IJDAR (2016) 19:253–267
DOI 10.1007/s10032-016-0269-z

ORIGINAL PAPER

Online recognition of sketched arrow-connected diagrams

Martin Bresler[1] · **Daniel Průša**[1] · **Václav Hlaváč**[2]

Received: 27 May 2015 / Revised: 15 March 2016 / Accepted: 1 May 2016 / Published online: 19 May 2016

Abstract We introduce a new, online, stroke-based recognition system for hand-drawn diagrams which belong to a group of documents with an explicit structure obvious to humans but only loosely defined from the machine point of view. We propose 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 where 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. Although the individual pipeline steps are tailored for these, the system can readily be adapted for other domains. Our entire diagram recognition pipeline is outlined. Its core parts are text/non-text separation, symbol segmentation, their classification and structural analysis. Individual parts have been published by the authors previously and so are described briefly and referenced. Thorough evaluation on benchmark databases shows the accuracy of the system reaches the state of the art and is ready for practical use. The paper brings several contributions: (a) the entire system and its state-of-the-art performance; (b) the methodology exploring document structure when it is loosely defined; (c) the thorough experimental evaluation; (d) the new annotated database for online sketched flowcharts and finite automata diagrams.

✉ Martin Bresler
breslmar@fel.cvut.cz

[1] Center for Machine Perception, Faculty of Electrical Engineering, Czech Technical University in Prague, Technická 2, 166 27 Prague 6, Czech Republic

[2] Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, Zikova 4, 166 36 Prague 6, Czech Republic

Keywords Diagram recognition · Online document analysis · Max-sum problem · Segmentation · Text/non-text separation · Flowcharts · Finite automata

1 Introduction

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. Even though recognizers with good precision have been presented, e.g. of mathematical formulas [2,21], the availability of diagram recognizers is still limited. One reason may be that diagram domains differ significantly and their structure is loose in comparison with, for example, mathematical formulas. This paper attempts to fill this gap. We present a general recognition system suitable for a variety of online sketched diagrams. We introduced the idea in [9]; since then, the system has evolved into a pipeline depicted in Fig. 1. We provide its comprehensive description, focusing on core principles proposed for segmentation, classification and structural analysis. A brief description and references are also given for those parts, in which existing solutions were adopted. Our experiments yield a thorough evaluation of the system accuracy, the discussion of failure types and the impact of particular steps on the overall performance.

An online 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. The most common device is a tablet, a tablet PC or a smart board. 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. The output is a structure which syntactically describes the sketched diagram. Individ-

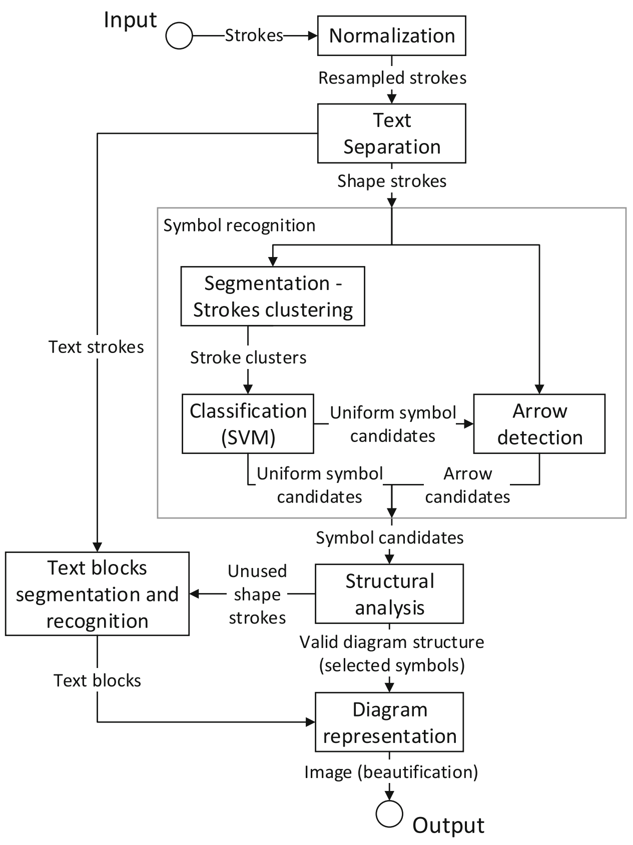


Fig. 1 Proposed recognition pipeline

ual 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 do not seem to be unified. We use the DOT graph description language, supported by the popular graph visualizer Graphviz.[1] An example of finite automata beautified by Graphviz is shown in Fig. 2

There has also been some research done in offline diagram recognition [31]. In that case, input is an image, and thus, any temporal information is missing. Offline recognition faces different challenges (especially in segmentation) and typically leads to different applications. This paper does not consider these topics.

We target domains which exhibit the following structure: the diagram consists of *symbols* connected by *arrows*, and there might be *text* labelling the symbols (the text is inside or along the border of a symbol) or arrows (the text is in a vicinity of an arrow). Although this structure is basic, there are various domains which fit it perfectly. We focus on two of them: flowcharts and finite automata. Other domains such as UML use case diagrams, Simulink diagrams, Concur Task Trees or business process diagrams fall into this group as

[1] http://graphviz.org/.

