Unified Pattern Recognition and its Application to Handwriting Recognition

Mark Venguerov

A thesis submitted for the degree of
Doctor of Philosophy in Computer Science
University of Dublin, Trinity College
Department of Computer Science

April 29th, 2002

Declaration

I declare that the work described in this thesis has not been submitted for a degree at any other university, and that the work is entirely my own.

Signature

Mark Venguerov,

April 29th, 2002

Permission to lend and/or copy

I agree that the library in Trinity College Dublin may lend or copy this thesis upon request.

Signature

MVenguerov

Mark Venguerov

April 29th, 2002

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Pádraig Cunningham. Without his encouragement, patience, and advice this work would never have been completed.

I am very grateful to my wife Svetlana for her support and appreciation.

Abstract

The goal of this thesis is the design and the implementation of a visual pattern recognition system based on the analysis of fundamental principles of human perception. The system must simultaneously be psychologically well founded and computationally tractable. The benefit of this strategy is two-fold: first, it can be used as a model of certain information processing mechanisms in human visual perception. Second, it can be applied to the creation of computer systems, which have performance comparable to that of humans in notoriously difficult domains, e.g. speech perception, unconstrained cursive handwriting recognition, real world scene analysis, etc.

It is not our intention to devise a complete theory of human visual perception. Such an ambitious goal lies well beyond the scope of a single thesis. But we hope that the thesis will serve as a cornerstone of such a theory. Nor is the goal of this thesis to create an industrial quality system performing recognition in the chosen domain. We demonstrate that it is possible to create such a system starting with fundamental principles based on the psychological research of human perception and certain hypotheses about information processing in the human brain. In order to prove computational viability of the model as well as its practical importance, we apply it for the recognition of unconstrained cursive handwriting.

The introductory part of the thesis consists of a review of existing pattern recognition methods, various system architectures (blackboard systems, production rule systems, etc.) bearing close resemblance to our system, and methods used for cursive handwriting recognition. Then we proceed with a review of the psychological research of human visual pattern recognition emphasising those results which can be used as guiding principles of the computer recognition system construction.

The main part of the thesis presents a detailed description of our system including its knowledge base and the recognition algorithms. We proceed with the demonstration of the application of our system to the recognition of cursive handwriting.

The main contributions of the thesis are:

- thorough theoretical analysis of human visual perception principles and their applicability to the construction of computer pattern recognition systems;
- design and implementation of a visual pattern recognition system, which performs simultaneous noise reduction, segmentation, feature extraction, and model matching while starting processing at the level of pixels;
- design and implementation of a new recognition algorithm employing domain-independent heuristics based on the most fundamental *part/whole* and *class/subclass* relations;
- demonstration of the viability of the system in the cursive handwriting recognition domain.

In the concluding part of the thesis we discuss possible technical improvements of the system as well as directions for future research, which can significantly broaden the area of applicability of the system.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

The idea which lead to the emergence of the research area known today as pattern recognition appeared long before even the first computer has been assembled. For quite a while the pages of science fiction books have been populated with robots – artificial creatures capable of seeing the surrounding world, hearing sounds and understanding their master's orders given not through teletypes, as the first computers did, but simply spoken aloud. On the other hand, there were very practical compelling reasons for the emergence of the new discipline, for instance, the necessity of automatic analysis of signals in military radar and sonar surveillance systems. All this lead to the fact that almost simultaneously with the advent of the first computers the methods for solving problems of signal analysis and discovery of objects of interest in those signals started to be developed.

Since its inception pattern recognition has achieved many remarkable successes. Various powerful methods for classification and clustering of patterns in various application areas have been developed. Still, there exist a number of difficult problems defying attempts to solve them.

In this thesis we assume that consistent application of the results of psychological research of human perception, especially in those areas where people demonstrate good performance (e.g. speech perception, real life scene analysis, and cursive handwriting recognition), may help to create computer recognition system achieving human-level performance.

We start with a review of the existing pattern recognition methods, highlighting similarities and inherent problems of those methods and focusing on their computer vision applications. Then we proceed with an overview of various system architectures similar to the architecture of our system. Those architectures come from different domains but share two important features: they work with explicitly structured data representations and they allow simultaneous assessment of conflicting hypotheses.

Since in this thesis we apply our recognition system for the recognition of cursive handwriting, we provide an insight into the specific problems of this area and methods used by researchers to solve them.

Chapter 5 is devoted to an overview of the psychological research of human visual perception. Unlike some other authors we don't focus solely on those areas of the research, which are directly related to the specific implementation of our system (the recognition of hand-written text), but try to take into account all relevant results. These latter include laws of perceptual grouping (Gestalt laws), the use part/whole relation for categorisation and recognition, the structure of perceptual categories, and visual language processing. We try to emphasise those principles, which can be directly used as guidelines in the design of a computer recognition system.

In Chapter 6 we summarise and discuss general design principles of a recognition system following from the review in previous chapters.

Next 3 chapters contain a detailed description of the knowledge representation, the recognition algorithm, and its application to low-level visual processing. The Recognition Knowledge Base is represented in our system as a graph. The recognition algorithm can logically be split into two interacting parts: bottom-up and top-down processes. After the discussion of the bottom-up process, we point out its high computational cost and introduce the top-down process, which uses a number of domain-independent heuristics to improve the performance of the combined algorithm. We briefly compare our architecture with similar architectures reviewed in Chapter 3. Then we show how our system simultaneously performs edge extraction, segmentation, detection of straight-line segments and curved lines, etc.

In Chapter 10 we describe an extension of the Recognition Knowledge Base facilitating recognition of letter parts, letters, and words. We conclude the chapter with a presentation of experimental results obtained on various sample databases.

In the last chapter we discuss limitations of the current system implementation, possible technical improvements, and a few potential directions for future research. One obvious drawback of the system is its inability to learn from experience. We discuss types of knowledge that the system should learn and learning algorithms that can be applied for this purpose. Another interesting area for the future research would be a neuronal implementation of the algorithm. We discuss potential problems with such an implementation. Finally, we provide a summary of the thesis.

The main contributions of the thesis are:

- thorough theoretical analysis of human visual perception principles and their applicability to the construction of computer pattern recognition systems;
- design and implementation of a visual pattern recognition system, which performs simultaneous noise reduction, segmentation, feature extraction, and model matching while starting processing at the level of pixels;
- design and implementation of a new recognition algorithm employing domain-independent heuristics based on the most fundamental relations *part/whole* and *class/subclass*;
- demonstration of the viability of the system in the cursive handwriting recognition domain.

Chapter 2

Pattern Recognition Methods

There exists no single formal definition of the pattern recognition problem. This fact can be related to the relative youth and immaturity of this research area. On the other hand, it can be explained by the fact that pattern recognition stemmed from various practical problems, each of them contributing a definition suitable for that particular problem. Here are some of the definitions:

> "... pattern recognition ... is a vast and explicit endeavor at mechanization of the most fundamental human function of perception and concept formation. " – [115, p. vii]

> "The patterns that we eventually want to study can range up to such abstract and complex entities as the beauty of a woman's face or a piece of chamber music, the profundity of a mathematical theorem or a haiku poem, or the pleasingness of certain smells, at certain times, from certain people." – [109, p. 19]

> "... the notion of pattern – some arbitrarily assigned structure or coherence in a collection of signals. Patterns are not restricted to structure which can be seen, of course; they are eminently handy in dealing with acoustics, linguistics, games, diagnosis and many other ensembles of symbols." – [39]

> "We shall use the term 'pattern' to denote the p-dimensional data vector... whose components x_i are measurements of the features of an object. ... a pattern classifier ..., so that it yields the optimal (in some sense) response for a given pattern. This response is usually an estimate of the class to which the pattern belongs." – [116, p. 2]

> "We are then given an image of an object, or a region in the image containing a single object or a partial view of it (that is, the object may be partially occluded). Given such a region, ... the problem is to identify, for example, to name, the object that gave rise to the image in question." - [111, p. 4].

As we can see even from this short list of definitions of the terms *pattern* and *pattern recognition* range from very specific, tailored to describe specific pattern recognition techniques to very generic, almost philosophical ones. Some authors ([54, p.10]) even claim that absence of the formal definition of the term *pattern recognition* is its main problem.

Although, as it follows from the above definitions, patterns are not limited to one certain modality, in this thesis we deal with patterns that arise in human and computer visual perception. These patterns possess special properties, some of which they inherit from the original input signal called in the case of visual recognition a (digitised) image. Images are 2-dimensional and static (unless motion detection in the sequence of images is of interest, which we don't consider in this thesis). Atomic elements of an image are called picture elements or *pixels*. Depending on the type of the image, pixels can bear various amount of information, e.g. foreground/background Boolean value for pixels in binary images, lightness value in grey-scale images, three values for colours in colour images.

Another term, *object recognition* is often used in the context of computer vision research. To define the term *object* is at least as difficult as to define the term *pattern*. For the purpose of the discussion in this thesis we shall assume that an object is a special type of pattern produced by a real world item with a well-defined and relatively constant shape. Thus, a rabbit is an object (even though as it moves its shape changes), while water in the sea or air are not objects. In English objects are named by count nouns; non-object entities have mass-noun names.

Visual pattern recognition is based on various types of pattern features: shape, colour, texture, location, motion, etc. The main information source is the shape of an object ([111, p.3]). Other sources of information are used as supplementary recognition cues, especially in those cases when the shape based recognition is hampered due to noise or object occlusion in the image.

The problem of visual pattern recognition is difficult. One of the main problems is the fact that the information associated with each individual pixel doesn't bear much relation to the properties of the pattern this pixel is a part of. As stated in [38, p. 2]:

> "The problems inherent in computer vision occur because the units of observation are not units of analysis. A pixel has the properties of position and value. By itself, knowledge of the position and value of a particular pixel almost always conveys no information related to the

> recognition of an object, the description of an object's shape, its position or orientation, the measurement of any distance on the object, or whether the object is defective."

In other words, at the level of pixels the problem of pattern recognition is very underconstrained.

Another difficulty consists in the huge variability of images corresponding to the same object under various illumination conditions, various object and observer mutual positioning, etc.

To solve the latter problem the associative memories approach was suggested. The systems based on this approach store big numbers of varied views of the same object. When an object is to be recognised in an image, this image is compared to all those views and the closest match determines which object is presented to the system. The effectiveness of this scheme depends heavily on the similarity measure used to compare the input image and the images stored in the system. Usually, the similarity measures compare corresponding pixels in images and are very simple, e.g. Hamming distance for binary images or L^2 norm for grey-level images. There is experimental evidence proving that large-scale associative memories play an important role in the animal or even insect visual perception ([112], [22]).

Unfortunately, the associative memory approach doesn't really solve the problem. One deficiency of this approach consists in the very big size of memories necessary to store all possible views even for relatively simple objects. Another problem arises because when simple similarity measures are used, due to various position of the object, various illumination conditions, object occlusion, changing shape, etc. the input image can be much more similar to images of other objects than to images of the same objects under different conditions. This effect is demonstrated in [66].

All methods of pattern recognition are usually divided into three broad groups: statistical pattern recognition, structural pattern recognition, and syntactic pattern recognition. In the following sections we review all these groups paying special attention to their similarities and specific problems of the recognition of visual patterns.

2.1 Statistical Pattern Recognition

Statistical pattern recognition, which historically was the first type of the recognition algorithm, appeared in the early 60s. The basic scheme of this approach is presented in Figure 1 (see [116] for further details).

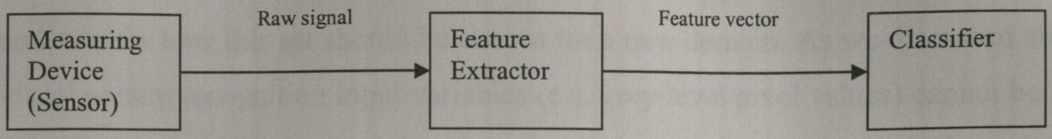


Figure 1 Statistical Pattern Recognition Scheme

The process of recognition starts with a collection of raw sensory data. In the case of visual pattern recognition the input device is usually a digital camera or a scanner. Then the raw data is pre-processed in order to extract information useful for recognition, and get rid of redundant data and noise. This process is also called *feature extraction*. Finally, the set of extracted features is supplied to the classifier, which estimates a probability that the given input data belong to a class or set of classes. This scheme is an oversimplification of the real recognition process, which can contain multiple pre-processing stages, and even multiple classifiers working in parallel.

Sometimes the input signal of a pattern recognition system can be used for classification without any pre-processing. More often, though, the input signal contains significantly more variables than it is necessary for classification. As we pointed out above, this is particularly true for digitised images, where the number of distinct variables (pixels) can reach millions. On the other hand, values of these variables are subject to change when external conditions and not the actual pattern change. Thus, in order to improve classification performance of the system, to lower error rate, and to extract features invariant under a certain task specific set of transformations, feature selection and feature extraction methods are used.

Feature selection means that from the set of *m* input variables we choose *n* variables ($n<m$) without any transformations. The choice is made according to various optimality criteria (see [116], chapter 8 for details).

More often it is necessary to apply a linear or non-linear transformation to the set of input variables in order to receive features suitable for classification. The process is called *feature extraction*. Feature extraction methods are divided into linear (principal component analysis, Karhunen-Loève transformation, factor analysis) and non-linear (e.g. multi-dimensional scaling). Linear feature extraction methods perform linear transformations of input variables based on the mean and standard deviation values of the input data.

The set of features varies from application to application and there is no general rule for how this set should be chosen for a new domain. As we discussed above, visual pattern recognition input variables (e.g. grey-level pixel values) cannot be used directly for classification and feature extraction methods are extremely important in this domain. In section 4.2 we shall see how some of these generic methods are used to extract character recognition specific features.

The set of features obtained at the previous stage is treated as an element of a vector space, where each feature corresponds to a dimension of the space. [116, p.6] gives the following definition of the feature space:

> "Given a set of measurements obtained through observation and represented as a pattern vector $\boldsymbol{x}$, we wish to assign the pattern to one of C possible classes, ω_i, $i = 1,\ldots,C$. A *decision rule* partitions the measurement space into C regions, Ω_i, $i = 1,\ldots,C$. If an observation vector is in Ω_i then it is assumed to belong to class ω_i. Each region may be multiply connected – that is, it may be made up of several disjoint regions. The boundaries between the regions Ω_i are the *decision boundaries* or *decision surfaces*."

One of the simplest classifiers is based on Bayes' theorem for conditional probabilities and is called Bayes decision rule for minimum error. According to this rule a vector $\boldsymbol{x}$ is assigned to the class ω_i if $p(\boldsymbol{x}|\omega_i)p(\omega_i) > p(\boldsymbol{x}|\omega_k)p(\omega_k)$ for $k=1,\ldots,C$ and $k \neq i$. Here $p(\mathrm{A}|\mathrm{B})$ denotes conditional probability of A given B. Methods based on the Bayes decision rule and its modifications are collectively called decision theoretic classification methods. Use of these methods requires knowledge of class-conditional densities, e.g. parameters of normal distributions, which are learned from training data.

An alternative to decision theoretic methods is the group of methods based on the use of discriminant functions. A discriminant function for a two-class problem is a function of pattern $\boldsymbol{x}$, such that if $f(x)<k$ then the pattern belongs to the class ω_1, otherwise – to class ω_2, for some constant k. A discriminant function can be linear or

non-linear. Linear discriminant functions split the feature space into class regions with hyperplanes. An important special case of a linear discriminant function is the nearest-neighbour rule. Decision regions for linear discriminant functions are always convex, which means that they cannot represent solutions of some classification problems.

Multi-layer perceptron (MLP) and radial-basis function (RBF) neural networks (see, for instance, [40]) are among the most important non-linear discriminant function methods. They use weighted sums of non-linear functions of a data vector and a vector of weights scalar products. In the case of MLP this combination is the projection of the data vector to the vector of weights, in the case of RBF it is a vector difference. Other important non-linear discriminant function methods include multivariate adaptive regression splines, alternating conditional expectations, and hinging hyperplanes (see [116] for details).

