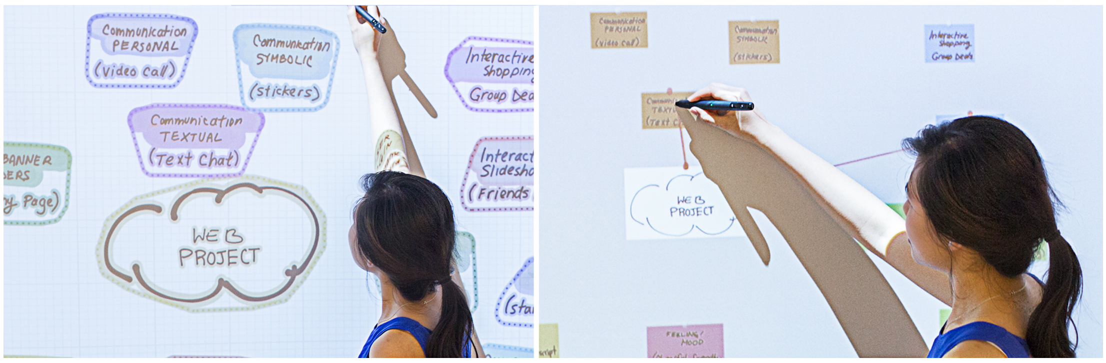

Communication PERSONAL (video call)
Communication SYMBOLIC (stickers)
Communication TEXTUAL (Text Chat)
WEB PROJECT

**Figure 5.15.** Clustering content enables users to port sketched content to object-based applications as part of more complex workflows.

## 5.7. Discussion and Limitations

The insights we gained from analyzing the clusters given by the participants in our background study showed that the expectations of what should be clustered are quite diverse.

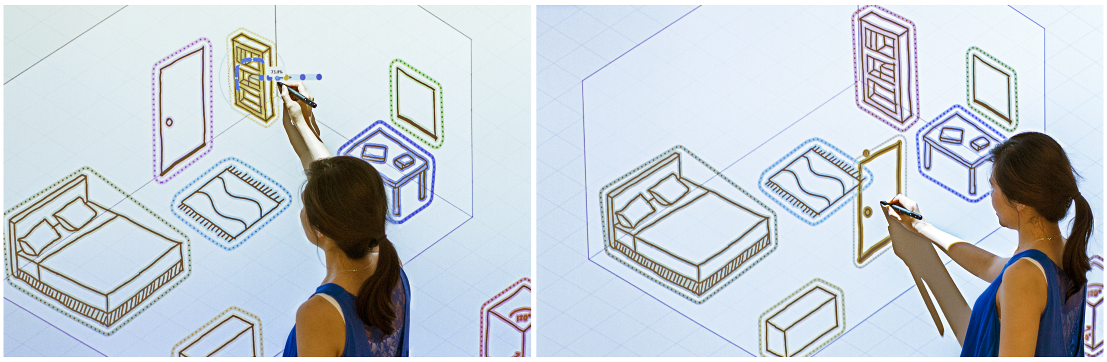


**Figure 5.16.** In an interior design context cLuster can substitute tracing paper. Elements do not have to be sketched on different layers but can be separated afterwards.

Choosing appropriate perspectives as well as the right features can be quite challenging when designing a clustering tool, which should be capable of clustering strokes in several different ways. During our experiments, it turned out that it is reasonable to keep both the perspectives and the features as general as possible. The perspectives should be diverse to cover various scenarios, but also need to generalize well to work for different types of strokes (text, figures). Defining too many perspectives can cause an over-fitting of the system. Similarly, the used features should describe very general characteristics. Amongst others, we experimented with additional features that describe the difference using the colour and thickness of strokes. These attributes are often used for headlines and hierarchies and we were optimistic to improve the separation of headers from the rest of the paragraphs. While the features helped in this very particular case, they spoiled the behaviour in many other cases, e.g. when selecting an initial cluster where the strokes were highlighted (crossed with a stroke with very high thickness and transparent colour). In this case, the thickness feature became too dominant and the importance of other features degraded. As a result, all non-highlighted clusters fell apart. Therefore, we did not use these features in the final version of the implementation.

The benefit of our clustering system is definitely the fast pace, in which it can cluster an entire page of strokes. While it can also be used to improve the process of single cluster selection, as we have shown with the *Advanced Crossing* and *Advanced Tapping* example, the real advantage is in the overall clustering. First and foremost, this removes the need for repetitive selections and thus enables fast reordering and structuring of content. This helps to users to maintain their flow of thoughts and stay focused on their main tasks. The automatic page clustering provides a promising alternative to manual grouping and/or layering as is done in ordinary sketch applications. Since clustering is applied afterwards, our approach does not distract users during the content creation process. This thought was also confirmed by the industrial designers who tested our prototype. Especially when creating perspective drawings of interior design concepts (cf. Figure 5.16), the clustering algorithm does well in dividing up the different elements within a room. This makes rearranging the elements in the room simple, and enables designers to freely explore a variety of configurations without having to consciously consider *layers*.

In a next step, knowing all clusters in a sketch can be used to define interesting schemes for interaction between manipulated objects. We explored this concept by implementing different *Rearrangement Strategies* and found that it is very valuable for manipulation, e.g. of entries in hierarchical lists.