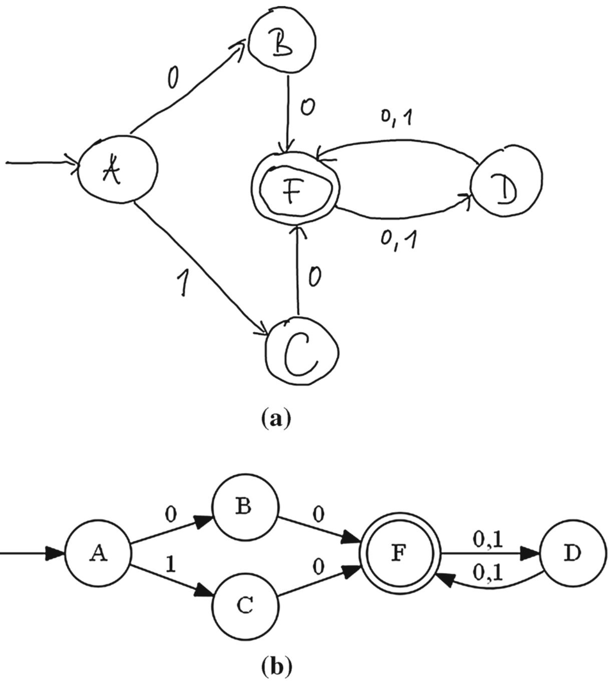


Fig. 2 An example of **a** a typical input and **b** the desired output of the proposed diagram recognizer

well—we leave adaptation to these domains as future work. Domains with a different structure (e.g. music scores or chemical diagrams) or a structure further extending the basic constructs (e.g. diagrams with structured text inside symbols) are beyond the scope of this paper.

The paper is organized as follows. Section 2 briefly surveys diagram recognition systems. Section 3 defines the example domains and introduces datasets used. Section 4 describes our recognition system, following the recognition pipeline in Fig. 1. Specifically, Sect. 4.1 describes our text/non-text separation, Sect. 4.2 explains the symbol segmentation through strokes clustering, Sect. 4.3 introduces our symbol detectors, and Sect. 4.4 presents the structural analysis. Section 5 contains experimental results. Section 6 analyses the developed system and deals with failure cases, and Sect. 7 presents conclusions.

2 Related work

2.1 Mathematical formulas

Recognition of mathematical formulas is a useful example as it has brought seminal methods and successful recognizers. A practical example is the Math Input Panel delivered by Microsoft since Windows 7. The Competition on Recognition of Online Handwritten Mathematical Expressions (CROHME) has further boosted research and provides a reliable comparison ground. The commercial winner of CROHME 2014 [26], MyScript,[2] achieves 62.7 % accuracy of correctly recognized formulas, while the non-commercial winner—Álvaro [2]—achieves 37.2 %.

Recognition of mathematical formulas and diagrams faces 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 [2,21]. In contrast, diagram structure is simpler and less formalized and grammars do not seem to be the best model for their capture.

2.2 Diagrams in general

Feng et al. [15] proposed a recognizer for online 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, and this makes the system slow and prohibits it from practical use. Sezgin and Davis [32] 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 [28], 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. [30] 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 on 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 max-sum labelling problem rather than pairwise features does not require additional training and yields optimal solutions. The model is thus much simpler and more open to adaptations.

Finally, there were attempts to develop universal formalisms for sketch recognition. LADDER [16] is a sketch description language that can be used to describe how shapes are drawn as well a the whole syntax specifying a domain. A multi-domain sketch recognition engine SketchREAD [1] 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 this low-level errors, it still impose additional source of error.

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. [22] proposed a grammar-based recognition system for flowcharts which uses the Description and MOdification of the Segmentation (DMOS) method for structured document recognition. The applied grammatical language Enhanced Position Formalism (EPF) provides a syntactic and structural description of flowcharts, which is used for joint symbol segmentation and classification. Carton et al. [10] 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 whether 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 still has trouble, with big uncertainty in the input. Experiments were performed on a benchmark database, which allows comparison. Work by Szwoch and Mucha [34] 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 [11] has recently introduced a purely statistical approach to diagram recognition based on strokes clustering and CRFs: 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 labellings can occur.

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

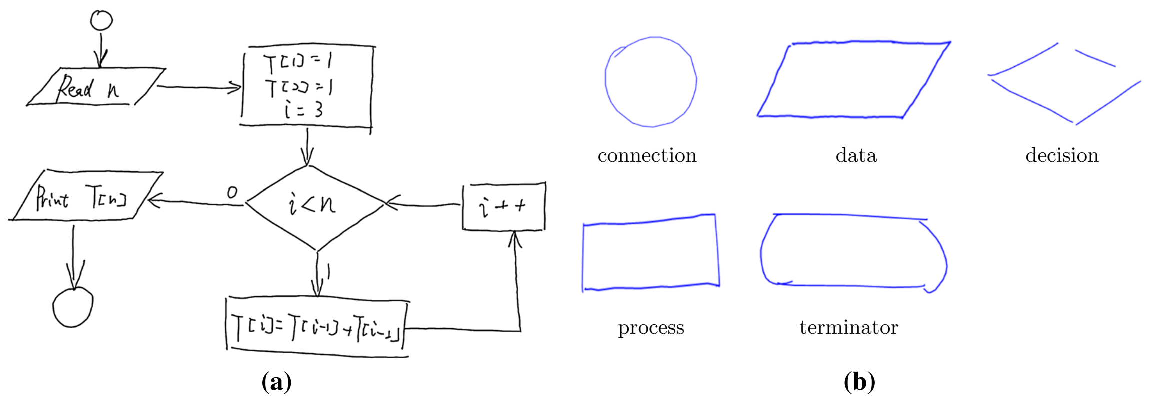


Fig. 3 Example of a flowchart (**a**) with examples of uniform symbol classes (**b**)

2.4 Alternative approaches

Though there are few systems directly comparable to ours, interest in diagram design/sketching is evident. Miyao and Maruyama [25] 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.

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 [3,29].