All the above methods are based on the approximation of an unknown discriminant function by a sum of predefined non-linear functions and inferring parameters of those functions from training data. A classification decision for a given feature vector is made based on the consideration of all features simultaneously. An alternative to this is the decision tree approach, where different subsets of features are used at multiple stages of the decision process. Decision tree based algorithms (for instance, CART [11], ID3 [83]) allow modelling very complex discriminant functions and are used to solve various problems. They efficiently classify new samples and demonstrated good generalisation capabilities.

The relative simplicity of the mathematical model underlying statistical pattern recognition (i.e. finite-dimensional vector space) allows rigorous investigation of properties of recognition problems and their solutions. Therefore, many concepts initially introduced in statistical pattern recognition, such as overfitting, decision region boundaries, locally and globally optimal solutions, etc. are now used in other research areas.

The main shortcoming of statistical pattern recognition methods is their inability to treat the inherent structural nature of recognition problems. In many cases the information about presence or absence of certain features in the input data doesn't suffice for the classification of the data and must be augmented with the information about relationships of those features.

2.2 Structural Pattern Recognition

The basic idea of structural pattern recognition consists in the explicit use of the fact that patterns contain parts that can be recognised easier than the whole pattern. When those parts have been recognised and when recognised parts satisfy certain constraints - we have recognised the whole pattern. Or, as Pavlidis puts it in [79, pp. 3-4]:

> "The basic idea was that *a complex pattern could be described recursively in terms of simpler patterns*. ... In essence, the *structure* of the objects was used for their description."

In [95] the following formal definition of structural descriptions (representations) is provided:

> "A *structural description* D of an object is a pair $D = (P, R)$. $P=\{P_1,\ldots,P_n\}$ is a set of *primitives*, one for each of the n primitive parts of the object. Each primitive P_i is a binary relation $P_i \subseteq A \times V$ where A is a set of possible *attributes* and V is a set of possible *values*. $R = \{PR_1,\ldots,PR_K\}$ is a set of *named N-ary relations* over P. For each k=1,...,K, PR_k is a pair (NR_k, R_k) where NR_k is a name for relation R_k, and for some positive integer M_k, $R_k \subseteq \mathrm{P}^{M}{}_{k}$. Thus, set P represents the parts of an object, and set R represents the interrelationships among the parts."

Based on this definition, structural descriptions of objects are stored in a recognition system. To recognise the object in the input image, primitives are extracted from the image along with their relations. Then this structured description of the image is compared to the descriptions of objects stored in the system (usually called object models). The closest match defines the object recognised in the image.

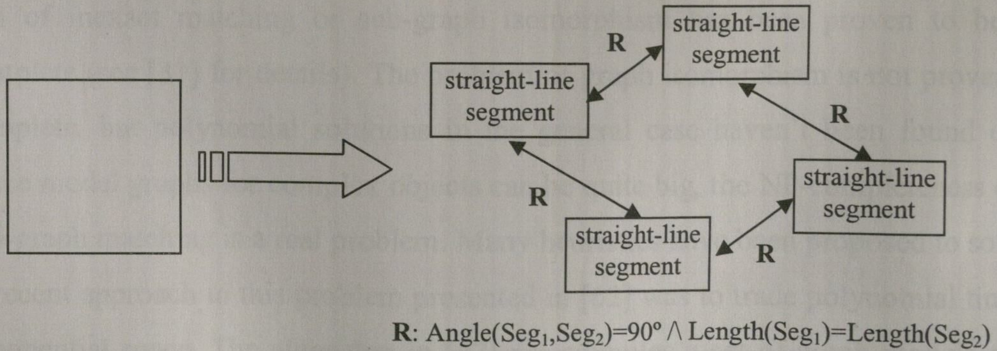


Figure 2 A square and its structural description

Figure 2 above shows an example of an object's structural representation. The object in this case is a square. It consists of 4 primitives – 4 straight-line segments. Not any 4 straight-line segments constitute a square; rather they have to satisfy a set of constraints or, in other words, they have to be members of certain spatial relations. The relation in the case of the square is the same for 4 pairs of adjacent segments and is denoted **R** in the figure. If we change the relation we get a description of a different object (or a class of objects), e.g. if we remove the angle constraint the structural description would describe a set of rhombi.

If we treat the *primitives* and the *relations* of the above definition as individual *features* the similarity of this approach to that described in the previous section becomes obvious. The main difference between statistical and structural pattern recognition consists in the use by the latter of explicitly structured representations for data. This representation usually is a labelled graph with nodes corresponding to *primitives* and edges corresponding to relations between those primitives. Correspondingly, the process of matching the input data to the set of models stored by the recognition system is the process of graph matching.

A structural matching algorithm has to take into account possible distortions of the input data because of noise, object occlusion, object deformation, etc. Therefore, the graph-matching algorithm was extended to perform *inexact matching*, i.e. partial matching of input and model graphs with a quantitative measure of their difference. We consider an example of such an algorithm in section 4.4. Other examples can be found in [95].

The problem of graph matching is its computational intractability. The problem of inexact matching or sub-graph isomorphism has been proven to be NP-complete (see [31] for details). The problem of graph isomorphism is not proven NP-complete, but polynomial solutions in the general case haven't been found either. Since model graphs for complex objects can be quite big, the NP-completeness of the sub-graph matching is a real problem. Many heuristics have been proposed to solve it. A recent approach to this problem presented in [62] was to trade polynomial time for exponential space. The algorithm in [62] pre-compiles a set of graph models of objects into a network allowing polynomial partial comparison between the set of models and an input graph.

The choice of proper primitives in the general case poses the same challenge as the choice of a set of features for statistical pattern recognition.

Structural pattern recognition methods have an obvious advantage over statistical ones in their explicit use of the input data structure. There exist also specific problems in structural pattern recognition. One of them we mentioned above – this is the problem of computational intractability of inexact graph matching. Another problem consists in the separation of the extraction process of primitives and their relations and the process of graph matching. This is an especially serious problem when a big set of relations between primitives is used by a structural recognition system. In this case the system has to calculate all relations between primitives even if they are never used in the subsequent matching process.

2.3 Syntactic Pattern Recognition

As defined in [30] and [29], syntactic pattern recognition is the application of methods of mathematical linguistics to pattern recognition. This approach appeared in the early 60s when researchers noticed similarity between the hierarchical structure of sentences, and the hierarchical structure of object images containing parts and subparts. A well-established theory of formal languages and powerful practical methods of parsing existed by that time in linguistics. The idea of syntactic pattern recognition was to apply this machinery to the recognition of generic patterns.

> "This approach draws an analogy between the (hierarchical, or tree-like) structure of patterns and the syntax of languages. Patterns are specified as being built up out of sub-patterns in various ways of composition, just as phrases and sentences are built up by concatenating words, and words are built up by concatenating characters." – Fu [30].

One of the first attempts to extend linguistic methods to visual pattern recognition was the picture description language (PDL, [96]). Each primitive in PDL has two connection points – head and tail. There are 4 binary operations – '+', '-', '*', '/' – corresponding to the 4 ways two primitives can be combined together. There is one unary operator, '~', reversing head and tail order of a primitive. PDL expressions are made of primitives and operators in the same way as usual algebraic expressions and can contain parentheses. Using PDL's notation pictorial patterns can be described as strings. It is interesting to note that PDL introduces new terminals (unary and binary operators) in order to represent spatial expressions between primitives in a string grammar with just one relation - adjacency. The language has the following major limitation: all connections between primitives and complex patterns are allowed only at two points.

There were a few attempts to overcome PDL's limitations. One of them, the Plex grammar ([27]), allowed multiple connections between elements. The basic element of the plex grammar is a *nape* – an n-attaching point set. Plex structures are created by attaching napes to each other. A plex structure consists of a list of napes, a list of internal connections of napes, and a list of attachment points, which can be used to attach this structure to another plex structure or a nape. The plex grammar suffers from the same problem as PDL: different orderings of primitives produce different description for the same object.

Syntactic pattern recognition methods are particularly suitable for recognition of on-line handwriting, where explicit temporal ordering of elements is known and primitives can be unambiguously mapped into a string grammar. [16] provides an example of such an application.

Generally speaking, all methods in syntactic pattern recognition can be divided into two groups: the methods which try to describe patterns using string grammars, and those ones which are based on multidimensional grammars (i.e. tree grammars, array grammars, graph grammars, etc.). Each of these two approaches has its own merits and drawbacks.

The string grammar approach (illustrated above by the PDL and Plex grammars) tries to introduce new terminals in order to represent multidimensional spatial relations in one-dimensional strings of terms with just one relation – that of adjacency. A big advantage of such a representation is that well-known efficient string parsing methods can be used for recognition. The price to be paid for this efficiency is the high sensitivity of the representation to noise and the lack of a unique representation for a given input pattern.

The approach based on the use of multidimensional grammars doesn't suffer from these problems. Instead, there exists one major problem of parsing. Unlike in the case of string grammars, there are no efficient generic methods for multidimensional grammar parsing.

Another line of research in this area stems from the idea of expansion of existing one-dimensional programming languages to 2 dimensions, which would allow diagram-like programming, description of technical drawings in linguistic terms, etc. This research gave birth to a multitude of grammar definitions and methods for their parsing. An example from this area can be found in [121], [122], and [123]. The authors define a class of grammars, which they call 'Relational Grammars'. Terminal and non-terminal symbols in these grammars are augmented with attributes. Rules in a relational grammar are the same as in normal string grammars, except that their right-hand side parts contain an unordered set of grammar symbols and relations or constraints defined on attributes of those symbols. It turned out that the straightforward application of well-known parsing methods from the domain of string grammars, such as the Early parser or the chart parser ([123], [121]), is possible only in a few cases when severe limitations are imposed on the structure of attributes and relations. In other cases (which are the most interesting from the practical point of view) no computationally tractable methods have been found.

As we could see from this short review of pattern recognition methods, there exists no single universal method capable of solving all problems. Each approach has its own merits and its inherent problems. In this thesis we try to demonstrate that by combining features of different pattern recognition approaches it is possible to improve the performance of a recognition system.

Chapter 3
System Architectures

The systems presented in this chapter come from various research areas but share the same important features: they work with explicitly structured representations (and therefore can be classified as either structural or syntactic recognition systems) and they allow simultaneous testing of conflicting hypotheses. As we shall discuss in Chapter 6, this latter is a very important requirement for a system, which is meant to match human recognition performance.

3.1 Blackboard Architecture: Hearsay II

The blackboard architecture is a particular way of structuring high-level computing systems. Each system based on the blackboard architecture contains three main parts:

- the blackboard – a global data structure, where the information about the current state of the system is stored in the form of individual items;
- the knowledge sources – computational modules, which operate upon the data in the blackboard;
- the control mechanism making decisions about the order of evaluation of individual knowledge sources.

The basic control cycle of a blackboard system consists of the following steps:

- determine which knowledge sources can be executed based on the current state of the blackboard;
- choose knowledge sources (usually just one) to execute;
- execute knowledge sources.

The only way the knowledge sources of a blackboard system communicate with each other is through the contents of the blackboard. A knowledge source can add new items and delete or modify existing items. These actions in turn modify the status of other knowledge sources, i.e. new knowledge sources can be executed with the modified data in the blackboard. The system evaluates the main cycle until one of the three events occurs: a solution is found; no knowledge source can be executed, which means that the system failed to find a solution; a domain dependent job termination condition evaluates to true.

There is no predefined order in which knowledge sources are executed. The computation in the system can proceed in a number of directions creating and testing conflicting hypotheses.

Usually, the blackboard is divided into a few levels, each containing data of the different degree of abstraction.

Hearsay-II is the first and probably the best known and mostly cited blackboard system. The system was developed at Carnegie Mellon University in early 1970s as a part of a speech-understanding project. A detailed description of its architecture and the results obtained by the use of the system can be found in [26] and [70]. The system was followed by a number of successors which all shared the same set of ideas developed in the Hearsay-II project.

The system blackboard consisted of 8 different levels[1], which stored interpretations of the input signal in the hierarchical manner. Data items residing in the blackboard were called nodes and contained AND links connecting parts of interpretations and OR links connecting competing hypotheses.

The control mechanism of Hearsay-II was a priority queue, which contained knowledge source activating records (KSARs). At each cycle the system executed the knowledge source corresponding to the highest priority KSAR. Once a knowledge source became available for execution a new KSAR for it was inserted into the queue. A separate module of the system called the blackboard monitor performed this action.

Blackboard systems are very similar to production systems (see section 3.3 for examples of production systems). In the case of production systems the blackboard is called 'the working memory', knowledge sources correspond to production rules, and the control strategy is called the conflict set resolution strategy. Still, as it is discussed

[1] This levels are sometimes known as *presentation levels*.

in [81, section 4], there are substantial differences between these two types of architectures.

The biggest difference is the granularity level of system knowledge sources. In the case of a production systems all rules share the same 'if … then …' form and therefore are relatively simple. In the case of a blackboard system the knowledge sources are procedures performing potentially complex computations.

The use of blackboard systems has the following benefits: high modularity of the system; simultaneous testing of multiple conflicting hypotheses; ease of adding new knowledge sources facilitating incremental development of a system; smooth integration of the bottom-up and the top-down processing order.

Along with these benefits the blackboard architecture has some disadvantages when compared to traditional computer architectures. A system based on the blackboard architecture is difficult to test, as in most applications the state of the blackboard can be determined by hundreds or thousands of individual items and thus is very difficult to monitor. It is difficult to devise a good control strategy, especially for a novel application (a review of the development of control strategies can be found in [14]). The fact that the architecture is very generic is a benefit, but it also means that there exist no clear guidelines on its applications in new domains.

Blackboard systems are usually less efficient than traditional systems performing the same job, and require higher development effort. And finally, as all communication within those systems is done through the blackboard, the blackboard access synchronisation can become the system bottleneck in parallel multi-processor implementations.

Because of these drawbacks, blackboard systems didn't get much appreciation outside limited application areas. Currently, the research focus shifted from this area to multi-agent systems, which are meant to cure most problems of the blackboard architecture and bring numerous advantages.

More detailed descriptions of the blackboard architecture can be found in [14], [18], [26], and [70]. An example of a successful application of a blackboard system for the recognition of 3-D objects and aerial images is provided in [101] and [102].

3.2 Semantic Networks: ERNEST

A semantic network is a knowledge representation scheme, which stores knowledge in a graph with nodes denoting concepts and labelled edges denoting relations between concepts ([119], Chapter 2). The term 'semantic' is used in this context because semantic networks convey meaning associated with the data stored in them. Semantic networks usually are used in artificial intelligence systems performing symbolic reasoning (e.g. the KL-ONE system, [10]), but have also found their way into the pattern recognition area.

The semantic network system ERNEST (**Er**langen Semantic **N**etwork **S**ystem and **T**ools) was developed at the University of Erlangen-Nürnberg in the mid-80s (see [69], [68], [49], and [91] for details). The system was implemented to treat general problems of image and speech understanding.

The system defines 3 main types of nodes (concept, modified concept, and instance) and 5 types of links (specialisation, part, concrete, model, and instance). Auxiliary information is presented in substructure items and includes attribute, link, relation, modality, value, and function descriptions. A node is a complex data structure consisting of 26 slots. A modified concept is a specialisation of a concept through tighter attribute range restrictions. The link types 'specialisation', 'part', and 'instance' are standard link types in semantic networks. The other two link types were introduced in ERNEST to facilitate connections between representations with different degree of abstraction.

In order to provide task independent control algorithms the system poses certain restrictions on the structure of the network and types of links. Those restrictions enforce partial ordering of nodes along 'part', 'specialisation', and 'concrete' dimensions.

The following three domain independent rules govern the process of concept instantiation (the process of recognition):

- If for a concept A there exist instances of those concepts that are referred by parts or concrete slots in A then build a partial instance of A.
- If a partial instance of concept A exists and there exist instances for context dependent parts of A, which are obligatory parts of A then build a new instance of A.