A strength of our tool is that it can adjust its way of clustering when given a new initial selection template, without needing its weights to be retrained. This makes the approach very flexible, while it remains simple to use. As mentioned in the evaluation, the selection of the initial cluster is critical for the system. It always tries to find a combination of the perspectives that preserves this cluster during clustering. However, there can be constellations that cannot be resolved (e.g. when some strokes are too close to the initial cluster, or there are big gaps in the initial cluster) resulting in the initial cluster breaking apart or merging with other strokes. In general, problems occur when users decide to initialize the clustering with an initial selection that is far away from every possibility the perspectives provide.

In the analysis, we have seen that our current approach failed for two sketches that had a table structure and included structural elements (lines). When trying to cluster content in cells, it often happened that nearby structural lines were included in clusters. Consequently, other distant strokes were then also added to these clusters. The exceptional position of structural lines was also confirmed in our background study. Many participants felt that they should not be part of the grouping, but rather be considered as external elements. However, currently our approach is not capable of distinguishing structural components from content. Classification of clusters is needed to solve this issue, which is beyond the scope of this work and is left for future research.

# 6. Conclusions

This thesis contributed to understanding of sketches drawn on-line on an ink-input device. It was devoted to exploit the structure of the sketch during its recognition. Two related topics are covered: recognition of formal diagrams and detection of objects in free-form sketches. This final chapter summarizes the developments in both topics and conclusions of this thesis. We also discuss possible future research directions.

We developed the *diagram recognizer* suitable for recognition of formal arrow-connected diagrams. Although there exist many domains of such diagrams, we have chosen two of the most common ones: flowcharts and finite automata. The system was thus adapted for and tested on these two domains. Our implementation has achieved the state-of-the-art results in comparison with other approaches. The recognition process follows the designed recognition pipeline. We contributed in a certain degree to each of the major steps of the recognition pipeline: text/non-text separation, symbol segmentation, symbol classification/detection, structural analysis. The system can be adapted for another diagrammatic domains with arrow-connected structure with only minor adjustments of some steps of the pipeline. The most of the adaptation work runs on its own when a new training dataset is available and consists mostly of retraining the classifiers. A summary of our contribution to the recognition pipeline follows:

- We showed that it is advantageous to perform *text/non-text separation* beforehand the recognition of the diagram structure. The text can be formed into logical blocks and recognized afterwards. It is much easier because the already known structure guides this process. It decomposes the problem and makes the recognition easier and faster. To do that, we used the state-of-the-art text/non-text classifier and made it biased to make smaller error in the non-text class.

- We observed that the recognition of arrows with a varying shape is a difficult task for appearance-based classifiers. Therefore, we created a novel *arrow detector*. It exploits the distinctive property of arrows – they connect two uniform symbols. Therefore, it searches for arbitrarily shaped connectors linking two symbols together. It works in a two-stage symbol recognition manner. Uniform symbols are detected first, because it is much easier with their stable appearance. Arrows are detected afterwards in two steps: a) detection of the shaft, b) detection of the head.

- With text removed and arrows being detected later in the second stage of the symbol recognition, only the uniform symbols need to be segmented in the segmentation step. This makes the task significantly easier. First, we employed a common stroke grouping based on a spatio-temporal proximity to perform over-segmentation. Later, we replaced this simple approach by a more sophisticated *over-segmentation based on hierarchical clustering* using a trainable distance function. We showed that it can achieve a significantly higher precision while the recall stays almost unchanged. It makes the whole recognition system faster, because a smaller number of clusters has to be concerned in further steps of the pipeline.

- We proposed a model for recognition by selection of symbol candidates, based on the evaluation of relations between candidates using a set of predicates. The *structural* *analysis* finding the optimal combination of candidates is cast as a *max-sum labeling problem.* It proved to be a better approach for simple structures than the grammar-based structural analysis, which seems to be slow an impractical in this case. We showed that the instances generated during the diagram recognition are small enough to be solvable by general solvers in real-time despite the fact that the max-sum problem is NP-hard.

Despite the fact that handwriting recognition is a very popular field of research with a lot of progress and attention in various domains like recognition of mathematical expressions, only little has been done in diagram domains. There was a lack of high quality benchmark database. Therefore, we collected and annotated our own *benchmark database* and made it publicly available. It helped us to develop and test our diagram recognizer. Furthermore, it will allow to reliably compare results of different approaches. We believe that it will likely attract attention of other researchers to the topic. Hopefully, we will see a competition on recognition of on-line handwritten diagrams soon.

The second topic covered in this thesis is object segmentation in free-form sketches. Segmentation by classification is a common approach applied in formalized domains. Unfortunately, it cannot be used in the case of free-form sketches, because there are no symbol classes to perform the classification or they are not known in advance. An alternative approach must be employed. We explored possibility to use a *cluster analysis.* Here we made three contributions. First, we conducted a *study* in order to find out what people see as individual objects in free-form sketches. It turned out that there are usually more valid ways how to segment one sketch. We identified seven major points of view how to perform the segmentation we call *perspectives.* Each perspective is de facto a generalized structure. Second, we showed that SLAC with trainable distance function can be trained separately for each of the defined perspectives and the desired final segmentation can be done by simple linear combination of distance functions from individual perspectives. Third, we designed a *user interface* allowing user to make an *initial selection* to indicate his intention. The tool combines automatically the trained perspectives according to one manually segmented object in such way, that the rest of the sketch is segmented into an object similar to the initial one. The user interface also contains tools for interaction with the created clusters. The user can thus adjust seamlessly the clusters with merge/split gestures or rearrange the objects.