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 or they rely on grammars which are too impractical for the minimalistic structure of diagrams.

3 Diagram structure and supported domains

The diagram recognition system we propose is general and can be used in several domains. Nevertheless, it requires adaptation for a chosen domain mostly by retraining classifiers. We introduce domains, together with benchmark databases used for training/testing of the system. 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. We worked in two diagram domains, flowcharts (FC) and finite automata (FA), for both of which there are freely available benchmark databases. FC and FA share important properties; however, their specifics should be considered when adapting the recognizer to achieve best results.

3.1 Flowcharts

Figure 3 shows an example flowchart, together with an enumeration of five uniform symbol classes: *connection*, *data*, *decision*, *process* and *terminator*.

Awal et al. [4] released a benchmark database of flowcharts (referenced as FC_A), on which several state-of-the-art methods were tested. The database consists of 327 diagrams drawn by 35 users: predefined diagram patterns representing well-known algorithms were used. The samples are divided into training and test datasets. 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. The data are of low quality (sampling frequency).

These deficiencies have motivated us to collect our own flowchart database and make it public:[3] we reference this database as FC_B. We collected 28 diagram patterns drawn by 24 users, 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.

3.2 Finite automata

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

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

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 Fig. 2.

No publicly available database of online sketched finite automata is known. We gathered and annotated our own database (referenced as the FA database) and make it public[3]. Compared to the previous release [9], the annotation was improved—arrows are provided with their connection points and heads annotated. The database contains samples of 12 diagram patterns drawn by 25 users, which results in 300 diagrams divided into training, validation and test datasets.

4 Recognition pipeline

4.1 Text detection and recognition

Separating text 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 and finding symbols, to which the blocks are assigned.

The text/non-text separation algorithm classifies single strokes into two classes—*text* and *shapes*. Although there exist quite precise algorithms for such separation [5,14,19,27,35], they are not able to separate all text strokes. Their accuracy achieved on the benchmark IAMonDo database[4] 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. 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.

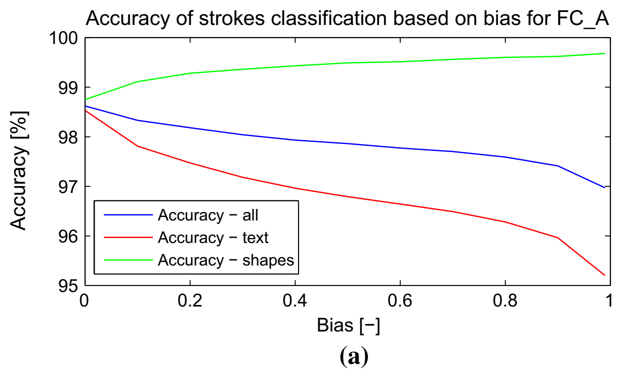
We bias the classifier result, and thus, only strokes where the classifier is almost certain are marked as text. We implemented two classifiers performing best on the IAMonDo database [27,35] and tested them on flowcharts and finite automata. The best performing was the classifier proposed by Phan and Nakagawa [35], 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 labelling 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. The bias value is simply used to increase the confidence of the label *shapes* for individual strokes. Its selection is a trade-off between the accuracy in both classes. Results for various bias values are shown in Fig. 4. We chose 0.99 for FC_A, 0.8 for FC_B and 0.7 for FA. Note that Phan and Nakagawa [35] 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.

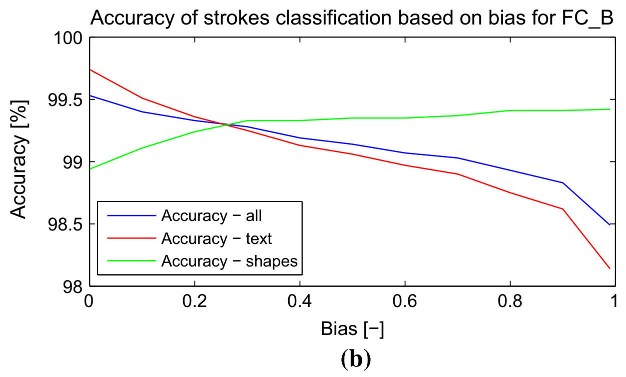
Text recognition is performed when the diagram structure is already known. Strokes removed during the text/non-text separation step together with strokes not identified as a symbol are processed; the diagram structure provides enough information to group the unused strokes into text blocks. Two types of text blocks are distinguished, labelling the uniform symbol and labelling the arrow. The grouping is accomplished in two steps. Text blocks inside symbols are found first; text blocks labelling arrows are found next. Each text block labelling 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 labelling arrows are found easily by a grouping based on the spatiotemporal proximity. These blocks are salient objects; hence, it is easy to find a suitable grouping threshold. Afterwards, each text block is attached to the closest arrow. Recognition of text inside a block is beyond the scope of this paper.

4.2 Symbol segmentation

Segmentation is a process which divides 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 *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

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





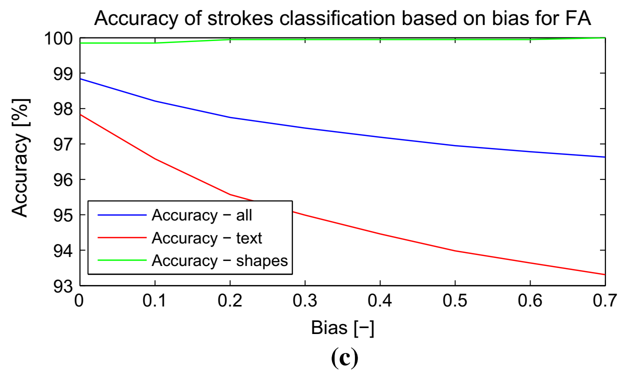


Fig. 4 Accuracy of stroke classification in relation to bias. Selection of the best bias is a trade-off between accuracy in *text* and *shapes* classes. **a** FC_A, **b** FC_B and **c** FA

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. Strokes grouping is performed iteratively. Within the first iteration, every single stroke forms a subset of size 1. Further, 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, and spatial proximity is determined simply by Euclidean distance. Temporal proximity is hinted at by stroke indices in the drawing sequence. The approach has been used in our previous recognizer [9] as well as by others [2,15,28].