- If there exist an instance of concept A and an instance of a concept that is optional in the definition of A then build an extended instance of A containing the instance of the optional concept as its part.

Three more similar rules are applied for modified concepts. The rules are independent of any control mechanism and thus don't specify any order of processing.

The system works in the bi-directional mode. The primary process is top-down search expansion of the network nodes along 'part' and 'concrete' links given the goal node. Simultaneously, the system tries to make use of the information about instances available from pre-processing and segmentation by creating modified nodes, i.e. nodes with restricted attribute ranges, in the bottom-up fashion. The algorithm alternates between top-down node expansion and bottom-up modified concept instantiation until a specified level of abstraction has been reached. The direction is determined by the *A** search algorithm.

The system was applied in a few domains of pattern recognition. For instance, it was used as a basis for the development of the aerial image understanding system MOSES ([84]).

3.3 Production Systems: SOAR and Act-R

A production system is a computational system based on the application of 'if-then' rules. A production in this terminology denotes an 'if-then' rule. A production system constantly tries to match items in its short-term memory with the 'if' part of productions stored in long-term memory. When this match succeeds the system 'fires' the production, i.e. it performs the action of the 'then' part. This action can generate one or more new items in the short-term memory. The situation, when more than one production is found that can be 'fired', is called a conflict. Various production systems have various heuristic strategies for conflict resolution allowing them to choose at each step only one production from the set of conflicting productions. Production systems are as powerful as the Universal Turing machine.

The two most advanced production systems developed to date are Act-R ([3], [46]) and SOAR ([51], [67], [46]). The goal of both systems is modelling human cognitive behaviour and, particularly, human problem solving mechanisms. As such, those systems are not directly comparable to our system, but their analysis provides a good insight into the functioning of modern production systems.

The Act-R system performs production condition matching in parallel and chooses one production to 'fire'. Its declarative memory contains a network of declarative memory elements (DMEs). The procedural memory of Act-R contains production rules. DMEs have activation values and associative strengths with other DMEs. The actions of productions modify the declarative memory; the procedural memory is unchanged while system processing. Act-R's main cycle consists of three steps: production instantiation for productions whose conditions match DMEs, selection of a single instantiation using conflict resolution mechanism, evaluation of the production action for the chosen instantiation.

The processing of information by Act-R is goal-oriented. Goals can produce sub-goals. The system maintains a single stack of sub-goals. The goal at the top of the stack is the current goal of the system. It has activation weight 1, which is equally divided between goals slots. A production rule can create a DME representing a new goal. This DME is pushed to the top of the system's goal stack. Only those productions, which match the current goal, can be instantiated by Act-R.

Act-R's conflict resolution strategy is based on the estimation of utility of the production instantiations available at each cycle. During the instantiation step of the main cycle the system needs to decide whether to proceed with instantiations or to 'fire' the best instantiation available so far. Each instantiation has a utility value, calculated as PG-C, associated with it. Here P is the probability to achieve the goal if the production is 'fired', G is the expected utility of the goal, C is the cost of the production action evaluation. Act-R stops to look for new instantiations when the difference between their expected utilities and the expected utility of the current best instantiation is less then the estimated cost of finding this new instantiations. The parameters P and C are adjusted for each rule, based on the actual experience of the system. Act-R's conflict resolution strategy is an example of an *adaptive satisficing (sic!) process*.

The Soar system is a parallel matching, parallel 'firing' system. At each cycle the system 'fires' all the rules, which match the current state of its working memory. Soar's working memory contains only declarative knowledge. Every rule 'fired' in one cycle can either propose an operator, or vote for or against an earlier proposed operator, or modify working memory.

The problem space search approach is the base of Soar's design. Each problem is represented as a space of states with distinguished goal and initial states and a set of operators performing transitions between states. In the main cycle the system first 'fires' rules that propose operators for the current state. Then it 'fires' operator preferential rules. Finally, the system selects the best operator and applies it to the current state. When no operator can be selected, the system generates an *impasse*. For instance, when multiple operators are available, and the system cannot choose the best one, it creates an *operator tie impasse* by putting a new goal in the stack of goals. This represents the problem of operator choice and has its own problem search space. Therefore, the conflict resolution strategy in Soar is based on representing a conflict as new problem to be solved by the system and then applying system's standard problem solving process. Unlike in other production-based systems, the conflict resolution strategy in Soar is completely knowledge-based.

The systems described in this section cannot be compared directly with other recognition systems as they were created for completely different purposes, namely for modelling human cognitive behaviour. Nevertheless, they provide interesting examples of a computer system architecture, which conforms to the requirements we shall discuss in detail in Chapter 6.

3.4 ADIK

Adaptive Drawing Interpretation Kernel (ADIK, [78]) was developed by B. Pasternak at the University of Hamburg for his doctorate dissertation. The system is based on the blackboard architecture and used for the recognition of engineering drawings. It is mentioned as one of the most advanced and promising drawing interpretation systems in [104].

The system input consists of vector data created at a separate vectorisation stage. The output of this stage contains straight-line segments, arcs, and text blocks. The system processes these elements building from them higher-level geometrical constructs: arrows, polygons, circles, connections, etc. These constructs are further combined into meaningful aggregates depending on the specific domain where the system is applied (electronics, electrical engineering, mechanical engineering).

The domain knowledge of the system is stored in its knowledge base consisting of object descriptions. An object description consists of the following items:

- object name;
- a link to the more general object description (super-class);
- a list of named variables – part placeholders;
- a 'trigger' reference to the placeholder that initiates construction;
- a list of geometrical constraints for the parts of this object;
- possible permutations of parts of the object;
- a list of attributes and expressions for their calculation;
- other items necessary for the recognition process control.

```
Name:         Triangle
Variables:    L1 L2 L3 of type Line
Trigger:      L1
Constraints:  NEAR L1.end L2.start
              NEAR L2.end L3.start
              NEAR L3.end L1.start
Parts:        (cyclic L1     L2      L3)
Partnames:            first  second  third
                      line   line    line
```

Figure 3 Triangle definition in ADIK

Figure 3 presents the definition of a triangle object in ADIK. This definition states that a triangle consists of three lines with endpoints adjacent to each other (in certain order).

ADIK uses a rich system of geometric predicates and binary relations to describe constraints of object parts. This system includes predicates specifying position, orientation, dimensions of an object and relations specifying mutual positions, orientations, etc. of end-points of parts. The system is designed in such a way that resulting constraints are position, orientation, and scale (where it is necessary) invariant. Besides, these predicates and relations can be parameterised with constants or ranges.

An important feature of the system is its explicit use of graphic object taxonomies. Once the system recognises a generic object it immediately tries to recognise all its specialisations, thus avoiding multiple recognition of the same object. Taxonomies of the system are based on the single-inheritance mechanism with some special methods helping in those situations where multiple inheritance is necessary.

In order to be able to process input data distorted by noise, the system introduces a set of quantitative and structural tolerances, e.g. distance tolerance, missing part tolerance, broken line tolerance, etc.

ADIK's processing mechanism is based on the blackboard architecture (see section 3.1). The system reads an input file containing pre-processed vectorised data in various CAD-compatible formats. Then some pre-processing including line split, line merge takes places. The system initialises the blackboard with instances of elementary vector data (straight-line segments, arcs, and text-boxes) resulted from this pre-processing stage. The control mechanism of the blackboard is based on fixed priorities assigned to object descriptions so that an object description can be activated only when all parts of this object have been already calculated. This control strategy is equivalent to the breadth-first traversal of the system state space. The only violation of this strict breadth-first strategy is the search for specialisations of a recognised object, which is performed in the depth-first manner.

An elementary step of the recognition algorithm in ADIK consists in the evaluation of an activated object description. A description becomes activated when an instance of the trigger object for this description is bound with its placeholder. The system then consequently checks relations and predicates of the description. If a relation refers to an unbound placeholder, the system retrieves from the blackboard all objects of the type, which can be bound with this placeholder, and then discards those of the objects, which don't satisfy constraints. Eventually, a list of objects corresponding to the activated description is created and added to the blackboard. These new objects are bound to the trigger placeholders of the object descriptions they can be parts of. This bounding in turn activates new descriptions and the process repeats until no new active description exists.

An interesting feature of ADIK is its support for recursive object descriptions. They are used to describe objects, which can contain unspecified number of parts, e.g. generic polygons.

ADIK is a very powerful system for processing of engineering drawings. It allows recognition of drawings from various domains, potentially distorted by noise. Its explicit use of taxonomies of graphical objects makes the knowledge base construction process significantly easier than it is in traditional drawing recognition systems. A potential drawback of the system is the exhaustive manner, which is used to perform recognition. This feature can turn out too computationally expensive, thus making the system non-scalable.

3.5 Parallel Terraced Scan: Letter Spirit

The motivation for the Letter Spirit project ([61], [42], and [43]) significantly differs from other systems described above. The ultimate goal of the project was to model high-level human creativity and its application to design of fonts. An essential part of the system (and the only implemented to date) is a letter recogniser, called by the authors the Examiner.

The Examiner is not designed as a generic recognition system. In order to reduce complexity of the implementation, the system imposes severe restrictions on the type of characters it can recognise: 26 letters of Latin alphabet rendered in so-called grid-fonts. Grid-font letters are produced by a combination of straight-line segments belonging to the grid. Figure 4 shows examples of grid-font letters.

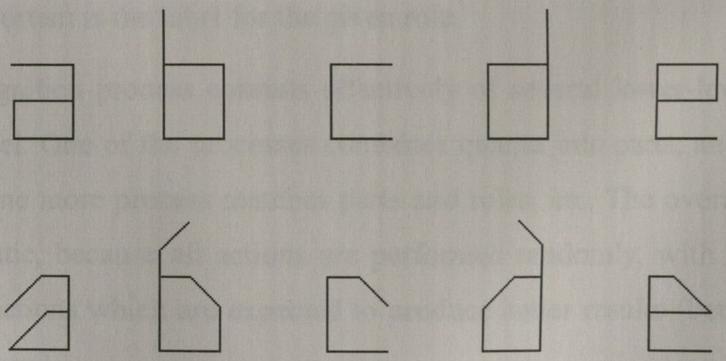


Figure 4 Grid-font letters recognised by the Letter Spirit Examiner

The process of character recognition in the Examiner is called parallel-terraced scan and can be conceived as a combination of simulated annealing with beam search using a highly dynamic heuristic value function.

The process starts when grid segments (called "quanta") are probabilistically merged into parts - sets of several adjacent quanta. The parts are marked with labels coming from the following 7 categories:

- height-width (no_height, very_short, short, medium, tall, very_tall, etc.);
- curviness (closure, straight, concave, bow, etc.);
- weight (light, normal_weight, heavy, huge);
- horizontalness (left, middle, right);
- verticalness (ascender, x-zone, descender, on_baseline, etc.);
- low-resolution view (square1, ..., square12);
- tips (point, ttips, rh-tip, lh-tip, etc.).

26 letters of the alphabet are represented in the system by 66 prototypes called "wholes" or role-sets, with up to 4 prototypes per letter. The prototypes consist of roles - highly abstract descriptions of letter parts shared by all letters - and r-roles, describing relations between roles of a letter. For instance, one of the prototypes of letter 'b' consists of two roles: left post and left bowl with two r-roles, describing how the two roles are combined to produce the letter.

The total number of roles in the system is 47. Roles themselves are represented as collection of norms. The norms describe the labels that a part has to be assigned in order to match a role and also ratings for different labels. The higher the rating the more important is the label for the given role.

The recognition process consists effectively of several lower-level processes running in parallel. One of the processes combines quanta into parts, another assigns labels to parts, one more process matches parts and roles, etc. The overall process is highly probabilistic, because all actions are performed randomly, with higher probability of those actions which are expected to produce better results (better labelling, better matching, etc.).

The Examiner achieved some remarkable success in recognition of grid-font letters (see [61] for details). Still, the authors claim that it should be relatively easy to extend the system so it can recognise usual fonts, leave alone cursive handwriting, has never been proven. Some design decisions of the Examiner are highly specific for the concrete task of the grid-font letter recognition. For instance, the system has only two levels of part-whole hierarchy: quanta are glued into parts, parts are combined to produce letters. While a reasonable simplification for grid-font letters, this rigid treatment of the part-whole relation is not suitable in general case, as we shall see in section 6.1. Besides, the system has a very big number of parameters, which are adjusted by hand. There is no guarantee that such a highly parameterised system can find in general case the optimal solution or any solution at all.

On the first glance the systems considered in this chapter don't have much in common. Some of them, such as Act-R, Soar, and Letter Spirit were not even created for pattern recognition. Nevertheless, the thorough review of those systems, which we provided in this chapter, demonstrates many common internal features. These features include:

- the representation of a system knowledge in the explicit structured form;
- parallel processing of multiple concurrent hypothesis;
- a sophisticated control mechanism choosing next action of the system from a set of possible actions;
- handling of the knowledge and input imprecision and distortion

.

Chapter 4
Handwriting Recognition

The specific area where we shall demonstrate practical application of the general theoretical principles developed in this work is the area of cursive handwriting recognition. The reasons for our choice of this area are two-fold. On one hand, the following features simplify the handwriting recognition problem:

- 2-dimensionality: a text is a 2-dimensional image. This means that difficult problems of restoring 3-dimensional structure from its 2-dimensional image are avoided in this area.
- The image is static – no movement information has to be extracted from the image.
- All the information in the text image is encoded in letter and word shapes, rather than, for instance, colour.
- To extract letter shapes it is usually enough to perform binarisation of the image – therefore the very difficult problem of segmentation object and background is simplified.

On the other hand, as we shall see in section 4.1, due to huge letterform variability in handwriting, letter-part occlusion, very high context-sensitivity, presence of noise, handwriting recognition is a hard *shape recognition* problem, which allows the demonstration of the power of our recognition framework.

Handwriting recognition is the process of conversion of document's digitised images into machine-readable text. This type of recognition is called *off-line* recognition in contrast to *on-line* handwriting recognition, where special devices (digitising tablets) are used for the input of information. The latter is quite a different task and is easier because of the availability of the temporal information. Depending on the type of text to be recognised, this process is usually divided into recognition of machine-printed text, hand-printed text, and hand-written (or cursive script) text. Figure 5 provides examples of these three categories of input. The recognition of machine-printed text is also called optical character recognition (OCR).

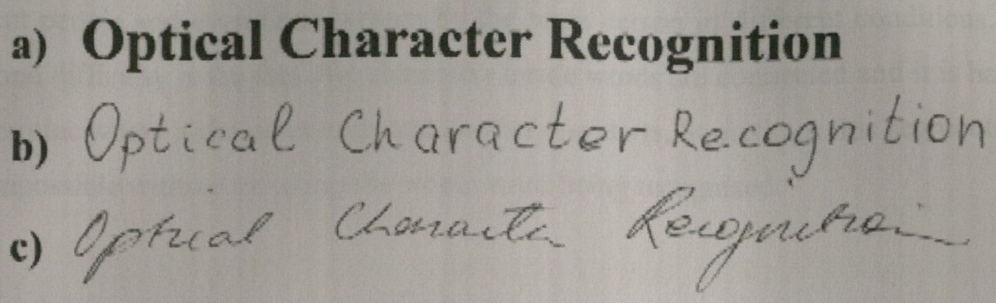

a) Optical Character Recognition
b) Optical Character Recognition
c) Optical Character Recognition

Figure 5 Examples of printed, hand-printed, and hand-written text

The main difference between hand-printed and hand-written text is that in the former characters are separated and therefore the problem of recognition of the text is reduced to the problem of recognition of characters. This is a significant simplification, because the problem of segmentation of words into characters is one of the most difficult problems in the process of recognition of handwriting. With regard to this problem, all recognition methods of cursive handwriting can be divided into *global* (or *holistic*) (section 4.5) and those performing *segmentation* (section 4.3). It is interesting to note, that in the case of Arabic language machine-printed text is also cursive and the segmentation problem arises even in this case. Other reasons for segmentation of characters in a machine-printed text are kerning, which can replace a pair of adjacent characters by one complex character (e.g. ‘fi’ -> ‘fi’), and noise, introduced by a scanning device.