Finally, we identified the following future research directions regarding diagram recognition and free-from sketching:

- The structural analysis and partially even other steps of the recognition pipeline have a combinatorial characteristics. Its time complexity implies certain limits to the scalability of the system. Therefore, it is reasonable to experiment with *iterative recognition* using immediate user's feedback on intermediate results. The problem can be decomposed this way efficiently to reduce the recognition time significantly. Moreover, it is easier for the user to recover the recognition from an error. We believe that the proposed system is capable of such adaptation. Typical applications of the diagram recognizer allow such iterative approach. It could be more attractive for the user with a proper user interface.

- As we stated earlier, the *adaptation* of the proposed recognizer to other diagrammatic domains should be straightforward. It makes sense to cooperate with diagram users professionally. It would be interesting to see the usage of diagrams during a creative process of sharing fresh ideas among cooperating people. This feedback should identify the most important domains and use cases. Professional diagram users could point out important new notations for which some further adjustments in the recognition pipeline could be developed. Another possibility is to extend it to support domains beyond the scope of arrow-connected diagrams. This extension would require a modification of the max-sum model. It is possible if the fundamental relation between arrows and symbols can be replaced by another relation defining the structure of the handwriting. The example is the relation between notes and staff lines in the case of music scores.

- Another possibility is to adapt the system for *off-line recognition*. In this case, the input is an image with a diagram. The recognizer faces different problems. The segmentation step becomes more challenging because any time information is missing. However, the structural analysis might remain the same. The typical application of such recognizer is different. Users would probably use it to digitize some old diagrams written on a paper. Therefore, the input image can be a scan or a photo of the document with the diagram.

- The proposed clustering method for finding objects of interest in free-form sketches might use very different features. For example, it might be beneficial to exploit a colour of the stroke. Unfortunately, different users tend to use colours for completely different purposes. Therefore, a different colour may imply that two strokes do not belong together in one case while in another case the colour imply the opposite or play no role at all. It can easily happen that some features can bring more confusion than useful information. One option could be a *dynamic feature selection*. It would be interesting to see if some features could be "deactivated" in dependence on other features. Another possibility is to design a user interface which would allow better indication of users intentions to achieve this feature selection.

# List of Acronyms

| | |
|---|---|
| BLSTM | Bidirectional Long Short-Term Memory |
| CRF | Conditional Random Fields |
| DMOS | Description and MOdification of the Segmentation |
| EPF | Enhanced Position Formalism |
| ER | Entity Relationship |
| FA | Finite Automaton |
| FC | Flowchart |
| FSM | Finite State Machine |
| HMM | Hidden Markov Model |
| ILP | Integer Linear Programming |
| InkML | Ink Markup Language |
| LSTM | Long Short-Term Memory |
| MAP | Maximum a posteriori Probability |
| RNN | Recurrent Neural Network |
| SD | Standard Deviation |
| SLAC | Single-Linkage Agglomerative Clustering |
| SVM | Support Vector Machines |
| UML | Unified Modeling Language |
| XML | Extensible Markup Language |

# List of Figures

# List of Tables

# List of Algorithms

# Bibliography

Accot, J. and Zhai, S. (2002). More Than Dotting the I's — Foundations for Crossing-based Interfaces. In *CHI '02: Proceedings of the Conference on Human Factors in Computing Systems*, pages 73–80, New York, NY, USA. ACM. 80

Alvarado, C. and Davis, R. (2004). SketchREAD: A multi-domain sketch recognition engine. In *UIST '04: Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, pages 23–32, New York, NY, USA. ACM. 10

Alvarado, C. and Lazzareschi, M. (2007). Properties of real-world digital logic diagrams. In *PLT 07': Proceedings of the First International Workshop on Pen-Based Learning Technologies*, pages 1–6. 10

Álvaro, F., Sánchez, J.-A., and Benedí, J.-M. (2014). Recognition of on-line handwritten mathematical expressions using 2d stochastic context-free grammars and hidden Markov models. *Pattern Recognition Letters*, 35(0):58 – 67. 8, 35, 67

Andreson, R. H. (1968). Syntax-directed recognition of hand-printed two-dimensional mathematics. In *Interactive Systems for Experimental Applied Mathematics*, pages 436–459, London, U.K. Academic Press. 8

Arvo, J. and Novins, K. (2005). Appearance-preserving manipulation of hand-drawn graphs. In *GRAPHITE '05: Proceedings of the 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, pages 61–68. ACM. 12

Awal, A.-M., Feng, G., Mouchere, H., and Viard-Gaudin, C. (2011). First experiments on a new online handwritten flowchart database. In *DRR '11: Proceedings of the 18th International Conference on Document Recognition and Retrieval*, pages 1–10. 16, 17

Belaid, A. and Haton, J.-P. (1984). A syntactic approach for handwritten mathematical formula recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(1):105–111. 8

Bresler, M., Průša, D., and Hlaváč, V. (2015a). Detection of arrows in on-line sketched diagrams using relative stroke positioning. In *WACV '15: IEEE Winter Conference on Applications of Computer Vision*, pages 610–617. IEEE Computer Society. 14, 42, 49

Bresler, M., Průša, D., and Hlaváč, V. (2013). Modeling flowchart structure recognition as a max-sum problem. In O'Conner, L., editor, *ICDAR '13: Proceedings of the 12th International Conference on Document Analysis and Recognition*, pages 1247–1251. IEEE Computer Society. 14, 41, 67, 109