Delaye and Lee [13] 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, the features express difference between geometric and temporal characteristics of two strokes. The distance is defined as:

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

where s and t are two given strokes and w_i is the weight for the feature d_i that needs to be learned. It is a hierarchical bottom-up clustering technique where larger clusters are created by iteratively merging the two closest clusters based on 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 $\mathcal{O}(n^2)$, where n is the number of strokes. The clustering gives a final segmentation, which reaches typically lower recall rates. We improved it in our previous work [8], performing the over-segmentation by successive clusterings with varying thresholds. In comparison with grouping-based over-segmentation, this increased the precision (i.e. generated significantly fewer subsets), at the cost of only very slightly decreased recall. In the current implementation, we achieved 95.1/16.7, 98.4/27.5, 99.8/26.5 % recall/precision on FC_A, FC_B, FA databases. High recall is the main objective directly affecting the precision of the whole system while 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.

4.3 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 vary, which is a difficulty for traditional classifiers. There are two stages. First, uniform symbols are recognized using a standard classifier. Second, arrows are detected as connectors between symbol candidates found earlier. Both recognizers/detectors provide an ordered list of symbol candidates. The structural analysis selects actual symbols from these lists.

4.3.1 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 candidates' quality.

We used an off-the-shelf solution and combined the trajectory-based normalization and direction features proposed by Liu and Zhou [23] as a descriptor, which serves 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. Besides the trajectory-based normalization, we do not perform any particular pre-processing. The descriptor consists of 512 features; it was primarily designed for recognition of Japanese characters and thus works well for 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. 5, and the achieved results confirm suitability of the selected descriptor. Although there are different descriptors available [12], 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 strokes clustering introduced in Sect. 4.2. 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 fivefold cross-validation. Therefore, we merged the training and validation datasets in case of FC_B and FA databases to have the same conditions.

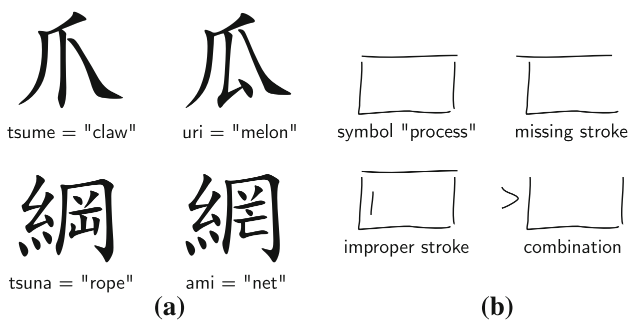


Fig. 5 Example of small differences between individual symbol classes in the case of Japanese characters (**a**) and uniform symbols in diagrams (**b**)

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* 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 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 [9]. However, our current implementation uses artificially synthesized samples to balance the database. The samples were synthesized using the approach of Martín-Albo et al. [24]. It is based on kinematic theory and the distortion of the Sigma–Lognormal parameters in order to generate human-like synthetic samples. 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. This approach not only helps to balance the dataset, but 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 achieves precision of 98.9, 97.5 and 100.0 % for FC_A, FC_B and FA, respectively. If rejection is incorporated 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 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.

4.3.2 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 [7] which did not lead to satisfactory results. Some generic recognizers for arrows are also known [17,18,20,33]; however, they expect a fixed form of arrows—the number of strokes as well as the range of angles between them is 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.

Our detector 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 shaft detection is done by adding strokes iteratively to a sequence such that the first stroke starts in the vicinity of the first symbol and the last stroke ends in the vicinity of the second symbol. Once the shaft is found, the head is detected by classification of strokes around both ends of the shaft into two classes: *head* and *not head*. This classification is based on relative positioning of strokes. Detected arrows are assigned a score given by the quality of the shaft and the head. More details can be found in our original paper [6].

To test the arrow detector, we took all annotated uniform symbols and tried to find arrows connecting them. Detected arrows were compared with the annotated arrows. Note that text strokes were removed by our text/non-text separator and all pairs of symbols were considered. The result of the arrow detector is a list of arrow candidates since it may detect several conflicting arrows between each pair of symbols. These conflicts are intended to be solved by the structural analyser. We achieved a 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 on average per diagram when searching for arrow heads when there are ten arrows on average per diagram.

4.4 Structural analysis

The input to the structural analysis comprises 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—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 score $s_c = -\infty$. Each relation of type (2) gets the score

$$s_o = -S_{A\cap B}/\min(S_A, S_B), \tag{2}$$

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, relations of type (3) get score $s_e = -\infty$. 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.

4.4.1 Max-sum formulation

The pairwise max-sum labelling problem [36] (a.k.a. computing the MAP configuration of a Markov random field) is defined as maximizing the sum of unary and binary cost functions (potentials of discrete variables)

$$\max_{\boldsymbol{k}\in K^V}\left[\sum_{u\in V} g_u(k_u) + \sum_{\{u,v\}\in E} g_{uv}(k_u, k_v)\right], \tag{3}$$

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 $\{u, v\}$ are 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. $g_{uv}(1, 1) = s_o$ if u, v are two non-arrow symbol candidates with overlapping boxes (s_o is given by (2)).
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.