The images processed by character or handwriting recognition systems are usually represented as binary or grey-scale bitmaps. Optical scanners, used to convert paper documents into image data, usually have resolution in the range 200 – 1200 dpi (dots per inch).

A detailed survey of the research in the area of optical character recognition can be found in [65]. Methods used to recognise cursive script are reviewed in [53].

4.1 Difficulties of Cursive Handwriting Recognition

There are two major sources of difficulties in cursive handwriting recognition. The first one is the high variability of the form of characters between texts written by different people and even texts written by the same person in different conditions. The second difficulty is the fact that characters inside words are connected and it is hard to determine where a character ends and another one starts. Sometimes this segmentation is impossible without knowing the whole word being recognised.

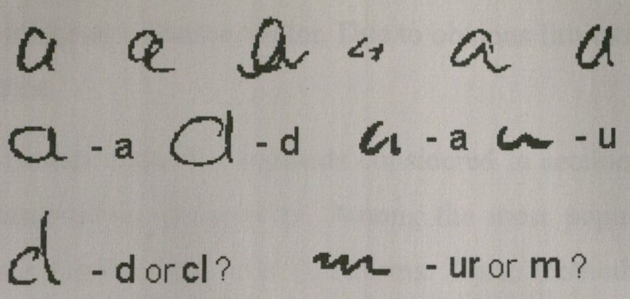


Figure 6 Difficulties of Cursive Handwriting Recognition

Figure 6 illustrates these problems. The first row demonstrates variability of the form of lower-case letter ‘a’. These examples have different size, shape, and even topology (i.e. shape connectedness). The second row demonstrates that hand-written letterforms for different letters sometimes can look extremely similar. Finally, the third row demonstrates the segmentation problem. Without context, judging just by letterforms it is impossible to decide which letter or letter combination was initially meant by the writer.

4.2 Feature Extraction

The feature extraction process is a very important part of traditional character and handwriting recognition systems, both implementing statistical and structural strategies of recognition. As stated in [106]:

> “Selection of a feature extraction method is probably the single most important factor in achieving high recognition performance.”

As we noted in section 4.1, one of the main difficulties of hand-written text recognition is high variability of the character form. Thus, features used for recognition should be invariant in regard to the variability of a single character while providing good separation of different characters. To find such features proved to be not an easy task. Nevertheless, at least invariance in regard to simple transformations, such as scaling, translation, and rotation can be achieved.

Some of the feature extraction methods can be applied to both grey-scale and binary images, while others are representation specific. The simplest method of recognition omits the feature extraction step altogether and performs template matching. it treats the whole image as a feature vector. Due to obvious limitations this method is rarely used in practice.

The linear feature extraction methods considered in section 2.1 are also used in character and handwriting recognition. Among the most popular are Karhunen-Loéve, Fourier, Sine, Cosine, and slant transforms. These methods compute unitary linear transformation of the image reducing the dimensionality of the classification problem without significant loss of information. The features of this group are not rotation or scale invariant, which means that image normalisation (i.e. rotation and size adjustment) are necessary before feature extraction can be performed. These methods can be applied to grey-scale images as well as to binary ones.

The next group of methods extracts features from character contour descriptions. The contours can be easily extracted from binary images or obtained from grey-scale images using edge detection methods ([13]). The methods include zoning, spline approximation, elliptic and other Fourier descriptors. Zoning methods split a character image into rectangular or square zones and calculate the number of lines with a given direction in each zone. A vector of these numbers is used as a feature vector in subsequent classification. The spline approximation method uses parameters of spline curves approximating the contour as a feature vector. Other methods extract Fourier descriptors for closed contours.

Finally, a number of feature extraction methods working with vectorised representation of characters were developed. They include discrete features such as the number of loops, the number of T-joins, the number of intersections, width to height ratio, total number of end-points, the number of crossings with vertical or horizontal axes, etc.

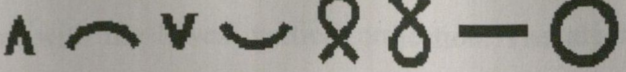


Figure 7 Basic features used in the NHR™ Technology

An interesting set of features was used in [28]. The eight basic features of this method are shown in Figure 7. This method was extended later to a more comprehensive set including 64 features resulting in a very good recognition rate of cursive words ([23], [24]).

4.3 Character Segmentation

Initially, cursive handwriting recognition was conceived as a mere extension of optical character recognition. It was suggested that a cursive handwriting recognition system should segment words into individual letters and then apply well-developed methods of OCR in order to recognise them. It turned out though that the problem of character segmentation in connected words is a hard one. No generic solution has been found for the problem so far. Instead, many heuristics have been proposed ([15], [55]), some of which we consider below.

Cursive word segmentation is usually performed in two steps: pre-segmentation and segmentation. In the first step segmenting algorithms extract pre-segments from words. There are two approaches to pre-segmentation: the first one tries to ensure that each pre-segment contains at most one character, the second one – that each pre-segment contains at least one character. At the second step algorithms process pre-segments so that resulting segments contain exactly one character.

The segmentation algorithm in [9] can segment non-slanted words with fully connected lower contour. The pre-segmentation step of the algorithm finds *potential segmentation points* (PSP). To calculate PSPs the algorithm searches for local minima along the word's lower contour. When a local minimum is found, the algorithm examines its left and right neighbourhoods to find eligible zones for PSPs. A zone is characterised by a continuous sequence of single vertical runs and the density value less than a pre-defined threshold. If a zone is found, a PSP is placed in its middle. Otherwise the algorithm would put a PSP to the point with the lowest vertical projection within a pre-defined limit to the right of the minimum point. Then sequences of PSPs, with the distance between them less than a threshold were replaced with a single PSP located at the point with the lowest vertical projection. The algorithm as presented here could miss segmentation points altogether, so its output was used in a complex recognition algorithm which could compensate for certain segmentation errors.

Another segmentation algorithm, which can serve as a typical representative of the whole class of segmentation algorithms is described in [98]. It is based on the notion of regular and singular features. First, a word skeleton represented as chain graphs is created at the pre-processing step. Then the skeleton is divided into its regular (*axis*) and singular (*tarsi*) parts. The axis of a word is the shortest path in its skeleton from word's left boundary to the right one. The tarsi is the rest of the skeleton. It can include forks, crossings, cusps, and loops. The segmentation is performed by the combination of axis and tarsi features. All these features are stored in the system lexicon for each word the system can recognise.

The methods considered above are also useful in pure OCR systems as accidental character connection happens quite often even in printed texts, especially after low-resolution scanning. It is necessary to emphasise, though, that many years of research in this area didn't produce any reliable universal techniques. There is some evidence that the main problem consists in the separation of segmentation and recognition stages into independent modules – the same problem as in generic computer vision.

4.4 Structural Pattern Recognition of Characters

An example of the application of structural pattern recognition to character recognition can be found in [89], where a method for recognition of multi-font printed characters is suggested. This method is based on the algorithm for direct feature extraction from grey-scale images, which was described in [114]. The algorithm detects two kinds of features: regular (convex arcs and strokes) and singular (branch points, ending points, concave vertices, sharp corners).

Extracted features are used to create a structural representation. Regular features (arcs and strokes) constitute nodes of a graph, while singular features are represented as links between nodes of the graph. Singular features also represent spatial relationships among regular features. Another important feature detected in low-level processing is a gap. Gaps help to associate other features for recognition. They are particularly important in the case of broken lines. Gap filling while graph matching is the only operation changing topology of the graph.

An equivalence relation, *homeomorphism*, is defined on the set of feature graphs. Two graphs are *homeomorphic* if one of them can be obtained from the other by the edge splitting operation, which preserves the topology of the initial graph.

The system contains a dictionary of character prototypes represented as feature graphs, similar to those generated from input images. The only difference is that prototypes are the simplest representatives of equivalence classes induced by the *homeomorphism* relation. So, when matching input graphs to prototypes it is necessary only to check the correspondence between paths in the input graph and edges in the prototype.

The process of inexact matching of input graphs with prototypes is described in terms of graph transformations. A transformation can eliminate a singularity present in an input graph which is absent from a prototype. The following transformations can be used by the system: concave vertex bypass, stroke straightening, stroke into arc conversion, feature insertion or deletion, attribute transformation.

A measure of graph deformation is associated with each transformation. This measure facilitates computation of the overall matching cost as the sum of transformation measures for a sequence of transformations applied to an input graph in order to match it with a prototype. Then the *optimal homeomorphism* is defined as a homeomorphism of an input graph to a prototype with minimal cost over all prototypes and transformation sequences. Finally, an input graph is considered to represent the same character as that prototype, which is optimally homeomorphic with the input graph.

The procedure of inexact matching presented so far is computationally expensive. So, in order to make it more tractable, the following two constraints are introduced:

- geometrical assumption: a matching should preserve the relative order of orientation of the features around two matched joints, and the overall orientation of the features, since rotations near 90° are very unlikely;

- good-continuity assumption: the orientation of a feature is enough to decide locally which feature to choose to extend a path, when the last joint is a branch point.

Notwithstanding the fact that the complexity of the constrained algorithm still is exponential on the number of features in the graphs, its execution time is relatively small, because the number of features or joints in a typical feature graph is usually under 5.

The algorithm has been applied for the recognition of printed numerals from the US Postal Service database of real printed addresses. Number of prototypes was less than 2 per class on the average. The recognition rate achieved 98.13%.

A similar algorithm is applied in [28] to digit recognition. The main difference between these two algorithms is the set of features used to construct graphs describing input characters (see Figure 7) and the set of graph transformations applied to graphs during the process of matching. The rate of recognition of 5-digit images from the NIST special database 1 reported in [28] varies between 61.4% and 83.1%.

Other examples of structural character and handwriting recognition can be found in [1], [9], [16], and [37].

4.5 Holistic Word Recognition

The failure to devise a reliable character segmentation algorithm for cursive handwriting recognition lead to the emergence of global or *holistic* word recognition methods. Some authors (e.g. [35], Chapter 2) argue that cursive handwriting recognition should be performed without recognition of individual letters because this is justified by psychological research of human reading performance. In Chapter 6 we dispute this claim. Nevertheless, the holistic approach has proven its merit in many systems independent of its psychological relevance.

The method described in section 4.4 has been extended in [90] to allow recognition of words without segmentation. The dictionary of the system in this case contains graph representations not for separate characters but for entire words. Graphs for words sharing letters or groups of letters are combined into a net, allowing more efficient matching with multiple candidate words simultaneously.

In [2] the authors present a system which performs recognition of connected characters without segmentation. The system extracts a set of primitives from the image using mathematical morphology methods and then tries to find the best grouping of primitives into characters. The recognition part of the system consists of two modules – the global control module, which at each individual step chooses a primitive from the set of available primitives and passes it to the second module – the matcher. The matcher, given a primitive, tries to find other primitives, which together satisfy a set of spatial constraints for a model of a symbol. The matcher is implemented as a nearest-neighbour classifier. Unfortunately, the authors don't provide any information about the performance of the system.

The holistic approach to recognition of handwriting facilitated some remarkable success. Systems based on word recognition have been implemented for the legal amount recognition in checks and postal address recognition ([24], [23]). Still, some hard problems in this area remain unsolved.

Holistic methods work best when the system dictionary is restricted as is the case in check and postal address processing. In this situation the dictionary filtering can amend even relatively poor performance at the level of feature extraction. A similar effect was observed in the Hearsay-II system (section 3.1), although in a completely different domain of pattern recognition. While the rate of correct recognition of phonemes was about 40%, the rate of correct recognition of words stored in the system achieved about 70% since the system could filter out all ambiguous phoneme combinations not stored in the system dictionary. Adding new words in this case would deteriorate system performance.

Another source of improvement comes from cross-checking of redundant information, e.g. legal and courtesy amounts on checks or city name and ZIP code in addresses. These contextual sources of information are highly domain specific and cannot be relied upon in the general case. Therefore, most holistic recognition methods don't scale up to general-purpose dictionaries.

4.6 Perception-oriented methods

In a recent review of offline cursive handwriting recognition [100] the authors have expanded the traditional classification of handwriting recognition methods by adding a new category, namely, the perception-oriented methods. Unlike all the methods presented above, methods of this group don't work sequentially. Rather, they try to find letters anywhere in the image using bottom-up processing. Then a decision procedure retrieves the best non-overlapping set of letters from all the variants generated at the previous step.

In [100] the authors state:

> "… we find this approach significant as it seems to resemble a good working model, namely the human reading scheme. However, there are only a few methods that prefer this approach, perhaps because of implementation issues."

Based on the alignment of letter prototypes within word images, the method introduced in [25] performs the following steps:

- anchor point extraction;
- stroke detection using affine transformations;
- letter hypotheses generation by matching with letter prototypes;
- instance filtering;
- interpretation;
- application of lexical knowledge.

The work by Côte et al. ([19], [20], [21]) is based on the Interactive Activation Model (IAM)by McClelland and Rumelhart [60], which we shall discuss in more detail in Chapter 5. Unlike IAM, which is purely a theoretical model of human perception of printed words, the system PERCEPTO presented in [21] was designed to perform recognition of cursive handwritten words. Therefore, it introduces a few new mechanisms, primarily, in order to overcome major practical limitations of IAM. These mechanisms include pre-processing, baseline extraction, and feature extraction. The system is based on the use of pre-segmentation, since its basic algorithm requires the knowledge (if even approximate) of letter positions within the word. Besides, the system uses extensive hard-wired knowledge of specific elements of handwritten words, such as ascenders and descenders.

In the following chapters we shall see that the application of our recognition framework to handwriting recognition according to the classification of this chapter falls exactly into this category, namely, perception-oriented methods. It is not surprising, as the main motivation for our work, as well as for the approaches presented in this section, is to try to model the human visual perceptual mechanism.

The methods we presented in this chapter (with the exception of the perception-oriented group) can be considered as 'standard'. They apply generic pattern recognition approaches: statistical, structural, and syntactic to the specific problem of character and handwriting recognition. Therefore, the main focus of these methods is on the proper choice of features for subsequent classification using statistical pattern recognition techniques considered in the section 2.1. Alternatively, if the structural pattern recognition approach (section 2.2) is used, those methods focus on the choice of primitives and relations, whereas the matching procedure is given significantly less attention.

In the last section of this chapter we look at the new, fast growing family of perception-oriented methods. The methods in this group explicitly try to use results of psychological research, but unlike our generic framework are designed only for handwriting recognition.

In contrast to cursive handwriting recognition the problem of optical character recognition can be considered as solved – numerous OCR packages with almost 100% recognition rate on standard fonts are available commercially. It is not true, though, for arbitrary fonts. For instance, most decorative fonts of the type presented at [43, p. 413] are still well beyond the capabilities of contemporary OCR systems.

Chapter 5
Human Visual Perception

In this chapter we review psychological research of human visual perception. This review will concentrate mostly on general principles of human perception organisation discovered by psychologists, rather than on particular models of human vision. In the following chapters we shall use these principles as guidelines for the design of a computer pattern recognition system.

The chapter starts with the formulation and the analysis of the problem of perceptual grouping. We provide a list of main grouping principles discovered by Gestalt psychologists and point out the problems of the classical Gestalt approach. We review contemporary efforts to solve those problems within theories of region segmentation.

Then we switch our attention to theories of shape representation in human visual processing. We consider strengths and weaknesses of those theories.

The following section is devoted to the analysis of theories of object categorisation. We present the two most popular theories: Recognition by Components and Multiple Views and discuss their strengths and weaknesses.

Since the specific area of application of our generic recognition framework in this thesis is recognition of handwriting, we also review psychological research of visual language perception.