Bresler, M., Průša, D., and Hlaváč, V. (2015b). Using agglomerative clustering of strokes to perform symbols over-segmentation within a diagram recognition system. In Paul Wohlhart, V. L., editor, *CVWW '15: Proceedings of the 20th Computer Vision Winter Workshop*, pages 67–74. Graz University of Technology. 35

Bresler, M., Průša, D., and Hlaváč, V. (2016). On-line recognition of sketched arrow-connected diagrams. *International Journal on Document Analysis and Recognition (IJDAR)*. Accepted for publication on May 1, 2016. 14, 58

Bresler, M., Van Phan, T., Průša, D., Nakagawa, M., and Hlaváč, V. (2014). Recognition system for on-line sketched diagrams. In Guerrero, J. E., editor, *ICFHR '14: Proceedings of the 14th International Conference on Frontiers in Handwriting Recognition*, pages 563–568. IEEE Computer Society. 14, 35, 40, 109

Carton, C., Lemaitre, A., and Couasnon, B. (2013). Fusion of statistical and structural information for flowchart recognition. In *ICDAR '13: Proceedings of the 12th International Conference on Document Analysis and Recognition*, pages 1210–1214. 10, 58

Chan, K.-F. and Yeung, D.-Y. (2000). Mathematical expression recognition: a survey. *International Journal on Document Analysis and Recognition (IJDAR)*, 3(1):3–15. 8

Dehmeshki, H. and Stürzlinger, W. (2009). GPSel: A Gestural Perceptual-Based Path Selection Technique. In *SG '09: Proceedings of the 10th International Symposium on Smart Graphics*, pages 243–252. Springer. 68

Dehmeshki, H. and Stürzlinger, W. (2010). Design and Evaluation of a Perceptual-based Object Group Selection Technique. In *BCS '10: British Computer Society Conference on Human-Computer Interaction*, pages 365–373, Swinton, UK, UK. British Computer Society. 68

Delaye, A. (2014). Structured prediction models for online sketch recognition. Unpublished manuscript, `https://sites.google.com/site/adriendelaye/home/news/unpublishedmanuscriptavailable`. 11, 58

Delaye, A. and Anquetil, E. (2013). HBF49 feature set: A first unified baseline for online symbol recognition. *Pattern Recognition*, 46(1):117–130. 39

Delaye, A. and Lee, K. (2015). A flexible framework for online document segmentation by pairwise stroke distance learning. *Pattern Recognition*, 48(4):1197–1210. 35, 36, 67, 72, 73, 75

Delaye, A. and Liu, C.-L. (2014). Contextual text/non-text stroke classification in online handwritten notes with conditional random fields. *Pattern Recognition*, 47(3):959–968. 32

El Meseery, M., El Din, M., Mashali, S., Fayek, M., and Darwish, N. (2009). Sketch recognition using particle swarm algorithms. In *ICIP 2009: Proceedings of the 16th International Conference on Image Processing*, pages 2017–2020. IEEE Computer Society. 46

Feng, G. and Viard-Gaudin, C. (2008). Stroke fragmentation based on geometry features and HMM. *CoRR: Computing Research Repository*. 46

Feng, G., Viard-Gaudin, C., and Sun, Z. (2009). On-line hand-drawn electric circuit diagram recognition using 2D dynamic programming. *Pattern Recogn.*, 42(12):3215–3223. 9, 35, 67

Freeman, I. J. and Plimmer, B. (2007). Connector semantics for sketched diagram recognition. In *AUIC '07: Proceedings of the Eight Australasian Conference on User Interface - Volume 64*, pages 71–78, Darlinghurst, Australia, Australia. Australian Computer Society, Inc. 10

Geyer, F., Pfeil, U., Budzinski, J., Höchtl, A., and Reiterer, H. (2011). Affinitytable - a Hybrid Surface for Supporting Affinity Diagramming. In *Interact '11: Proceedings of the 13th IFIP TC13 Conference on Human-Computer Interaction*, pages 477–484, Berlin, Heidelberg. Springer. 68, 69

Glaser, B. G. and Strauss, A. L. (1967). *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine de Gruyter, New York, NY. 72

Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional {LSTM} and other neural network architectures. *Neural Networks*, 18(5–6):602 – 610. 48

Grossman, T., Baudisch, P., and Hinckley, K. (2009). Handle Flags: Efficient and Flexible Selections for Inking Applications. In *GI '09: Proceedings of the 35th International Conference on Graphics Interface*, pages 167–174, Toronto, Ont., Canada, Canada. Canadian Information Processing Society. 68, 69

Guimbretière, F., Stone, M., and Winograd, T. (2001). Fluid Interaction with High-resolution Wall-size Displays. In *UIST '01: Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*, pages 21–30, New York, NY, USA. ACM. 68, 69

Hailpern, J., Hinterbichler, E., Leppert, C., Cook, D., and Bailey, B. P. (2007). TEAM STORM: Demonstrating an Interaction Model for Working with Multiple Ideas During Creative Group Work. In *C&C '07: Proceedings of the 6th Creativity & Cognition Conference*, pages 193–202, New York, NY, USA. ACM. 69