4.4.2 Example

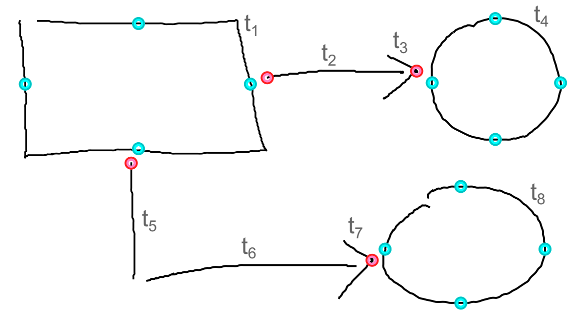
We illustrate the structural analysis by a simple example. Figure 6a contains an input flowchart with labelled strokes. We assume that the following symbol candidates were detected in this diagram:

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.

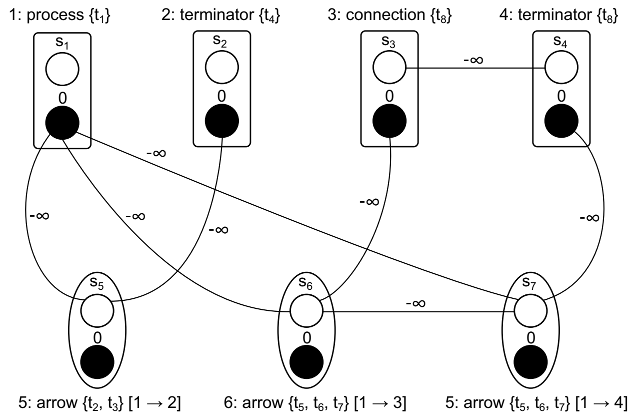
The strokes forming each symbol candidate are in curly brackets. For arrow candidates, the values in the square brackets say which symbols are connected by the arrow. The resulting max-sum model is depicted in Fig. 6b.

4.4.3 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 a polynomial time [36]. 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 fast (see Sect. 5). We tested the max-sum solver Toulbar2.[5] 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.[6] The conversion is based on linear programming relaxation of the problem [36].



(a)



(b)

Fig. 6 Input flowchart (**a**) consisting of eight strokes $t_1, \ldots, t_8$ forming three uniform *symbols* and two *arrows*. Connection points are marked in *blue* and *red* for uniform *symbols* and *arrows*, respectively. There are four 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 ℓ in a node v is assigned by value $g_{uv}(k, \ell)$. Edges with zero costs are not shown (colour figure online)

[5] http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro.

[6] http://www.ibm.com/software/integration/optimization/cplex-optimizer/.

We made experiments with both formulations and compared the performance of Toulbar2 and CPLEX. Both solvers find the exact solution, and the recognition result is the same. Toulbar2 is approx. 60 % faster on average. It appears slower for the smallest diagrams; this is caused by the data exchange trough files, which is the only possibility supported by the binary distribution of Toulbar2. Some minimal time is thus needed for the solver initialization. This minor technical limitation may be resolved in future. Details on runtimes are provided in Sect. 5.

5 Experiments

We present an overall performance evaluation of the proposed diagram recognition system (abbreviated γ). We made a comparison with two state-of-the-art methods: the grammar-based method (α) by Carton et al. [10] and the purely statistical method (β) by Delaye [11], and compared our performance to their published results. The former was evaluated on the FC_A database only, while the second was evaluated on the FA database as well. FC_B is new (introduced in this work), 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 (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 whether 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 α and β, and thus, they allow fair comparison of individual systems. However, exact stroke matching is not necessarily required for the correct diagram structure recognition. Users sometimes draw symbols by multiple strokes when correcting or when beautifying symbols. Some strokes are redundant in some way. 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. Contrary, 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 Fig. 7 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 1, 2 and 3. Differences in symbol segmentation and recognition results given by the two criteria SR1 and SR2 are shown in Table 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. The average runtime needed for recognition was 0.78, 0.89 and 0.69 s

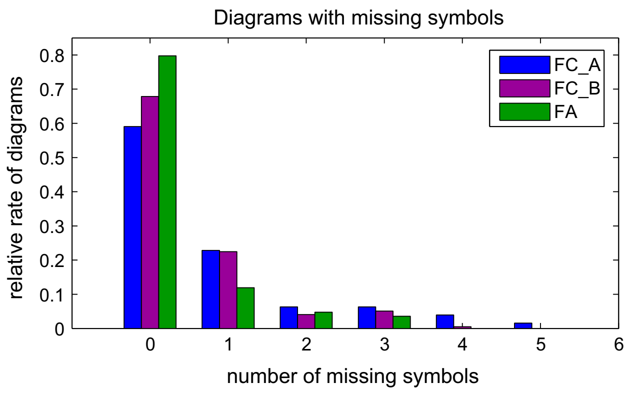


Fig. 7 Counts of diagrams by the number of missing symbols in the result

Table 1 Recognition results for FC_A database

Class	SL (%)			SR1 (%)		
	(α)	(β)	(γ)	(α)	(β)	(γ)
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

Comparison of the proposed recognizer (γ) to the grammar-based method (α) and to the purely statistical method (β). We list correct stroke labelling (SL) and symbol segmentation and recognition measured with the strict (SR1) method

Table 2 Results for FC_B database

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

We list correct stroke labelling (SL) and symbol segmentation and recognition measured with strict (SR1) method

Table 3 Recognition results for FA database

Class	SL (%)		SR1 (%)	
	(β)	(γ)	(β)	(γ)
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

We compared the proposed system (γ) with the purely statistical method by Delaye (β). We list correct stroke labelling (SL) and symbol segmentation and recognition measured with strict (SR1) method

Table 4 Comparison of the symbol segmentation and recognition rates using strict (SR1) and structure-based (SR2) method

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

for diagrams from FC_A, FC_B and FA, respectively. This means that our system is faster than the grammar-based system 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 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 6.