It is important to emphasise that we don't hope to provide in this chapter a comprehensive review of such a vast area of scientific research as psychology of visual perception[2]. Our review is with necessity very selective omitting, for instance, perception of motion, stereo matching, perception of colour and depth, etc.

[2] A relatively concise overview of the area [8] compiled more than a decade ago consists of two weighty large-format volumes!

5.1 Perceptual Grouping

The problem of perceptual grouping was first formulated by Gestalt psychologists in 20th century. The question they tried to answer is: how can people perceive objects with well-defined shape rather than a chaotic array of coloured pixels produced by the retinal receptors. There must be a way for the human perceptual system to combine those pixels into meaningful object descriptions. Moreover, this process must be a pretty unique one since of all the myriads of possible pixel arrangements we usually perceive only one. The study by Gestalt psychologists of this problem revealed a number of general principles of grouping ("laws of grouping" in Gestalt terminology) followed by the human perceptual system. The set of principles varies from a researcher to a researcher, so here we list only those ones upon which everybody agrees. The principles are:

- *proximity*: close visual elements tend to be grouped together;
- *colour similarity*;
- *size similarity*;
- *orientation similarity*;
- *common fate* principle: all else being equal, elements moving synchronously tend to be grouped together;
- *good continuation* of lines and edges: all else being equal, elements that can be seen as smooth continuation of each other tend to be grouped together;
- *closure*: all else being equal, elements forming a closed figure tend to be grouped together.

Most principles of perceptual grouping can be considered as the result of the adaptation of the human visual system to conditions of the real world we live in. For instance, the good continuation principle is valid only because most real-life object boundaries are relatively smooth. We can imagine a world in which object boundaries are fractal-like at any scale. In such a world the good continuation principle is meaningless and visual perception would probably not use it.

The hypothesis of the adaptational nature of at least some perceptual grouping principles is also supported by the study of perceptual grouping development in babies. This study was performed by Kellman and Spelke (see [48]) using the so-called habituation paradigm. It turned out that 4-month old babies rely on the common fate grouping principle and practically ignore static grouping principles, such as the good continuation principle[3]. The ability to use the latter principle either is learned or develops spontaneously between the 5th and the 7th month of the baby's life ([99]). Although there doesn't exist a complete theory that explains all the empirical facts in this area, it seems that babies are born with *some* mechanisms of perceptual grouping, but most are learned later.

When we consider the definition of Gestalt grouping principles, the following question arises immediately: where do elements, referred by the above principles, come from? The visual elements used in Gestalt psychologists' experiments were by no means pixels, but rather small circles, squares, objects of arbitrary shape, etc. So, to put the above question is other words: why are those elements perceived as units in the first place?

To answer this question without completely abandoning Gestalt principles of grouping, Palmer and Rock [74] came up with a new grouping principle, which they called *uniform connectedness*. This principles states that connected regions of uniform image properties (luminance, colour, texture, motion, binocular disparity) are perceived as initial units of perceptual organisation, i.e. the *elements* of other grouping principles.

The process of dividing an image into areas defined by the uniformity of a certain image property is called *region segmentation*. Region segmentation can be performed either by finding region boundaries (boundary-based approaches) or by 'growing' regions based on neighbourhood relation between pixels with uniform property (region-based approaches).

The most common way of finding region boundaries in boundary-based approaches is application of edge-detecting operators, e.g. the Marr-Hildreth edge detector ([58]). This edge detector finds pixels with zero-crossings of luminance second derivatives. Those pixels constitute closed contours used to partition an image into regions of roughly constant luminance. Unfortunately, this edge detector, as well as all other edge detectors, tends to produce segmentations of real-life images, which

[3] i.e. the babies perceived as a whole two visually disconnected parts of a moving object (when the middle part was obstructed), but when the object was static those parts were perceived as separate objects.

don't correspond to intuitive segmentation performed by the human visual system. Parts of the same object are segmented into different regions, whereas pieces of different objects are combined into one region if they have similar luminance. This behaviour is often a consequence of light and surface interaction, resulting in shadows and highlights. Sometimes good region segmentation can be achieved by procedures using texture-based techniques.

5.2 Perception of Parts

Most complex objects are perceived to be composed of *parts* – portions of the object, which are perceived as objects themselves. In order to form an object, its parts have to satisfy a set of spatial relations, i.e. to take certain positions relative to each other. Psychological evidence of part perception comes in the form of linguistic evidence, phenomenological evidence, and perceptual experiments. Linguistic evidence is the most straightforward one – when we talk about complex objects, we tend to describe them as consisting of certain parts (e.g. a human body consisting of a head, torso, arms, etc.).

A number of phenomenological demonstrations provided in [41] suggest that even smooth and continuous objects are perceived to contain parts.

And finally, in [71] and [72] Palmer provides experimental evidence of part perception. The stimuli in his experiments were 2-D nonsense figures. Those figures couldn't be described in any reasonable way, nevertheless, they were perceived as having well-defined part structures.

There are two basic theories of the way human visual perception divides objects into parts (*parses* objects). The *shape primitive* theory states that there exists a set of basic undividable shapes. Any object can be divided into parts, which are represented by primitive shapes. This approach is based on an analogy between visual perception and languages, where words are constructed as strings made of a small set of letters.

One of the problems of this theory is that it doesn't take into account contextual effects in part segmentation. Experiments (e.g. [72]) demonstrated that the same part could be easily detected in one context and not in another. Another problem is related to our ability to perceive parts to contain sub-parts. This means that a theory of part segmentation has to explain complex *part/whole hierarchies*. The problem can be solved within the *primitive shapes* theory by the introduction of *multiple scales* of description. Then the same primitive shape can play different roles at different scales. Finally, despite certain claims (e.g. [4]), nobody has so far devised a set of primitive shapes (to say nothing about a *small* set of primitive shapes!) capable of representing the huge variety of real-life object shapes.

The *boundary rules* theory ([41]) assumes that objects can be divided into parts in certain locations specified by a set of universal rules. The authors of this theory suggested the following rules:

- the concave discontinuity rule: the visual system divides an object into parts at abrupt changes in surface orientation towards the interior of the object (i.e. concave surface discontinuities);
- the deep concavity rule: the visual system divides an object into parts at negative extrema of the object surface curvature.

Although this theory exhibits a number of difficulties, it is very popular in the computer vision community as it can be easily implemented as a computer algorithm. The first difficulty of the theory is that it specifies where to choose part dissection points, but doesn't specify which points to connect.

More profound difficulty stems from the fact that people tend to perceive parts even where no concavities or discontinuities exist at all. An example of such an object is an egg. Although its shape normally is very smooth and doesn't have any extrema of negative curvature, people tend to distinguish the large end of an egg and the small end[4].

5.3 Theories of Shape Representation

In [73, p. 363] Palmer states:

> "Of all the properties we perceive about objects, shape is probably the most important. Its significance derives from the fact that it is the most informative visible property in the following sense: Shape allows a

[4] On one occasion this part separation even lead to a war! See: Jonathan Swift, Gulliver's Travels. W.W. Norton & Company Inc. New York, London, 2002, pp. 40-41 (originally published in 1726).

> perceiver to predict more facts about an object than any other property. ... It is also the most complex."

This observation emphasizes the difficulty and practical importance of understanding shape representation in the human visual perception system. Unfortunately, no satisfactory solution of this problem has been found so far, a solution that would capture all the power and versatility of human shape perception. Below we review a few different theories (or, rather, types of theories) that more or less adequately explain certain aspects of shape perception.

5.3.1 Templates

Template theories were inspired by the associative memory approach in computer recognition systems, which we mentioned in Chapter 2. The recognition process according to a template theory is the process of consequent matching of the input image with the set of stored template images. The measure of similarity between the input image and a template is simply the ratio of the number of pixels with corresponding colours to the total number of pixels in the image. The more pixels in the input image and the template image coincide (in other words, the more is the 'overlap'), the higher is their similarity. After all similarity values for all templates are computed, decision-theoretic methods are used to classify the input image as an instance of one of the categories. This process of matching is neither scale, nor rotation, nor translation invariant and thus requires a pre-processing normalisation stage.

Considering the cellular structure of retina, we can assume that templates are used at the lowest level of human vision, as they provide the only means to extract initial information about primitive shapes such as lines and edges from the iconic image created be the eye.

Template theories have never been seriously considered as *general theories* of human perception as they exhibit the same set of problems as the associative memory approach in pattern recognition. A more detailed account for template theories can be found in [109].

5.3.2 Feature Theories

Feature theories appeared as an alternative to template theories. They suggest that a set of *n* tests is applied to the image. The results of these tests constitute either a set of *n* Boolean values or a vector in an *n*-dimensional linear space. A decision-theoretic method is then used to classify the input image. Psychological feature theories were inspired by the development of statistical pattern recognition, as a comparison with the description of the latter in section 2.1 reveals.

There are two main types of feature theories - theories based on binary features and theories with continuous values of features. In the first case the categorisation of an input image is done on the basis of similarity between the set of binary features of the image and the stored feature representations of categories. The similarity measure in such a theory is a function of the number of common features and the numbers of features present in the image, but not in the stored representation and vice versa. The most popular similarity measure of this type was introduced in [107].

In the second case the set of feature measurements is treated as a vector of an *n*-dimensional linear space. Then standard methods based on the use of discriminant functions (described in section 2.1) perform image categorisation.

Similar to statistical pattern recognition methods, feature theories are criticised on the basis that they cannot represent structural relationships between features. This is not quite true, as a feature, extracted from an image, can be virtually anything measurable in the image, including the Boolean value corresponding to a given relation between other features. Thus, at least in theory, a set of features can be extended to reflect structural relationships between features. Unfortunately, the size of the feature set in this case grows exponentially with the growth of the number of non-relational features and the number of relations.

5.3.3 Structural Description Theories

The next group of theories state that images are stored in the form of their structural descriptions, i.e. descriptions of parts and spatial relations between them. Structural descriptions are a very powerful method of visual information storage and they overcome many problems of template and feature-based descriptions. We discussed structural descriptions in section 2.2.

Unfortunately, there is a price to be paid for all the benefits of structural descriptions: as we saw in section 2.2, matching structural descriptions is a computationally expensive (intractable) process. Besides, in order to be able to represent subtle differences between objects, for instance, in the case of face recognition, a sufficiently rich set of parts and relations has to be provided. Finally, those parts and relations have to be computable from real-life images and this turns out be a rather difficult problem itself.

In Chapter 6 we shall discuss how these problems are addressed in our recognition framework.

5.4 Theories of Object Categorisation

Recognition of an object is a two-fold process. On one hand, we can talk about recognition of the object identity (e.g. is this John?); on the other hand, object recognition can be considered as object *categorisation* (e.g. is this object a human being?). Object categorisation is a very important process, which allows determining object function and utility. Palmer ([73], p. 413) distinguishes four components of object categorisation:

- object representation;
- category representation;
- comparison process;
- decision process.

Any model of object categorisation must specify all four components. Before presenting the two most influential theories of human object categorisation below, we shall discuss the nature of perceptual categories.

5.4.1 Perceptual Categories

The most important fact about categories (including perceptual categories) is that they constitute a hierarchy. Any category can be a sub-category of a more general category (e.g. the category of black cats is a sub-category of the category of cats). Sometimes it is assumed that categories form a tree, that is, each category has just one more general category. In reality, categories form a structure known in mathematics as lattice. The above example of category of black cats can demonstrate it. On one hand, this category is a sub-category of the category of all cats. On the other hand, it is also a sub-category of the category of all black animals. Both these categories are sub-categories of the category of all animals. This is an example of a (very simple) lattice.

Since the classical analysis of the categorical structure provided by Aristotle, categories are defined by their corresponding lists of constraints that an object has to conform to, in order to belong to the category. These constraints are called necessary and sufficient conditions of the category. This definition still dominates the mainstream mathematics and mathematical logics. And for many years it was assumed that this is the way categories are represented in human minds.

Experimental studies by the psychologist Eleanor Rosch ([85], [86], and [87]), showed that all natural perceptual categories might be structured around one distinguished exemplar, called a prototype. The prototype view of category formation differs from the classical one in three aspects:

- rule-based vs. instance based representation;
- binary vs. fuzzy membership;
- different degree of 'typicality' of objects.

The first aspect means that that a category is based on a certain measure of similarity of a given object to the prototype object of the category and is represented in memory not by the list of constraints, but rather by the prototype. The second aspect denoted the fact that objects can belong to the category to a certain degree. And finally, even among clear (100%) members of a category some are considered as more typical representatives of the category and some – as less.

Another important study performed by Rosch sought to answer the question: at what level of the categorical hierarchy categorisation initially happens? It turned out that most people initially recognise objects at some intermediate level of the hierarchy. Rosch called categories at this level *basic-level categories*. More abstract categories are called superordinate categories, more specific – subordinate categories.

Further studies ([47]) demonstrated effects consistent with the claim that typical members of a basic category are first classified to this category, whereas atypical ones first classified to one of its subordinate categories. Those subordinate categories got the name *entry-level categories.*

Some terminological confusion stems from the fact that perceptual classification to the entry-level categories is called sometimes object recognition, which makes it difficult to distinguish from recognition of object identity. For instance, in the context of reading we speak of 'recognition of letters and words'. This is a typical example of entry-level categorisation.[5]

Unfortunately, the prototype theory of object categorisation is not free of problems. The basic problem lies in the fact that category membership is specified in terms of some *similarity measure* between the category prototype and an object. In the context of experiments it was relatively easy to specify this measure, but it is much more difficult, if possible at all, in general case. Originally Rosch provided three basic criteria of basic-level category formation: shape similarity, motion similarity and common attributes. But what is shape similarity, to start with? Another problem consists in the fact that in the categorisation process people tend to take into account attribute relations, such as higher variability of some attributes relative to others. Finally, the prototype theory does not answer the question of what makes a category natural.

The two theories of object categorisation presented in the following sections provide computational models of object categorisation. The first theory is based on the assumption that categorisation is performed by perceiving object parts and spatial relations between them. The second one uses an object and the category prototype alignment.

5.4.2 Recognition by Components

The Recognition-by-Components (RBC) theory was developed by Biederman ([4]). The theory is based on the following three observations of basic phenomena of human visual recognition [4, p.117]:

> "1. Access to the mental representation of an object should not be dependent on absolute judgments of quantitative detail, because such judgments are slow and error prone....
>
> "2. The information that is the basis of recognition should be relatively invariant with respect to orientation and modest degradation.

[5] How often are you interested in recognising that specific letter 'L' from the second line of the third page of your diary? This is an example of the 'true' letter recognition rather than categorisation.

> "3. Partial matches should be computable. A theory of object interpretation should have some principled means for computing a match for occluded, partial, or new exemplars of a given category."

RBC postulates that 3-D shapes are represented internally by a small set (usually 36) of primitive shapes called 'geons', which can have qualitative attributes. The set includes such simple shapes as boxes, cylinders, and wedges. Geons are claimed to be easily detectable in images due to their non-accidental properties, i.e. properties which are almost always (except in the case of an 'accidental' viewpoint) present in a 2-D projection of a shape. Non-accidental properties include collinearity of point or lines, curviliniarity of points or arcs, symmetry, parallelism of curves, and presence of vertices.

As a representative of the class of structural description theories RBC is not a novel theory. Similar theories were developed earlier by Marr and Nishimara ([59], [58]), Binford ([5]), Tversky and Hemenway ([108]). The essence of RBC is the specification of a small set of primitives (geons) and the claim that an arbitrary 3-D shape can be represented as a hierarchical structural composition thereof.

One of the drawbacks of the theory is its reliance on purely qualitative distinctions between shapes. Quantitative (metric) information is important in recognition of particular objects, e.g. distinguishing one face from another.

A more severe problem (which is faced by all structural description theories) is the need for reliable detection of parts in images. There are many ways in which a set of edges and edge junctions detected in a grey-level image can be matched to particular geons. This problem precluded application of RBC to recognition of objects in real life images. Another serious problem is instability of structural representations, i.e. existence of multiple structural representations for the same object, depending on the set of primitives chosen to describe it.