Haller, M., Leitner, J., Seifried, T., Wallace, J. R., Scott, S. D., Richter, C., Brandl, P., Gokcezade, A., and Hunter, S. (2010). The NiCE discussion room: Integrating Paper and Digital Media to Support Co-Located Group Meetings. In *CHI '10: Proceedings of the Conference on Human Factors in Computing Systems*, pages 609–618, New York, New York, USA. ACM. 68

Hammond, T. and Davis, R. (2005). LADDER, a sketching language for user interface developers. In *Computers & Graphics*, volume 29, pages 518–532, New York, NY, USA. Elsevier. 10

Herold, J. and Stahovich, T. F. (2011). Classyseg: A machine learning approach to automatic stroke segmentation. In *SBIM '11: Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*, pages 109–116. 46

Hilliges, O., Terrenghi, L., Boring, S., Kim, D., Richter, H., and Butz, A. (2007). Designing for collaborative creative problem solving. In *C&C '07: Proceedings of the 6th Creativity & Cognition Conference*, pages 137–146, New York, New York, USA. ACM. 68, 69

Igarashi, T., Matsuoka, S., and Masui, T. (1995). Adaptive recognition of implicit structures in human-organized layouts. In *VL '95: Proceedings of the 6th Symposium on Visual Languages and Human Centric Computing*, pages 258–266. IEEE. 68

Indermühle, E., Frinken, V., and Bunke, H. (2012). Mode detection in online handwritten documents using BLSTM neural networks. In *ICFHR '12: Proceedings of the 13th International Conference on Frontiers in Handwriting Recognition*, pages 302–307. 32, 48

Kara, L. B. and Stahovich, T. F. (2004). Hierarchical parsing and recognition of hand-sketched diagrams. In *UIST '04: Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, pages 13–22. ACM. 9, 41

Lank, E. and Saund, E. (2005). Sloppy selection: Providing an accurate interpretation of imprecise selection gestures. *Computers & Graphics*, 29(4):490–500. 68, 69

Le, A. D., Van Phan, T., and Nakagawa, M. (2014). A system for recognizing on-line handwritten mathematical expressions and improvement of structure analysis. In *DAS '14: Proceedings of the 11th IAPR International Workshop on Document Analysis Systems*, pages 51–55. 8

Lemaitre, A., Mouchére, H., Camillerapp, J., and Coüasnon, B. (2011). Interest of syntactic knowledge for on-line flowchart recognition. In *GREC '11: Proceedings of the 9th IAPR International Workshop on Graphics Recognition*, pages 85–88. 10

Lindlbauer, D., Haller, M., Hancock, M., Scott, S. D., and Stuerzlinger, W. (2013). Perceptual Grouping: Selection Assistance for Digital Sketching. In *ITS '13: Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces*, pages 51–60, New York, NY, USA. ACM. 68

Liu, C.-L. and Zhou, X.-D. (2006). Online Japanese Character Recognition Using Trajectory-Based Normalization and Direction Feature Extraction. In Lorette, G., editor, *IWFHR 06': Proceedings of the 10th International Workshop on Frontiers in Handwriting Recognition*. Université de Rennes 1, Suvisoft. 39

Liwicki, M. and Knipping, L. (2005). Recognizing and simulating sketched logic circuits. In Khosla, R., Howlett, R., and Jain, L., editors, *Knowledge-Based Intelligent Information and Engineering Systems*, volume 3683 of *Lecture Notes in Computer Science*, pages 588–594. Springer Berlin Heidelberg. 10

Mankoff, J., Hudson, S. E., and Abowd, G. D. (2000a). Interaction techniques for ambiguity resolution in recognition-based interfaces. In *UIST '00: Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, pages 11–20, New York, New York, USA. ACM. 77

Mankoff, J., Hudson, S. E., and Abowd, G. D. (2000b). Providing integrated toolkit-level support for ambiguity in recognition-based interfaces. In *CHI '00: Proceedings of the Conference on Human Factors in Computing Systems*, pages 368–375, New York, New York, USA. ACM. 77

Martín-Albo, D., Plamondon, R., and Vidal, E. (2014). Training of on-line handwriting text recognizers with synthetic text generated using the kinematic theory of rapid human movements. In Guerrero, J. E., editor, *ICFHR '14: Proceedings of the 14th International Conference on Frontiers in Handwriting Recognition*, pages 543–548. IEEE Computer Society. 40

Miyao, H. and Maruyama, R. (2012). On-line handwritten flowchart recognition, beautification and editing system. In *ICFHR '12: Proceedings of the 13th International Conference on Frontiers in Handwriting Recognition*, pages 83–88. 11

Mizobuchi, S. and Yasumura, M. (2004). Tapping vs. Circling Selections on Pen-based Devices: Evidence for Different Performance-shaping Factors. In *CHI '04: Proceedings of the Conference on Human Factors in Computing Systems*, pages 607–614, New York, NY, USA. ACM. 80

Moran, T. P., Chiu, P., and van Melle, W. (1997). Pen-based Interaction Techniques for Organizing Material on an Electronic Whiteboard. In *UIST '97: Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology*, pages 45–54, New York, NY, USA. ACM. 69

Mouchère, H., Viard-Gaudin, C., Zanibbi, R., and Garain, U. (2014). ICFHR 2014 Competition on Recognition of On-line Handwritten Mathematical Expressions (CROHME 2014). In Guerrero, J. E., editor, *ICFHR '14: Proceedings of the 14th International Conference on Frontiers in Handwriting Recognition*, pages 791–796. IEEE Computer Society. 8