Table 5 Optimization/total running time

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 6 Running time consumed to solve the optimization by Toulbar2/CPLEX

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

6 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.

6.1 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 an arrow 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 a text block, misrecognition of which is always caused by the misrecognition of a symbol it labels, and thus, it is not a part

Table 7 Numbers of misrecognized symbols from the individual symbol classes with rates of the reasons for their misrecognition (dominant shown bolded)

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**

The displayed result correspond to FC_A/FC_B/FA databases

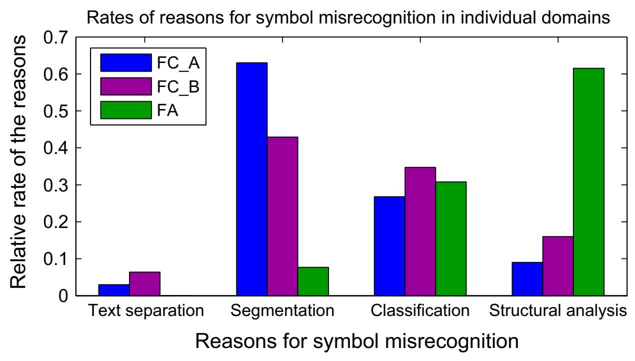


Fig. 8 Rates of reasons for symbol misrecognition

of the analysis. The rate of each reason for symbol misrecognition with respect to the symbol class is shown in Table 7. The average rates are shown for individual datasets in Fig. 8. It has turned out that the most frequent reason for failure is the symbol misclassification in the case of flowcharts and wrong selection of symbol candidates by the structural analyser in the case of finite automata. Diagrams with the highest number of misclassified symbols are analysed in Fig. 9.

6.2 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 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. 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 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 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 time information which is necessary for synthesis of the artificial samples; thus, we could not do this analysis on the database.

6.3 Analysis findings

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

The *text/non-text separation* step 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 flowcharts, 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 ground truth led to faster recognition, because all text strokes were removed.

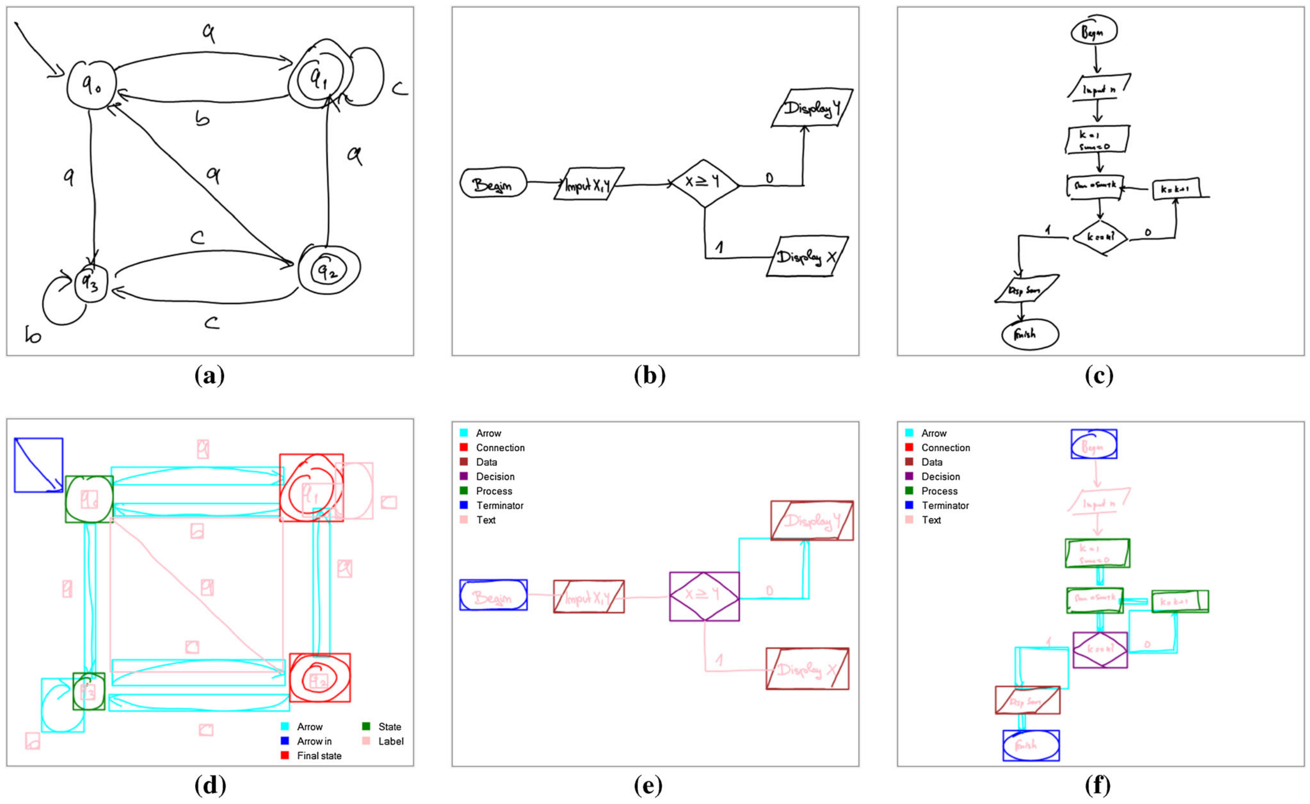


Fig. 9 Examples of misrecognized diagrams from the FA and FC_B databases. The input diagrams are shown in the *upper row*; recognition results are in the *bottom row*. *Symbol colouring* is explained in the legends which are parts of the images. Each recognized *symbol* is surrounded by its *bounding box*. **a** Input diagram writer020_fa_002, **b** input diagram writer018_fc_003b, **c** input diagram writer019_fc_006b, **d** misrecog. *arrows*—colliding heads, **e** misrecog. *arrows*—a missing head and **f** unrecognized data—interstroke gaps (colour figure online)