In [44] the authors present a neural network-based implementation of the model.

5.4.3 Multiple Views Theories

Recently, a new family of psychological theories of object recognition emerged, which is supposed to solve problems of classical structural theories, such as RBC.

This is the class of so-called 'multiple 2-D views' theories, which claim that structural 3-D information is not stored by the human visual perception system. Rather, it stores multiple 2-D views of the same object, which can be either templates or structural 2-D descriptions, depending on a particular theory. An example of such a theory can be found in [12] and [103].

5.5 Visual Language Perception

Visual language perception or, in other words, recognition of words and letters in the process of reading is an important special area of psychological research of visual perception. Its practical importance is related to the very important role of reading in contemporary society. Reading is much more complex process than the process of recognition of words as its final result is the meaning of text. In the following sections we will not deal with all the aspects of reading but rather focus on the first stage of the process – recognition of word forms.

5.5.1 Writing Systems

All existing writing systems can be divided into four major classes: *semasiographies*, *logographies*, *syllabaries*, and *alphabets* ([32, chapter 6], [33]).

The international system of road signs can serve as an example of a *semasiography*. Distinguishing features of this type of a writing system are lack of phonetic information associated with symbols and their propositional character.

Logographies are best represented by the Chinese hieroglyphic system. Hieroglyphs contain both semantic information (at the word level) and restricted phonetic (based on similarity of pronunciation) information.

Unlike in *semasiographies* and *logographies*, symbols in *syllabaries* and *alphabets* don't have any meaning associated with them and only represent sounds or pronunciation. The difference between *syllabaries* and *alphabets* lies in the type of phonetic information associated with their symbols. In *syllabaries* the phonetic information is represented at the level of the syllable, in alphabets – at the level of the phoneme. The Japanese Katakana system consisting of 74 symbols is an example of a *syllabary*. Writing systems of such languages as Spanish or Russian are pure *alphabets*.

Some contemporary Western languages (e.g. English, French) cannot be described as pure alphabetical systems. Since the spelling of many words significantly differs from their contemporary pronunciation and, therefore, conveys only partial (if any at all) phonetic information, the written form of these words should be considered as a kind of hieroglyph[6]. Thus, the writing system of these languages has features of both logographies and alphabets.

An essential factor for the recognition of language graphical symbols is the manner in which the symbols are combined visually. In case of logographies they just stand next to each other without explicit connection or overlapping. In case of contemporary alphabetic writing systems the symbols (letters) constituting one word and written by hand or even printed (in the case of the Arabic language) are usually connected, with words being separated from each other by some space. This feature makes the recognition of cursive text particularly difficult, as it was demonstrated in section 4.1.

5.5.2 Multimodal Visual Language Processing

On the level of the human nervous system, there exists a specialised mechanism for processing of auditory language called Wernicke's area. This can probably be explained by the long evolutionary adaptation of the brain to speech perception. There is significantly less evidence for any biological specialisation related to reading and writing, which is quite understandable from the evolutionary point of view – the visual form of language is a relatively modern invention. This fact suggests that proper study of the human reading process should take into account interaction between purely visual processing of visual language and the mechanisms of phonological and semantic code formation.

There are two main differences of the auditory and visual information processing in the human brain. The first one related to the fact that signals from eyes and ears come to different regions of the brain. The second difference is in the nature of signals coming from external sensors – eyes and ears. The information, conveyed through the visual path, is static, whereas auditory information is a temporally arranged sequence.

[6] Naturally, this does not mean that separate letters contain any semantics, unlike parts of real hieroglyphs.

5.5.3 Word Superiority Effect

The empirically found contextual dependence of performance in recognition of letters is called the word superiority effect. It turns out that the letters can be reported more easily when they appear in meaningful words rather than in arbitrary letter strings. This effect was first reported by Cattell in 1886. Unfortunately, his experiments didn't allow for separations of effects in recognition performance vs. memorisation performance.

The definitive study was done almost a century later by Gerald Reichert. In 1969 he performed modified experiments, which undeniably demonstrated the difference in the recognition performance, namely, that letters in words are identified more accurately than letters in non-words. This effect is also called the word-nonword effect.

In a different experiment set-up Reichert also showed that letters within words are recognised more accurately than single letters. This effect is called the word-letter effect. Finally, it turned out that even non-words which are easily pronounceable, improve the recognition accuracy. This is the pseudo-word superiority effect.

It is worth noting that similar *object superiority* effects were reported for generic objects ([117]).

All these effects demonstrate a very important feature of the human visual perception: categorization of letters in words (as well as objects in scenes in more general case) is highly context dependent.

5.5.4 The Interactive Activation Model

The Interactive Activation Model is one of the most influential *computational models* of human word reading. It was introduced by McClelland and Rumelhart in 1981 ([60]). The goal of this model is to simulate perceptual processes responsible for word and letter categorization in human visual perception, and, specifically, provide an explanation of the word superiority effect (see section 5.5.3).

Recognition of words in the IAM is performed by a 3-layer neural network. The bottom layer of the network corresponds to letter features, the middle layer – to whole letters and finally, the top layer – to words. The model can recognise 4-letter words consisting of grid letters similar to those used in the Letter Spirit project (see

Figure 4). All letters are constructed as sets of 12 grid segments. The recognition network contains a node at the bottom layer for each grid segment in each of 4 possible letter positions. Thus, the bottom layer consists of 48 nodes. These nodes are activated when the corresponding grid segment is present in the letter at corresponding position in the input word. The feature level has no feedback from upper layers.

The next layer consists of 104 nodes, each representing one of 26 letters in one of 4 possible positions. The letter nodes are connected by excitatory connections with the segment nodes at the bottom layer present in the letter and by inhibitory with those, which are absent. All letter nodes corresponding to the same position are connected with inhibitory connections, which makes it a winner-takes-all type network, allowing only one letter node to be active at each position one the activation pattern has settled.

The topmost layer contains nodes corresponding to more than 1000 4-letter words. They are connected with inhibitory and excitatory connections with letter nodes at the previous layer depending on the presence or absence of the corresponding letter at the corresponding position in the word. The word nodes are interconnected with inhibitory connections similar to letter nodes in the previous layer. Besides, there exist 'word feedback' connections with letter nodes. These connections allow modelling of context-related aspects of human perception.

The model correctly simulates a number of contextual effects, such as the word superiority effect, the pseudo-word superiority effect, and the word-letter effect.

A detailed survey of psychological models of human word recognition can be found in [45].

In this chapter we considered the following important features of human visual perception:

- Gestalt principles of perceptual grouping;
- properties of shape representation and theories explaining them;
- use of parts for shape representation and object categorisation;
- structure of perceptual categories;
- contextual sensitivity;
- specific effects and models of word recognition.

Psychological research of human visual perception is a vast and quickly developing research area. We didn't hope to provide a comprehensive review of this area (it can be found in [8] and [73]), but rather, we have chosen those specific lines of research, which we perceived as mostly relevant to our goal – creation a computer pattern recognition system based on the same principles as human pattern recognition. This choice is with necessity a subjective one, but we hope to have managed to include into this review all the important topics.

Chapter 6
General Principles of Design

In this chapter we shall present general design principles of a pattern recognition system, which we derive from the results of the psychological research of human visual perception reviewed in the previous chapter. Another source of design requirements is the analysis of problems of the recognition systems reviewed in Chapter 2.

We start the chapter with the formulation of the thesis' main conjecture. Then we describe the system's simplest objects – pixels – and their categorisation performed by template matching. We proceed with the discussion of the following concepts underlying our recognition framework: the *abstraction* relation of categories, the *part/whole* relation and the *combination* operation, the categorisation process, and properties of attributes and constraints.

Other elements of design presented in this chapter are parallel multiple hypotheses testing, contextual sensitivity, the system knowledge representation, and the ability of the system to learn new information.

Some of the ideas presented in this chapter were introduced in [113].

6.1 The Main Conjecture

Comparing Chapter 2 and Chapter 5 one can notice an undeniable liaison between psychology of visual perception and pattern recognition research. Historically, these two areas of scientific research borrowed ideas and theories from each other. Many psychological theories, especially those dealing with low-level visual processing, claim that the human brain processes perceptual data exactly in the same way and uses the same data structures as some specific pattern recognition algorithm. In turn, the behaviour of computer pattern recognition algorithms often is described in psychological terms. Following this tradition we shall formulate the main conjecture of this thesis, which is equally useful as a (yet another) psychological theory of perceptual categorisation, and as a guideline for the design and implementation of a pattern recognition computer algorithm. In the following chapters we will not discuss psychological implications of our assumptions, but rather concentrate on the practical issues of the algorithm implementation.

Before formulating the conjecture, let us recall that one of the most important functions of the visual perception system is categorisation of objects[7]. Most theories define categorisation as a two-step process. On the first step some functional modules (different from theory to theory) extract information describing an object from the raw image; on the second step this information is used for categorisation.

For instance, in the case of RBC (section 5.4.2), the first step consists of the extraction of primitive shapes (geons) and their spatial relations from the image. On the next step this structural information is matched against prototype models representing various categories. In the case of feature-based theories, the first step consists in the extraction of a vector of features from the image, whereas at the second step a classification algorithm chooses the best-matching category for the object represented by the vector of features.

In the contrast to the above, we conjecture that **in the actual algorithm underlying the functioning of the perceptual system these two steps are not separated and constitute the indivisible body of the algorithm. The algorithm performs categorisation as well as it builds complex objects from their constituting parts at each conceptual step of recognition.**

This conjecture has profound consequences for the design of the algorithm, but before we proceed with their analysis, we shall consider some theoretical implications of the conjecture.

First of all, our theory, being obviously a structural representation type theory, goes further in this direction than most theories of the class. The only unstructured elements in our theory are pixels. All other objects are made of parts, combined in a hierarchical manner. We claim that even objects, which are seemingly perceptually indivisible (such as a circle), consist of parts (in the case of circle - arcs). Therefore,

[7] In certain sense, even the dual problem of object identification can be expressed as a special case of categorisation, where the category of interest consists of just one element – the object itself.

we can state that our recognition framework is based on a consistent *mereological*[8] approach.

Postulating the absence of elementary objects other than pixels we avoid many problems of theories discussed in Chapter 5. All objects in our system are subject to Gestalt grouping principles, although as we shall see later, different principles control the recognition process at different scales. Moreover, some grouping principles are implemented as properties of the recognition algorithm itself, whereas others are expressed through rules of composition in the knowledge base (e.g. the 'good continuation' principle, section 9.6). This division roughly corresponds to the discussed in section 5.1 division of the Gestalt grouping principles into innate and learned. Thus, we don't need to answer the question where the elements of Gestalt grouping principles come from.

It is necessary to stress that we do not specify the exact hierarchy of parts as a component of our theory. The actual structure of parts and their relations is a subject of empirical research. Moreover, it is very likely that some theories of object perception and object shape representation reviewed in Chapter 5 correctly describe perceptual processes at their specific scales. For instance, a RBC-like geon-based representation can be used by the human visual system for coarse 3-D models of objects, whereas for human faces more fine-grain representations are used. All these representations can co-exist within the recognition knowledge base of our system. Thus, our system can inherit strengths of the above theories while simultaneously solving their problems.

6.1.1 Pixels and Templates

Pixels are the simplest, indivisible objects in our theory. They roughly correspond to elementary responses of retina cells to external stimula. An individual pixel doesn't have any shape but, as any other object in our theory, can have a set of attributes associated with it. The most important pixel attributes are pixel's coordinates.

This 2-D coordinate system is effectively a concise representation of all the *spatial* relations between pixels. For instance, if we know coordinates of two pixels

[8] Mereology (*meros*, Greek "part") is the logic of a whole conceived as though physically constituted by its parts. It was created by the Polish logician Stanislaw Lesniewski ([56]).

we can calculate distance between them, whereas for three pixels we can calculate the angle between the lines they define, etc.

Other pixel attributes contain the information that is available as the result of the interaction of the system's sensory elements with the environment. In the case of black-and-white vision there is one numeric attribute – pixel's luminance. For colour vision there are three attributes corresponding to the light intensities in three independent colour channels.

As we noted in section 5.3.1, in order to extract initial information from the image consisting of pixels we have to compare the image with templates. Local template matching is a special type of categorisation in our system. For instance, the Marr-Hildreth edge detector can be implemented as a template, which categorises pixels to those belonging to edges and the rest. In section 9.3 we describe a set of four simple templates that we use to detect edge pixels in a binary image and to assign them to one of four possible categories depending on the edge direction.

Categorisation performed by template matching can be explicit or implicit. The two examples above demonstrated explicit categorisation when a new category was assigned to a pixel. Instead, template matching can create new pixel attributes. For instance, in the case of the luminance gradient calculation, each pixel is augmented with new attributes corresponding to the gradient direction and absolute value. This is an example of the implicit pixel categorisation.

6.1.2 Categorisation and Abstraction

As we mentioned in section 5.4.1, the classical way to define categories is to specify a list of constraints an object has to conform to in order to belong to the category. The other method of category specification includes the specification of a prototype object and a similarity measure. Besides, it was noted that unlike categories in classical logic, perceptual categories are fuzzy, i.e. each member belongs to them to a certain degree, which ranging from 0 (not a category member) to 1 (full category member).

In our system we define categories using the classical way of specifying a list of constraints. This might seem to contradict to experimental results cited in section 5.4.1, but in reality it doesn't. The constraints that we use are *fuzzy constraints*, and calculate not Boolean true/false values, but rather a degree of the category membership. Classical binary (or crisp) constraints constitute a special case of fuzzy constraints and also are used by the system.

What about prototypes and similarity measures? It turns out, that though not formally defined, they are evolving properties of our categorisation algorithm. Indeed, fuzzy constraints of a category will calculate maximum category membership value for a certain (not necessarily unique!) set of object attributes. Any object with this set of attributes can be considered as the category prototype; then category constraints can be considered as the measure of similarity between an object and the category prototype.

In section 5.4.1 we pointed out that categories (both perceptual and conceptual) are organised into the mathematical structure called 'lattice' with respect to the *abstraction* relation. More specific categories are subcategories of more abstract ones. This structure of perceptual categories is extensively used in the control mechanism of the categorisation process (section 6.2) as well as for the knowledge base structuring and refinement (sections 6.3 and 6.4). Therefore, proper understanding of how the *abstraction* relation of categories can be specified becomes very essential.

In mathematical logics a class (the concept corresponding to the 'category' concept in the context of psychology) can be specified either extensionally or intensionally. The first specification explicitly lists all elements constituting the class. The second provides a set of necessary and sufficient conditions, which a class element has to satisfy. By analogy with this definition we call two mechanisms of an abstract category specification *extensional* and *intensional* abstraction.

In the case of extensional abstraction an abstract category is defined as an explicit list of (more specific) subcategories. Indeed, the resulting category is an abstraction of any subcategory as it includes any element that the subcategory includes, but the opposite is not true. An example of this type of abstraction can be found in linguistics, where, for instance, the category of all verbs is a combination of the category of transitive verbs and the category of non-transitive verbs.

When an abstract category is specified through a set of constraints which are more permissive than constraints defining its specific subcategory, we call it *intensional* abstraction. Intensional abstraction can be achieved either by dropping some of constraints defining the subcategory, or by 'relaxing' them. An example of this type of abstraction is the category of ellipses, which is an abstraction of the category of circles (naturally, if we consider a circle as a special ellipse, and not as a different kind of a geometrical object). Note that neither of the two categories can realistically be specified extensionally.

Finally, if we recall that any object in our system (apart from pixels) is made of parts, we arrive at the third type of abstraction – *structural* abstraction. If categories A and B have the same set of constraints, but objects of the category A consist of more abstract parts than objects of the category B then the category A is structural abstraction of the category B. An example of structural abstraction can be found, for instance, in linguistics, where there exist general verbal phrases and more specific transitive verbal phrases. Elements of these two categories consists of the same parts, except that in the first case any verb can be a part of a phrase, whereas in the second – only a transitive verb.