Mynatt, E. D., Igarashi, T., Edwards, W. K., and LaMarca, A. (1999). Flatland: New Dimensions in Office Whiteboards. In *CHI '99: Proceedings of the Conference on Human Factors in Computing Systems*, pages 346–353, New York, NY, USA. ACM. 68, 69, 81

Noris, G., Sýkora, D., Shamir, A., Coros, S., Whited, B., Simmons, M., Hornung, A., Gross, M., and Sumner, R. (2012). Smart Scribbles for Sketch Segmentation. *Computer Graphics Forum*, 31(8):2516–2527. 69

Otte, S., Krechel, D., and Liwicki, M. (2013). JANNLab Neural Network Framework for Java. In *MLDM '13: Proceedings of the 9th International Conference on Machine Learning and Data Mining*, pages 39–46. 49

Otte, S., Krechel, D., Liwicki, M., and Dengel, A. (2012). Local feature based online mode detection with recurrent neural networks. In *ICFHR '12: Proceedings of the 13th International Conference on Frontiers in Handwriting Recognition*, pages 531–535. 32, 48

Ouyang, T. Y. and Davis, R. (2011). Chemink: A natural real-time recognition system for chemical drawings. In *IUI '11: Proceedings of the 16th International Conference on Intelligent User Interfaces*, pages 267–276. ACM. 9, 35, 67

Pedersen, E. R. n., McCall, K., Moran, T. P., and Halasz, F. G. (1993). Tivoli: integrating structured domain objects into a freeform whiteboard environment. In *CHI '93: Proceedings of the Conference on Human Factors in Computing Systems*, pages 391–398, New York, New York, USA. ACM. 69

Perteneder, F., Bresler, M., Grossauer, E.-M., Leong, J., and Haller, M. (2015). cluster: Smart clustering of free-hand sketches on large interactive surfaces. In *UIST '15: Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology*, pages 37–46, New York, NY, USA. ACM. 15, 67

Plimmer, B., Purchase, H. C., and Yang, H. Y. (2010). Sketchnode: Intelligent sketching support and formal diagramming. In *OZCHI '10: Proceedings of the 22nd Conference of the Computer-Human Interaction Special Interest Group of Australia on Computer-Human Interaction*, pages 136–143. ACM. 12

Qi, Y., Szummer, M., and Minka, T. P. (2005). Diagram structure recognition by Bayesian conditional random fields. In *CVPR 05': Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 191–196. IEEE Computer Society. 9

Refaat, K., Helmy, W., Ali, A., AbdelGhany, M., and Atiya, A. (2008). A new approach for context-independent handwritten offline diagram recognition using support vector machines. In *IJCNN '08: IEEE International Joint Conference on Neural Networks*, pages 177–182. 8

Saund, E., Fleet, D., Larner, D., and Mahoney, J. (2003). Perceptually-supported Image Editing of Text and Graphics. In *UIST '03: Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology*, pages 183–192, New York, NY, USA. ACM. 69

Saund, E. and Mahoney, J. (2004). ScanScribe: Perceptually Supported Diagram Image Editing. In *Diagrammatic Representation and Inference*, volume 2980 of *Lecture Notes in Computer Science*, pages 428–431. Springer. 69

Sezgin, T. M. and Davis, R. (2005). HMM-based efficient sketch recognition. In *IUI '05: Proceedings of the 10th International Conference on Intelligent User Interfaces*, pages 281–283, New York, NY, USA. ACM. 9

Shilman, M., Pasula, H., and Newton, S. R. R. (2002). Statistical visual language models for ink parsing. In *AAAI Spring Symposium on Sketch Understanding*, pages 126–132. 68, 69

Shilman, M., Viola, P., and Chellapilla, K. (2004). Recognition and grouping of handwritten text in diagrams and equations. In *IWFHR 04': Proceedings of the Ninth International Workshop on Frontiers in Handwriting Recognition*, pages 569–574. 68

Shilman, M., Wei, Z., Raghupathy, S., Simard, P., and Jones, D. (2003). Discerning Structure from Freeform Handwritten Notes. In *ICDAR '03: Proceedings of the 7th International Conference on Document Analysis and Recognition*, pages 60–65, Washington, DC, USA. IEEE. 68

Stahovich, T. F., Peterson, E. J., and Lin, H. (2014). An efficient, classification-based approach for grouping pen strokes into objects. *Computers & Graphics*, 42:14 – 30. 67

Stoffel, A., Tapia, E., and Rojas, R. (2009). Recognition of on-line handwritten commutative diagrams. In *ICDAR '09: Proceedings of the 10th International Conference on Document Analysis and Recognition*, pages 1211–1215. 9, 41

Stria, J., Bresler, M., Průša, D., and Hlaváč, V. (2012). MfrDB: Database of annotated on-line mathematical formulae. In Guerrero, J. E., editor, *ICFHR '12: Proceedings of the 13th International Conference on Frontiers in Handwriting Recognition*, pages 540–545, Los Alamitos, USA. IEEE Computer Society. 20, 109

Szwoch, W. and Mucha, M. (2013). *Recognition of Hand Drawn Flowcharts*, volume 184 of *Advances in Intelligent Systems and Computing*. Springer Berlin Heidelberg. 11

Taranta, E. M. and LaViola, Jr., J. J. (2015). Math boxes: A pen-based user interface for writing difficult mathematical expressions. In *IUI '15: Proceedings of the 20th International Conference on Intelligent User Interfaces*, pages 87–96, New York, NY, USA. ACM. 8