Table 8 Results of various analyses showing the effect of individual solutions to the steps of the pipeline

	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	1220	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	2273	1060	–
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	3429	1205	930

It is compared to the baseline which was used to obtain the results in Sect. 5. We measured correct rate of stroke labelling (SL), symbol segmentation and recognition measured with strict (SR1) and structure-based (SR2) method, and average recognition time (AT)

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 cannot 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.

7 Conclusion

We have designed a recognizer that understands diagrams such as flowcharts or finite automata. We studied which methods are optimal for the implementation of individual pipeline steps. Some existing algorithms were suitable for this purpose and were thus integrated. Beyond these, several new approaches were introduced. The system performed well in comparison with the state of the art. We encourage the reader to experiment with our demo application available at http://cmp.felk.cvut.cz/~breslmar/diagram_recognizer/.

The low quality of the benchmark flowchart database motivated us to gather our own database. The main weaknesses of the FC_A database are the lack of temporal information and incomplete annotation—our database contains temporal information as well as information about pressure which may be useful for development of new methods. It provides complex annotations, including meaning of the text and the annotation of arrow heads. This higher quality database better reflects current standards in ink input interfaces. Experiments proved that high-quality data yield more accurate results. As another contribution, we make this database available for the community.

In future work, we would like to adapt the proposed method for other diagrammatic domains, and we are planning to cooperate with diagram users professionally. We are interested especially in 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. Next, we would like to experiment with iterative recognition using immediate feedback to the user of intermediate results. This may well reduce recognition time, and we believe that the proposed system is capable of such adaptation. Another possibility is to extend it to support domains beyond the scope of arrow-connected diagrams. This extension would require 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. For examples, the relation between notes and staff lines in the case of music scores.

Acknowledgments The first author was supported by the Grant Agency of the CTU under the project SGS16/085/OHK3/1T/13. The second and the third authors were supported by the Czech Science Foundation under grant no. 15-04960S. The authors thank Truyen van Phan for his help with the text/non-text separation, Daniel Martín-Albo for creating the synthesized samples for the SVM classifiers and Roger Boyle for proofreading of the paper.

References

1. Alvarado, C., Davis, R.: SketchREAD: a multi-domain sketch recognition engine. In: UIST '04: 17th Annual ACM Symposium on User Interface Software and Technology. UIST '04, pp. 23–32. ACM, New York (2004)
2. Álvaro, F., Sánchez, J.A., Benedí, J.M.: Recognition of on-line handwritten mathematical expressions using 2d stochastic context-free grammars and hidden Markov models. Pattern Recogn. Lett. **35**, 58–67 (2014)
3. Arvo, J., Novins, K.: Appearance-preserving manipulation of hand-drawn graphs. In: 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia, GRAPHITE '05, pp. 61–68. ACM (2005)
4. Awal, A.M., Feng, G., Mouchere, H., Viard-Gaudin, C.: First experiments on a new online handwritten flowchart database. In: DRR'11, pp. 1–10 (2011)
5. Blagojevic, R., Plimmer, B., Grundy, J., Wang, Y.: Using data mining for digital ink recognition: dividing text and shapes in sketched diagrams. Comput. Graph. **35**(5), 976–991 (2011)
6. Bresler, M., Průša, D., Hlaváč, V.: Detection of arrows in on-line sketched diagrams using relative stroke positioning. In: WACV '15: IEEE Winter Conference on Applications of Computer Vision, pp. 610–617. IEEE Computer Society (2015)
7. Bresler, M., Průša, D., Hlaváč, V.: modeling flowchart structure recognition as a max-sum problem. In: O'Conner, L. (ed.) ICDAR '13: 12th International Conference on Document Analysis and Recognition, pp. 1247–1251. IEEE Computer Society (2013)
8. Bresler, M., Průša, D., Hlaváč, V.: Using agglomerative clustering of strokes to perform symbols over-segmentation within a diagram recognition system. In: Paul Wohlhart, V.L. (ed.) CVWW '15: Proceedings of the 20th Computer Vision Winter Workshop, pp. 67–74. Graz University of Technology (2015)
9. Bresler, M., Van Phan, T., Průša, D., Nakagawa, M., Hlaváč, V.: Recognition system for on-line sketched diagrams. In: Guerrero, J.E. (ed.) ICFHR '14: 14th International Conference on Frontiers in Handwriting Recognition, pp. 563–568. IEEE Computer Society (2014)
10. Carton, C., Lemaitre, A., Couasnon, B.: Fusion of statistical and structural information for flowchart recognition. In: ICDAR '13:

12th International Conference on Document Analysis and Recognition, pp. 1210–1214 (2013)
11. Delaye, A.: Structured prediction models for online sketch recognition (2014). Unpublished manuscript. https://sites.google.com/site/adriendelaye/home/news/unpublishedmanuscriptavailable
12. Delaye, A., Anquetil, E.: HBF49 feature set: a first unified baseline for online symbol recognition. Pattern Recogn. **46**(1), 117–130 (2013)
13. Delaye, A., Lee, K.: A flexible framework for online document segmentation by pairwise stroke distance learning. Pattern Recogn. **48**(4), 1197–1210 (2015)
14. Delaye, A., Liu, C.L.: Contextual text/non-text stroke classification in online handwritten notes with conditional random fields. Pattern Recogn. **47**(3), 959–968 (2014)
15. Feng, G., Viard-Gaudin, C., Sun, Z.: On-line hand-drawn electric circuit diagram recognition using 2D dynamic programming. Pattern Recogn. **42**(12), 3215–3223 (2009)
16. Hammond, T., Davis, R.: LADDER, a sketching language for user interface developers. Comput. Graph. **29**, 518–532 (2005)
17. Hammond, T., Davis, R.: Tahuti: A geometrical sketch recognition system for UML class diagrams. In: ACM SIGGRAPH 2006 Courses, SIGGRAPH '06. ACM, New York (2006)
18. Hammond, T., Paulson, B.: Recognizing sketched multistroke primitives. ACM Trans. Interact. Intell. Syst. **1**(1), 4:1–4:34 (2011)
19. Indermühle, E., Frinken, V., Bunke, H.: Mode detection in online handwritten documents using BLSTM neural networks. In: ICFHR '12: 13th International Conference on Frontiers in Handwriting Recognition, pp. 302–307 (2012)
20. Kara, L.B., Stahovich, T.F.: Hierarchical parsing and recognition of hand-sketched diagrams. In: 17th Annual ACM Symposium on User Interface Software and Technology, UIST '04, pp. 13–22. ACM (2004)
21. Le, A.D., Van Phan, T., Nakagawa, M.: A system for recognizing online handwritten mathematical expressions and improvement of structure analysis. In: DAS '14: 11th IAPR International Workshop on Document Analysis Systems, pp. 51–55 (2014)
22. Lemaitre, A., Mouchére, H., Camillerapp, J., Coüasnon, B.: Interest of syntactic knowledge for on-line flowchart recognition. In: GREC '11: 9th IAPR International Workshop on Graphics Recognition, pp. 85–88 (2011)
23. Liu, C.L., Zhou, X.D.: Online Japanese character recognition using trajectory-based normalization and direction feature extraction. In: Lorette, G. (ed.) Tenth International Workshop on Frontiers in Handwriting Recognition. Université de Rennes 1, Suvisoft (2006)
24. Martín-Albo, D., Plamondon, R., Vidal, E.: Training of on-line handwriting text recognizers with synthetic text generated using the kinematic theory of rapid human movements. In: Guerrero, J.E. (ed.) ICFHR '14: 14th International Conference on Frontiers in Handwriting Recognition, pp. 543–548. IEEE Computer Society (2014)
25. Miyao, H., Maruyama, R.: On-line handwritten flowchart recognition, beautification and editing system. In: ICFHR '12: 13th International Conference on Frontiers in Handwriting Recognition, pp. 83–88 (2012)
26. Mouchère, H., Viard-Gaudin, C., Zanibbi, R., Garain, U.: ICFHR 2014 competition on recognition of on-line handwritten mathematical expressions (CROHME 2014). In: J.E. Guerrero (ed.) ICFHR '14: 14th International Conference on Frontiers in Handwriting Recognition, pp. 791–796. IEEE Computer Society (2014)
27. Otte, S., Krechel, D., Liwicki, M., Dengel, A.: Local feature based online mode detection with recurrent neural networks. In: ICFHR '12: 13th International Conference on Frontiers in Handwriting Recognition, pp. 531–535 (2012)
28. Ouyang, T.Y., Davis, R.: Chemink: A natural real-time recognition system for chemical drawings. In: 16th International Conference on Intelligent User Interfaces, IUI '11, pp. 267–276. ACM (2011)
29. Plimmer, B., Purchase, H.C., Yang, H.Y.: Sketchnode: intelligent sketching support and formal diagramming. In: 22nd Conference of the Computer-Human Interaction Special Interest Group of Australia on Computer-Human Interaction, OZCHI '10, pp. 136–143. ACM (2010)
30. Qi, Y., Szummer, M., Minka, T.P.: Diagram structure recognition by Bayesian conditional random fields. In: Conference on Computer Vision and Pattern Recognition, pp. 191–196. IEEE Computer Society (2005)
31. Refaat, K., Helmy, W., Ali, A., AbdelGhany, M., Atiya, A.: A new approach for context-independent handwritten offline diagram recognition using support vector machines. In: IJCNN '08: IEEE International Joint Conference on Neural Networks, pp. 177–182 (2008)
32. Sezgin, T.M., Davis, R.: HMM-based efficient sketch recognition. In: IUI '05: 10th International Conference on Intelligent User Interfaces. IUI '05, pp. 281–283. ACM, New York (2005)
33. Stoffel, A., Tapia, E., Rojas, R.: Recognition of on-line handwritten commutative diagrams. In: ICDAR '09: 10th International Conference on Document Analysis and Recognition, pp. 1211–1215 (2009)
34. Szwoch, W., Mucha, M.: Recognition of Hand Drawn Flowcharts, Advances in Intelligent Systems and Computing, vol. 184. Springer, Berlin (2013)
35. Van Phan, T., Nakagawa, M.: Text/non-text classification in online handwritten documents with recurrent neural networks. In: J.E. Guerrero (ed.) ICFHR '14: 14th International Conference on Frontiers in Handwriting Recognition, pp. 23–28. IEEE Computer Society (2014)
36. Werner, T.: A linear programming approach to max-sum problem: a review. IEEE Trans. Pattern Anal. Mach. Intell. **29**(7), 1165–1179 (2007)