Two subcategories of the same abstract category are *complementary*, if they don't have common elements, or, alternatively, if they don't have common subcategories. The fact of complementarity of two categories is used to optimise the categorisation process: if an object belongs to one of those categories, the membership test for the second category can be omitted.

6.1.3 Parts and Wholes

In section 5.2 we reviewed evidence for the psychological importance of perception of parts and wholes. The part/whole relation structures not only visual perception, but also hearing (see section 3.1), perception of events, etc. Therefore, the main conjecture can be also rephrased as a claim that human perception of different modalities evolves around and is governed by the two universal relations: abstraction and part/whole. The role of the former we have discussed in the previous section. In this section we shall focus on the latter.

The universal nature of the part/whole relation has profound implications for the representation of categories in our system. The fact that our categories are *categories of objects consisting of parts* induces the following taxonomy of constraints defining categories:

- structural or relational constraints;
- part-type constraints;
- part-attribute constraints.

Structural constraints, as it follows from their name, define the way the parts of an object have to relate to each other in order to form the object. Part-type constraints specify categories of object parts. Part-attribute constraints further restrict the set of objects, which can serve as parts of a member object of the category.

Let us consider as an example the category of squares with sides parallel to coordinate axes. According to the above taxonomy, mutual position constraints (i.e. adjacent sides with right angles between them) and length constraints (i.e. equal length of all sides) are structural constraints. The definition of sides as straight-line segments is a part-type constraint. And the fact that at least one of the sides has to be parallel to a coordinate axis is a part-attribute constraint.

Unlike the above domain-independent classification of constraints, the nature of structural and part-attribute constraints changes from domain to domain. In the case of visual perception we deal with spatial constraints, i.e. relations between points, lines, regions in the 2-D or 3-D Euclidean space. In Chapter 9 and Chapter 10 we will meet multiple examples of spatial constraints.

The taxonomy of constraints we provided in this section is important for the understanding of the representation of intensional abstraction in our system as well as for the understanding of the principles of the recognition knowledge base refinement (section 6.4).

6.1.4 Object Attributes and Constraints

An individual object in our system is defined by the category it belongs to, by the parts it consists of, and by its attributes.

What is the role of attributes in the representation of information necessary for recognition? The most straightforward answer to this question is that attributes are necessary for:

- calculation of constraints;
- calculation of a parent object's attributes;

- storage of auxiliary information, e.g. in the case of handwriting recognition the average slant of handwriting or other stylistic properties.

Another, a less obvious role of attributes is that they provide a means of implicit sub-categorisation. This becomes more apparent if we consider a category of objects with a nominal attribute. Such a category is a union of sub-categories, each containing only the objects with one fixed value of the attribute. Note that in this representation the attribute becomes redundant because its value for a given object is defined by the sub-category the object belongs to.

In the case of numerical attributes the above explicit splitting of a category to sub-categories becomes impossible (or, in the very least, impractical), as we would need to represent a countable (or at least very large) number of distinct categories. Nevertheless, we define by analogy with the previous case the implicit sub-categorisation induced by numeric attributes. Reversing the above analysis we can consider an attributed category as a union of 'proper' categories of purely symbolic, attribute-free objects, which we combine into the attributed category for practical reasons.

This role of attributes in a category definition is important for understanding of learning processes discussed in section 6.4.

6.2 Features of Recognition Algorithm

Starting with Marr's influential book [58], visual processing systems are designed around the concept of modular structure. A system consists of a number of modules, each fulfilling its own function and passing information to other modules.

Our recognition framework is a radical departure from this scheme. It consists of just one module, which is functionally equivalent to multiple modules of traditional systems. This is achieved by implementing various functional modules as virtual processes within a single computational process.

The order of processing of various categorisation and composition operations in our system is controlled by a set of static and dynamic priorities. This feature provides the system with the ability to run various virtual recognition sub-processes at different speeds.

One of the consequences of the main conjecture is that visually perceived objects are *built* of parts, but are never *parsed* into parts. Thus, our approach avoids the object parsing problem (section 5.2) altogether. At the same time the difference in the speed of various virtual recognition sub-processes can explain subjective perceptual phenomena in this area.

For instance, a simple region segmentation process based on uniform connectedness (section 5.1) can be very fast because it accesses only pixel attributes and combines neighbouring pixels with equivalent attributes of some modality. The process of determining fine-grain shapes of those regions can be much slower as it requires significantly more complex processing, as we shall see in Chapter 9. The result of this speed difference reveals itself as if the system first detected the regions and then parsed them into constituting parts. The difference of processing speeds can explain also why some parts are perceived as 'good' and some as 'bad' ([71]).

As any structural description based recognition system, our recognition framework is susceptible to the problem of computational intractability. One component of the solution of this problem in our system is the representation of multiple object models in the recognition knowledge base as a lattice of parts and wholes at different levels of abstraction. This representation enables parallel simultaneous object model matching. Another means of tackling the problem is a set of domain-independent heuristics presented in section 8.5.

A lattice of category abstractions induces special ordering of the categorisation process. First, the constraints of the most abstract category are tested. If this test fails, testing of more specific categories can be omitted. Therefore, the system performs category testing from the most abstract categories to the most specific and creates only the most specific instances, which satisfy corresponding category constraints. A detailed description of this process can be found in section 8.3.4.

6.2.1 Multiple Hypothesis Testing

As we noted in Chapter 2, pattern recognition is usually a severely underconstrained process. At each step the system is presented with many ways to interpret the current state of the recognition process and choose further actions. For instance, a system fitting lines into sets of points can fit either a straight line, or a curve, or even a corner into a particular set of points.

Various systems have various strategies of the ambiguity resolution, some of them having been discussed in Chapter 3. So far, all the attempts to devise a single universal strategy of ambiguity resolution failed. Practical experience shows that if a system is designed to perform recognition in a real-life domain it has to be capable of tracking simultaneously a number of conflicting interpretations of perceptual data and postpone final decisions until later stages of the recognition process. Since the blackboard system architecture presented in section 3.1 was specially designed to have this capability, we have chosen it as the basis for our system. We shall discuss details of the implementation in Chapter 8.

6.2.2 Context Sensitivity

Context sensitivity can be seen as the ability of a system to resolve ambiguity in the interpretation of a particular object by using its context.

In the case of cursive handwriting recognition this feature of human perception acquires a paramount importance. As we demonstrated in Chapter 4 (see Figure 6) the interpretation of many handwritten letters and letter combinations depends on the context they appear in. The reliance on this ability of human perception reaches its peak in the contemporary shorthand writing ([34]), where the interpretation of every element depends on its relative position, relative size, etc.

Our system permits the specification and the use of context dependencies for categories that are specialisation of other, more abstract categories. A context dependency is a reference to a property of a whole, which contains the context-referring object as a part (not necessarily immediate). Consider, for instance, the interpretation of the letter 'e' in a handwritten word. In some handwriting styles it can be distinguished from the letter 'l' only by its size relative to the x-height attribute of the word it is a part of.

The resolution of this type of context dependency is implemented as a top-down process. The details of the implementation will be discussed in section 8.3.8.

Another type of context sensitivity, which we encountered in section 5.5.3, is responsible for the acceleration of object recognition in familiar, frequently arising contexts. In our system this is achieved by the use of dynamic priorities of individual recognition sub-tasks modified by a top-down process (section 8.5.1).

6.3 Knowledge Representation in Recognition Systems

Since the purpose of a pattern recognition system is categorisation of its input data, the main type of information stored in the knowledge base of the system is a set of category definitions. Each definition should include a set of constraints, definitions of attributes of category objects, and the category label (e.g. a letter code for a category representing a letter). Another type of important information, as it follows from the discussion in section 6.1.2, is the abstraction relation between categories. A special kind of category is a set of pixel categories induced by pixel-level templates. And finally, the knowledge base can store processing hints, e.g. priorities of different routes in the part/whole hierarchy.

A detailed description of the representation of all this information in our system can be found in Chapter 7. Here we just note that in order to increase the system flexibility we tried to increase the proportion of learnable, declarative information. Although the system as it stands now is not capable of learning, the analysis of the information necessary for the system successful functioning undertaken in this thesis is the important first step in this direction.

6.3.1 Recursive Definitions

The knowledge representation in our system allows for recursive definitions of object categories, that is, a category can refer to itself in a part-type constraint.

By its definition (see [56]) the part/whole relation is a partial-order relation. This means that the relation is:

- reflexive, i.e. an object is its own (improper) part;
- anti-symmetric, i.e. if the object A is a part of the object B and the object B is a part of A then A = B;
- transitive, i.e. if the object A is a part of the object B and the object B is a part of the object C then A is a part of C.

Anti-symmetricity of the part/whole relation ensures that the object cannot be a proper (i.e. different from the whole) part of its own part. Therefore, recursive definitions serve to describe those situations where an object can contain a proper part belonging to the same category as the object itself. Consider, for instance, a straight-line segment, which is a composition of two adjacent straight-line segments with the same direction. Similarly, a convex line can be obtained as a concatenation of two convex lines if their connection is also convex.

Recursive category definitions sometimes reflect invariant properties of objects. In the above example of a straight line this is shift invariance in the direction of the line. In more complex situations it is not trivial to specify the transformation that leaves the object unchanged.

As we shall see in Chapter 9, a recursive definition is a powerful tool allowing for the concise specification of many geometrical categories. The downside of recursive definitions is that if special care is not taken, they can easily slow down the recognition process so that it becomes impractically long (see details in section 8.5.2).

6.3.2 Representation Abstraction Range

One of the major problems of traditional pattern recognition systems is the change of data representation between different modules comprising the system. Those modules usually work at different levels of abstraction and therefore cannot share data representations. We stressed this problem when reviewed blackboard architectures in section 3.1. The problem is usually solved by introducing module interfaces, which convert data from one format to another.

In contrast to this, our system has just one representation, and this representation is abstraction level independent. Each object (except pixels) is represented as a combination of its parts and has a set of attributes, which are functions of attribute values of objects' parts. The concrete meaning of attributes depends, naturally, on the specific type of object. In Chapter 9 we shall see that the attributes of the most abstract type of line, which has just one topological property – connectedness, are the positions of the line's ends and its minimum bounding rectangle. More specific line types have other attributes defining their properties, e.g. average curvature of a smooth curve.

Therefore, our approach allows change of the degree of abstraction without changing actual representation.

6.4 Learning as the Process of Recognition Knowledge Base Refinement

Simon ([97]) defined learning as changes to the content and organisation of a system's knowledge enabling it to improve its performance on a particular task or set of tasks. The performance of a recognition system is determined by its ability to make correct recognition of the input, by the processing time, and by the generalisation ability of the system. The latter is defined as the ability of the system to recognise (categorise) correctly the input it has never encountered before. These three criteria are obviously conflicting as, for instance, the best in terms of the correct recognition rate is the system, which just stores associations between input instances and corresponding categories, but such a system would have a very poor generalisation capability as well as it would be too slow for sufficiently variable input. Therefore, the process of optimisation of a recognition system has to find a suitable compromise between the above three criteria.

At a very abstract level all recognition errors can be divided into two categories: errors caused by over-generalisation and errors caused by under-generalisation. In the first case, the knowledge base definition of a category is more permissive than it should be, and accepts objects, which in reality don't belong to this category. In the second case, the definition is too restrictive and forces the system to reject correct categorisations. A mis-categorisation error is a combination of the above two errors. Therefore, in order to improve the recognition performance of a recognition system it is necessary to modify category definitions when the system makes an error.

Unlike in many other systems capable of learning (e.g. neural networks, case-based reasoning systems, production rule systems), this modification of category definitions can be done in our recognition framework in an explicit and controlled manner.

Let us consider, for instance, a case of a too restrictive category definition. There are several ways to amend the definition:

- a constraint (relational or part-attribute) can be dropped from the category definition;
- a constraint (relational or part-attribute) can be relaxed, e.g. by specifying less steep fuzzy membership functions or by increasing the range of acceptable values;

- a more abstract part type can be specified;
- a new category definition can be created, which would accept the mis-categorised instance, and a new extensional abstraction of the two categories is added to the knowledge base.

Any of the above actions can be performed more than one time and for multiple constraints. The goal of the procedure is to find the minimum modification of the knowledge base allowing correct recognition of the mis-categorised object.

Over-generalisation can be cured by the following actions:

- a new constraint (relational or part-attribute) can be added to the category definition;
- an existing constraint (relational or part-attribute) can be toughened, e.g. by specifying more steep fuzzy membership functions or by reducing the range of acceptable values;
- a less abstract part type can be specified;
- the category can be split into a set of more specific categories.

The process of the knowledge base refinement, as it stands now, is not sufficiently formalised in order to implement it as an automatic learning procedure. Nevertheless, our practical experience (see section 10.2) shows that it is possible to apply it in real-life situations to correct recognition errors.

In this chapter we discussed general design principles of our recognition framework. Some of these principles are direct consequences of what we learned about human visual perception in Chapter 5. Others (e.g. the main conjecture) are plausible hypotheses about the organisation of perceptual processes. In the following chapters we shall describe how these general principles are implemented in our recognition system.

We would like to emphasise at this point that the design principles we have chosen as the basis for our system are not specific for a certain narrow area of pattern recognition, e.g. 2-D shape analysis. We intentionally tried to keep the system as generic as possible, so that it could be applied to solve various pattern recognition problems, provided the information necessary for recognition can be represented in the form described in the next chapter.

Chapter 7

Recognition Knowledge Base

In this chapter we describe the organisation of knowledge in our recognition system. The data structure used to store the system knowledge is a graph. We provide a detailed description of various types of nodes, links, constraints, and attributes constituting the knowledge base. We also discuss the external representation of the knowledge base and review the language used for its specification.

7.1 General Organisation of Recognition Knowledge Base

The Recognition Knowledge Base (RKB) is represented in our system as a graph. Nodes of this graph represent classes of pattern, whereas links between nodes represent part-whole, class-subclass, and abstract-specific relations. Spatial constraints of parts constituting a 'whole' are stored in the corresponding node. Nodes also contain definitions of attributes.

The RKB stores constraints and attributes in the form of expressions built of constants, references to attributes of parts, and invocations of built-in functions.

7.2 Node Types

There are four different types of nodes in the recognition knowledge base: the unique 'Start' node, 'Template' nodes, 'Whole' nodes, and 'Class' nodes. Every node in the RKB represents a class of patterns and, in the case of visual pattern recognition, every pattern is a set of image pixels.

A node of type 'Whole' describes a class of patterns constructed as combinations of other patterns – their parts. A node of type 'Class' describes a class of patterns consisting of subclasses, which are described by other nodes. The class of patterns represented by a 'Template' node contains those sets of pixels, which match the template. Finally, the 'Start' node corresponds to individual pixels.

Every node in the RKB has a unique name and sets of incoming and outgoing links associated with the node.

7.2.1 'Start' Node

The system automatically creates the unique 'Start' node. This node has links of the type 'part-of' to all template nodes in the knowledge base. During the initialisation of the system (see section 8.3.1 for details) in the 'Pixel' mode, for each foreground pixel an item of the type 'Start' is created in the working memory and a 'SEARCH'-type job is inserted into the agenda. This allows matching all foreground pixels with all templates.

7.2.2 'Template' Nodes

The most basic, low-level type of a node in our recognition knowledge base is a template. Template nodes are necessary in those cases when pixel attributes don't contain the information necessary for constraint testing but this information is encoded in the pixel neighbourhood relation.

Consider, for instance, a black-and-white image. Pixels of such an image have only one Boolean attribute - IS_FOREGROUND (another possible attribute - IS_BACKGROUND - is a simple negation of the former one). This attribute is not sufficient to distinguish internal pixels (i.e. foreground pixels surrounded by only foreground pixels) from edge pixels (i.e. foreground pixels having both background and foreground pixels as their neighbours). The standard part-whole relation coding scheme cannot be used in this case, as background pixels are not parts of an edge, but their presence in the immediate neighbourhood of a foreground pixel make that pixel an edge pixel.