Van Phan, T. and Nakagawa, M. (2014). Text/non-text classification in online handwritten documents with recurrent neural networks. In Guerrero, J. E., editor, *ICFHR '14: Proceedings of the 14th International Conference on Frontiers in Handwriting Recognition*, pages 23–28. IEEE Computer Society. 32, 33

Walny, J., Carpendale, S., Riche, N., Venolia, G., and Fawcett, P. (2011). Visual Thinking In Action: Visualizations As Used On Whiteboards. *Visualization and Computer Graphics, IEEE Transactions*, 17(12):2508–2517. 70

Werner, T. (2007). A Linear Programming Approach to Max-Sum Problem: A Review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(7):1165 –1179. 9, 50, 51

Wolin, A., Paulson, B., and Hammond, T. (2009). Sort, merge, repeat: An algorithm for effectively finding corners in hand-sketched strokes. In *SBIM '09: Proceedings of the Sixth Sketch-Based Interfaces and Modeling Symposium*, pages 93–99. 46

Wolin, A., Smith, D., and Alvarado, C. (2007). A pen-based tool for efficient labeling of 2d sketches. In *SBIM '07: Proceedings of the Fourth Sketch-Based Interfaces and Modeling Symposium*, pages 67–74, New York, NY, USA. ACM. 20

Xu, P., Fu, H., Au, O. K.-C., and Tai, C.-L. (2012). Lazy selection. *ACM Transactions on Graphics*, 31(6):142. 68, 69

Zanibbi, R. and Blostein, D. (2012). Recognition and retrieval of mathematical expressions. *International Journal on Document Analysis and Recognition (IJDAR)*, 15(4):331–357. 9

# A. Author's Publications

## A.1. Publications Related to the Thesis

### Impacted Journal Papers

**Bresler, M.**, Průša, D., and Hlaváč, V. (2016). On-line recognition of sketched arrow-connected diagrams. *International Journal on Document Analysis and Recognition (IJDAR).* DOI 10.1007/s10032-016-0269-z. Accepted for publication on May 1, 2016.
Authorship: [50%–30%–20%].
Number of Citations: 0.

### Conference Papers Excerpted by ISI

**Bresler, M.**, Průša, D., and Hlaváč, V. (2013). Modeling flowchart structure recognition as a max-sum problem. In O'Conner, L., editor, *ICDAR '13: Proceedings of the 12th International Conference on Document Analysis and Recognition*, pages 1247–1251. IEEE Computer Society.
Authorship: [45%–45%–10%].
Number of Citations: 5.
14, 41, 67, 109

Stria, J., **Bresler, M.**, Průša, D., and Hlaváč, V. (2012). MfrDB: Database of annotated on-line mathematical formulae. In Guerrero, J. E., editor, *ICFHR '12: Proceedings of the 13th International Conference on Frontiers in Handwriting Recognition*, pages 540–545, Los Alamitos, USA. IEEE Computer Society.
Authorship: [25%–25%–25%–25%].
Number of Citations: 5.
20, 109

### Conference Papers Excerpted by Scopus[1]

Perteneder, F., **Bresler, M.**, Grossauer, E.-M., Leong, J., and Haller, M. (2015). cluster: Smart clustering of free-hand sketches on large interactive surfaces. In *UIST '15: Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, pages 37–46, New York, NY, USA. ACM.
Authorship: [45%–40%–5%–5%–5%].
Number of Citations: 0.
15, 67

**Bresler, M.**, Průša, D., and Hlaváč, V. (2015). Detection of arrows in on-line sketched diagrams using relative stroke positioning. In *WACV '15: IEEE Winter Conference on Applications of Computer Vision*, pages 610–617. IEEE Computer Society.
Authorship: [75%–20%–5%].

[1]We believe that these publications should be excerpted by ISI as well. We see it as a fault of Web of Science that the publication are still not excerpted. Papers from the same conferences from previous years are excerpted. The ISI database is updated too slowly.

Number of Citations: 0.
14, 42, 49

**Bresler, M.**, Van Phan, T., Průša, D., Nakagawa, M., and Hlaváč, V. (2014). Recognition system for on-line sketched diagrams. In Guerrero, J. E., editor, *ICFHR '14: Proceedings of the 14th International Conference on Frontiers in Handwriting Recognition*, pages 563–568. IEEE Computer Society.
Authorship: [60%–20%–10%–5%–5%].
Number of Citations: 3.
14, 35, 40, 109

### Other Conference Papers

**Bresler, M.**, Průša, D., and Hlaváč, V. (2015). Using agglomerative clustering of strokes to perform symbols over-segmentation within a diagram recognition system. In Paul Wohlhart, V. L., editor, *CVWW '15: Proceedings of the 20th Computer Vision Winter Workshop*, pages 67–74. Graz University of Technology.
Authorship: [34%–33%–33%].
Number of Citations: 0.
35

**Bresler, M.** (2013). Text/non-text classification of strokes using the composite descroptor. In Husník, L., editor, *17th International Student Conference on Electrical Engineering*, pages 1–5, Technická 2, Prague, Czech Republic. Czech Technical University in Prague, Czech Republic, Czech Technical University in Prague.
Authorship: [100%].
Number of Citations: 1.
110

**Bresler, M.**, Průša, D., and Hlaváč, V. (2013). Simultaneous segmentation and recognition of graphical symbols using a composite descriptor. In Kropatsch, W. G., Ramachandran, G., and Torres, F., editors, *CVWW 2013: Proceedings of the 18th Computer Vision Winter Workshop*, pages 16–23, Karlsplatz 13, Vienna, Austria. Vienna University of Technology.
Authorship: [34%–33%–33%].
Number of Citations: 2.
110

## A.2. Other Publications

### Conference Papers not Excerpted by ISI

Fojtů, Š., **Bresler, M.**, and Průša, D. (2012). Nao robot navigation based on a single VGA camera. In Kristan, M., Čehovin, L., and Mandeljc, R., editors, *CVWW '12: Proceedings of the 17th Computer Vision Winter Workshop*, pages 121–128, Ljubljana, Slovenia. Slovenian Pattern Recognition Society.
Authorship: [45%–45%–10%].
Number of Citations: 1.
110

# B. Citations of Author's Work

The 17 known citations of the author's work are grouped together according to the paper they cite and listed below. The corresponding H-index is 3 in Google Scholar.Web of Science failed to resolve most of the citations and thus the H-index in Web of Science is 1.

Bresler et al. [2014] Recognition system for on-line sketched diagrams.

- Phan, T. V. and Nakagawa, M. (2016). Combination of global and local contexts for text/non-text classification in heterogeneous online handwritten documents. *Pattern Recognition*, 51:112–124.
- Delaye, A. and Lee, K. (2015). A flexible framework for online document segmentation by pairwise stroke distance learning. *Pattern Recognition*, 48(4):1197–1210.
- Delaye, A. (2014). Structured prediction models for online sketch recognition. Unpublished manuscript, `https://sites.google.com/site/adriendelaye/home/news/unpublishedmanuscriptavailable`.

Bresler et al. [2013] Modeling flowchart structure recognition as a max-sum problem.

- Wu, J., Wang, C., Zhang, L., and Rui, Y. (2014). Sketch recognition with natural correction and editing. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 951–957.
- 邓维, 吴玲达, 张友根, 杨超, and 吕雅帅 (2014). 采用快速图元匹配的手绘非规则军标识别. 计算机辅助设计与图形学学报, 26(10):1835–1843.
- Inatani, S., Van Phan, T., and Nakagawa, M. (2014). Comparison of MRF and CRF for text/non-text classification in Japanese ink documents. In Guerrero, J. E., editor, *ICFHR '14: 14th International Conference on Frontiers in Handwriting Recognition*, pages 684–689. IEEE Computer Society.
- Chen, Q., Shi, D., Feng, G., Zhao, X., and Luo, B. (2015). On-line handwritten flowchart recognition based on logical structure and graph grammar. In *ICIST '15: Proceedings of the 5th International Conference on Information Science and Technology*, pages 424–429. IEEE Computer Society.
- Wu, J., Wang, C., Zhang, L., and Rui, Y. (2015). Offline sketch parsing via shapeness estimation. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 1200–1206. AAAI Press.

Stria et al. [2012] MfrDB: Database of annotated on-line mathematical formulae.

- Mouchère, H., Viard-Gaudin, C., Zanibbi, R., Garain, U., Kim, D. H., and Kim, J. H. (2013). ICDAR 2013 CROHME: Third international competition on recognition of online handwritten mathematical expressions. In *ICDAR '13: Proceedings of the 12th International Conference on Document Analysis and Recognition*, pages 1428–1432. IEEE Computer Society.

- Hu, Y., Peng, L., and Tang, Y. (2014). On-line handwritten mathematical expression recognition method based on statistical and semantic analysis. In *DAS 14': Proceedings of the 11th IAPR International Workshop on Document Analysis Systems*, pages 171–175. IEEE Computer Society.

- Mouchère, H., Zanibbi, R., Garain, U., and Viard-Gaudin, C. (2016). Advancing the state of the art for handwritten math recognition: the CROHME competitions, 2011–2014. *International Journal on Document Analysis and Recognition (IJDAR)*, pages 1–17.

- Hirata, N. S. and Julca-Aguilar, F. D. (2015). Matching based ground-truth annotation for online handwritten mathematical expressions. *Pattern Recognition*, 48(3):837–848.

- de Oliveira, M. V. (2014). Um estudo empírico sobre classificação de símbolos matemáticos manuscritos. PhD thesis, Universidade de Sãao Paulo.

Bresler [2013] Text/non-text classification of strokes using the composite descriptor.

- Phan, T. V. and Nakagawa, M. (2016). Combination of global and local contexts for text/non-text classification in heterogeneous online handwritten documents. *Pattern Recognition*, 51:112–124.

Bresler et al. [2013] Simultaneous segmentation and recognition of graphical symbols using a composite descriptor.

- 吴玲达, 邓维, 张友根, and 杨超(2015). 在线草图识别研究综述. 计算机应用研究, 32(6):1601–1607.

- Cheema, S. (2014). Pen-based methods for recognition and animation of handwritten physics solutions. PhD thesis, University of Central Florida Orlando, Florida.

Fojtů et al. [2012] Nao robot navigation based on a single VGA camera.

- Wen, S., Chen, X., Ma, C., Lam, H., and Hua, S. (2015). The Q-learning obstacle avoidance algorithm based on EKF-SLAM for NAO autonomous walking under unknown environments. *Robotics and Autonomous Systems*, 72:29–36.