Template nodes are specified as rectangles with left, top, right, and bottom coordinates relative to the pixel with the neighbourhood of which the template is matched, followed by two arrays with the size of the template rectangle. The first array contains values for the attributes of corresponding pixels (e.g. in the black-and-white case - foreground and background colours). The second array is a mask. TRUE value of an element of this array means that the corresponding pixel needs to be matched against the value in the first array, FALSE - that it should be ignored. This scheme allows coding templates of any shape. For the image border pixels we use an opportunistic strategy: all the pixels outside of the actual image are assumed to match the template.

When a template 'is applied' to a pixel, the template matching procedure calculates the number of image pixels that match corresponding template pixels. Then the ratio of this number and the total number of active pixels in the template (i.e. pixels with the TRUE value in the mask array of the template) is used as the degree of matching.

Specific templates used in our implementation are presented in section 9.3

7.2.3 'Whole' Nodes

'Whole' nodes are the main means of structural description in the recognition knowledge base. The following information is stored in each individual 'Whole' node:

- a set of parents, i.e. nodes of type 'Whole' of which this node is a part and nodes of type 'Class' of which this node is a subclass;
- a set of constraints stored as a list of expressions (see section 7.8);
- a set of parts;
- a set of attribute definitions (see section 7.5);
- a mapping between global names of attributes and their slots in this node;
- a set of nodes which are intensional abstractions of this node;
- a set of nodes which are intensional specialisations of this node;
- result nodes contain the result value (e.g. letter nodes in the case of handwriting recognition contain the code of the letter).

'Whole' nodes correspond to production rules in production systems (section 3.3) as they used by the recognition algorithm to construct new working memory items corresponding to the 'whole' RKB node from their parts.

7.2.4 'Class' Nodes

'Class' nodes serve to represent extensional abstraction in the system (see a discussion of different types of abstraction in section 6.1.2).

The structure of a 'Class' node is similar to the structure of a 'Whole' node represented in the previous section. The only difference is the interpretation of the set of parts – in the case of 'Class' nodes their 'parts' are their subclasses. A 'Class' node cannot have a set of constraints associated with it but can have a set of attributes. Those attributes are inherited by all the children of the 'Class' node.

7.3 Link Types

All nodes in the system are connected with various types of links. These links represent the following relations between nodes:

- The 'part-of' relation – links of this type connect 'Template', 'Whole', and 'Class' nodes with the 'Whole' node parts of which they are; the link contains the position of a part in the whole (the part slot number).
- The 'subclass-of' relation – links of this type connect 'Template', 'Whole', and 'Class' nodes with the 'Class' node subclasses of which they are.
- The 'abstraction-of' relation – links of this type connect 'Whole' nodes with 'Whole' nodes and represent the intensional abstraction relation (section 6.1.2).

A link of each type has a corresponding link connecting nodes in the opposite direction. Each link also has a priority value associated with it and is used by the system control mechanism as a multiplicative factor to calculate job priorities (see section 8.2). At the moment the priorities are chosen manually whereas in the future they may be adjusted by the system automatically.

7.4 Recursion

As we already discussed in section 6.1.3, recursive definitions are necessary to describe situations where a whole is similar to its part at a certain level of abstraction. We shall see in Chapter 8 that our system makes extensive use of recursive definitions. Those recursive definitions are not explicitly specified in the knowledge base, but rather are automatically detected when the knowledge base is initialised. Recursive definitions are given special treatment by the system control mechanism (section 8.5.2).

7.5 Attributes and Constraints

In order to constitute a whole, a set of parts has to conform to a set of constraints specific for this whole. The set of constraints is stored as a list of expressions assumed to be connected with the logical conjunction operator. Attribute definitions in the knowledge base are represented simply as expressions used to calculate their values for specific items.

Expressions in our system are evaluated by the system expression interpreter. Use of interpreted expressions leads to a certain degradations of the system performance but simplifies experimentation with various RKBs as it is not necessary to recompile the system to test a modification of the RKB. Our experiments showed that the overhead of interpreted expressions is minimal and no significant performance gain can be achieved by compiling them.

Attribute and constraint expressions can contain constants, references to attributes of parts, and function calls connected by operations. Currently implemented operations and built-in functions are shown in Table 1. The value of an expression can have one of the following types:

- number: numbers are represented in the double precision floating format;
- vector: a pair of integer numbers representing co-ordinates of a pixel in the image or vector defined by two pixels;
- range: a pair of numbers representing an interval of real numbers;
- item: a reference to a working memory item;
- undefined.

Operations and functions produce new values with potentially different type. For instance, the 'dist' function takes two arguments of type 'Vector' and returns a result of type 'Number'; the range creation operation ('[]') convert two numbers into a result of type 'Range'.

Table 1 Operations and built-in functions

Name	Meaning
‘+’, ‘-‘, ‘*’, ‘/’	Standard addition, subtraction, multiplication, and division
‘+^’, ‘-^’	Addition and subtraction of angles: the result is normalised into the]-PI, PI] interval
Unary ‘-‘	Negation
‘|’	Union: applicable to ranges and numbers
‘&’	Intersection of two ranges
‘[]’	Creates a range from two real numbers
‘{}’	{a} is a shortcut for the [-a, a] range
in	Tests if a number is inside a range
‘=’, ‘<>’, ‘<’, ‘>’, ‘<=’, ‘>=’	Tests if two values are equal, not equal, the first is less, greater, less or equal, and greater or equal, correspondingly
‘~’	Fuzzy predicate modifier: applicable to ‘in’, binary predicates, and predicate functions
‘:’	Combines alternatives: if the left-hand side expression is defined returns its value; otherwise calculates and returns the value of the right-hand side expression
‘!’	Fuzzy Boolean value negation (calculated as 1 – fuzzy value)
‘<<’ or ‘«’	Fuzzy “significantly less than” predicate
‘>>’ or ‘»’	Fuzzy “significantly greater than” predicate
‘-=’	The sign of two numbers is the same
‘<->’	Different signs
abs	Absolute value of a number or a vector(component-wise)
avg	Average value of two numbers or two ranges
dist	Distance between two points
connected	Are two nodes connected?
angle	Direction of the vector defined by two points
min	Minimum of two numbers or lower bound of a range
max	Maximum of two numbers or upper bound of a range
delta	Span of a range
length	Length of a vector
inside	Is given shape inside of a combination of other shapes?
close	Is given shape (or point) close to another shape (point)?
above	Is given point (shape) above of another point (shape)?
left	Is given point (shape) left of another point (shape)?

sin	Sine
cos	Cosine
tan	Tangent
leftmost	Returns the leftmost pointfrom the given set of points
rightmost	Returns the rightmost point from the given set of points
topmost	Returns the topmost point from the given set of points
lowest	Returns the lowest point from the given set of points
x	Returns the x-coordinate of a point
y	Returns the y-coordinate of a point
point	Combines two numbers into a point with corresponding coordinates

Boolean values TRUE and FALSE are represented as numbers 1 and 0 correspondingly. An expression can also calculate a fuzzy class membership value, which is represented as real number in the range [0,1]. Fuzzy constraints are calculated by fuzzy predicates corresponding to binary predicates 'in', '=', '<>', '<', '>', '<=', '>=', and predicate functions. It is possible to define the steepness of a class membership function. The use of fuzzy predicates is explained in detail in section 8.3.6.

7.6 Abstraction: Recognition Knowledge Base Representation

As we have discussed earlier, there are two types of abstraction - intensional and extensional (see section 6.1.2). Correspondingly, there are two ways to represent abstraction in the RKB – through 'abstraction-of' links and 'Class' nodes.

Two nodes are connected with an 'abstraction-of'-type link when the following conditions are satisfied:

- they have equal numbers of parts;
- part nodes of the first node coincide or are abstractions (either intensional or extensional) of corresponding part nodes of the second node;
- the set of constraints of the first node is a subset of the set of constraints of the second node.

These conditions guarantee that an instance of the second node is also an instance of the first node. A node which is a specialisation of another node inherits constraints and attributes of the latter. This leads to a more compact representation of the RKB because it is not necessary to repeat attribute and constraint definitions in the definition of the specialisation node. Besides, it leads to more efficient processing, as we shall see in section 8.3.4. At the moment the system doesn't create 'abstraction-of' links between nodes and just checks the above conditions for existing in the knowledge base links. In the future the system can be amended to infer the 'abstraction-of' relation between nodes automatically.

Any node can be a subclass of some 'Class' node (except itself, obviously) and any 'Whole' node can have a part represented by a 'Class' node.

7.7 Lattice of Specialisations

A node in the RKB can be a specialisation of multiple nodes. This case corresponds to multiple inheritance in object-oriented programming. For instance, we want to represent in the RKB the concept of convex low-curvature line. It can be defined as the low-curvature specialisation of the node representing the convex line concept or, alternatively, as the convex specialisation of the low-curvature line concept. In both cases the duplication of concept node is unavoidable if a node can be a specialisation of only one other node. In our system we allow a node to be a specialisation of multiple nodes and in the above example we create a single node, which is a specialisation of the two nodes and thus represents the concept we need.

The abstraction relation is a partial order relation (see section 6.3.1 for the definition of a partial order relation). A concept is its own (albeit a trivial) abstraction. If the concept A is an abstraction of the concept B and the concept B is an abstraction of A then A = B. Therefore, abstract and specific concepts constitute a mathematical structure known as 'lattice'. In the RKB this structure is represented by a directed acyclic graph (DAG) containing nodes and 'abstraction-of' / 'specialisation-of' links.

As it was mentioned above, specialisation nodes automatically inherit attributes and constraints from their abstractions. A node which is a specialisation of multiple nodes inherits attributes and constraints of all those nodes. If the system detects a conflict of attribute definitions, i.e. an attribute with the same name is defined for multiple abstractions, it must be resolved by specifying this attribute explicitly for the specialisation node.

Unlike in the case of attributes, inheritance of constraints doesn't mean that constraints are copied from abstract nodes to their specialisations. The effect of inheritance is achieved by choosing certain order of constraint evaluation, as it is described in section 8.3.4.

7.8 Recognition Knowledge Base Specification Language

Full specification of the RKB description language in BNF is provided in Appendix A. In this section we discuss the main features of the language and provide examples of its use.

The full definition of a domain specific RKB is stored outside of the system in a text file and is loaded during the system initialisation. The definition consists of multiple sections describing constants, shared expressions, and nodes. One or more empty lines separate sections from each other. The file can also contain C++ style comments (i.e. line endings or entire lines delimited by the '//' symbol combination). These comments improve the readability of the definition and are ignored by the system.

A definition of a shared expression starts with the symbol '$' followed by the expression name and the '=' sign. The right-hand side of the definition contains the body of the expression. This expression can be referred to in other expressions by its name preceded by the '$' sign. A definition of a constant is a special case of the shared expression definition, where the right-hand side is a constant expression. The system recognises by name the constant 'PI'.

A definition of a node can describe a 'template', a 'whole', or a 'class' node. In the first case the definition specifies the template and the mask array sizes followed by the specification of values of individual pixels (see section 7.2.2). The value 'x' means that the pixel should be ignored during the template matching and corresponding element of the mask array is set to FALSE.

A 'whole' node definition contains references to the part nodes constituting this 'whole' connected by the '+' sign. Each reference can contain a priority value in parentheses and can be followed by the '!' sign denoting exclusive use of the corresponding part. The definition can contain a constraints part denoted by the 'WHERE' clause. The clause consists of one or more constraint expressions connected by the

‘AND’ keyword, each starting on a new line. The optional ‘SPECIALISATION OF’ clause specifies a list of nodes, for which the current node is a specialisation.

A ‘class’ node definition contains a list of references to subclass nodes connected with the ‘|’ sign.

Both ‘whole’ node and ‘class’ node definitions can contain descriptions of node attributes in the corresponding ‘ATTRIBUTES’ clause. The system supports a set of predefined attributes such as ‘FLAGS’, ‘LABEL’, and ‘COLOUR’. Other attributes are node specific ones. Their definitions start at a new line and contain the attribute name followed by the ‘=’ sign and the expression used to calculate the attribute.

Expressions in the system are represented in the usual infix forms used by most contemporary programming languages. They can contain constants, attribute references, mathematical operations signs, range manipulation operation, predicates (both binary and fuzzy), function calls, and parentheses. The operation signs correspond to those in the language C, except for the equality sign ‘=’ and non-equality sign ‘<>’. There are two special operations for angles: ‘+^’ and ‘-^’. They normalise the result of usual addition and subtraction to the]-PI, PI] range. The functions currently implemented in the system are listed in Table 1.

Fuzzy constraints are specified by ternary expressions, i.e. a normal binary predicate or a predicate function followed by the ‘~’ sign and a numeric value defining the steepness of the class membership function (the ‘bell curve’).

Range operations include the union of two ranges (the ‘|’ sign), the intersection of two ranges (the ‘&’ sign), and the range operation (square brackets) which creates a range from two numeric values.

Attribute references are represented as the ‘@’ symbol followed by the part number (starting with 0), the ‘.’ sign and the attribute name. Special part reference sign ‘@’ is used to reference an item as its own part, thus allowing attribute cross-reference and helping to avoid multiple calculations of the same expression. The system recognises a few predefined attribute names related to the attributes of the minimal bounding rectangle of an item and the beginning and the end points of a contour segment. Other attributes must be defined in node definitions. The system allows forward attribute and node references.

A special type of attribute references is a context reference, which is represented by the '#' symbol followed by the context node name, the '.' sign and the attribute name. Unlike normal attribute references, a context reference refers to an attribute of a parent rather than a part. See details of their processing in section 8.3.8.

The description language of the system is case insensitive.

Table 2 Examples of node definitions

A 'whole' node definition	A 'class' node definition

EDGE = EDGES(1.1) + EDGES(0.9)
WHERE @0.end=@1.beg
AND @0.endAngle -^ @1.begAngle <> PI
ATTRIBUTES flags = TRACE
begAngle = @0.begAngle
endAngle = @1.endAngle
	EDGES = RVE(0.3) | LVE(0.3) | THE(0.3) | BHE(0.3) | RT | RB | LT | LB | EDGE
	ATTRIBUTES flags = TRACE

Table 2 provides an example of node definitions. According to these definitions the 'whole' node 'EDGE' consists of two parts, both having types 'EDGES'. The definition specifies priorities of the evaluation for each part. The constraint part of the definition contains two constraints: the first one states that the end-point of the first node should coincide with the start-point of the second part; the second constraint excludes combinations when the two parts are exact opposites of each other. The node definition contains definitions of attributes, both system ones ('flags') and node-specific ('begAngle' and 'endAngle').

The definition of a 'class' node 'EDGES' states that this class consists of the subclasses 'RVE', 'LVE', 'THE', 'BHE', 'RT', 'RB', 'LT', 'LB', and 'EDGE', It also contains the 'ATTRIBUTES' part.

It is easy to see that these two node definitions together provide an example of a recursive node specification as an EDGE-type item can be a part of another EDGE-type item.

In this chapter we provided a detailed description of the knowledge base structure used by our system. We listed all types of nodes, links, attributes, and constraints and explained their purpose in the system. We also described our expression language, which facilitates specification of attributes and constraints.

The recognition knowledge base is stored in an external file in textual format and is loaded when the system starts. The use of the textual external representation allows easy prototyping, development, and maintenance of the recognition knowledge base.

Chapter 8
Recognition Algorithm

This chapter provides a detailed description of our recognition algorithm. First, we start with a general overview of the algorithm. It can be logically split into the bottom-up process responsible for assembling wholes from parts and the top-down process, which controls the bottom-up process by assessing hypotheses probabilities and modifying the order of evaluation of individual elements of the bottom-up process. We provide a detailed description of the processes and their interactions along with some argument justifying the architecture. Finally, we contrast the architecture of our system with some existing similar architectures including blackboard systems, production systems, etc.

8.1 Recognition Algorithm Overview

A simplified diagram of the algorithm is shown in Figure 8.

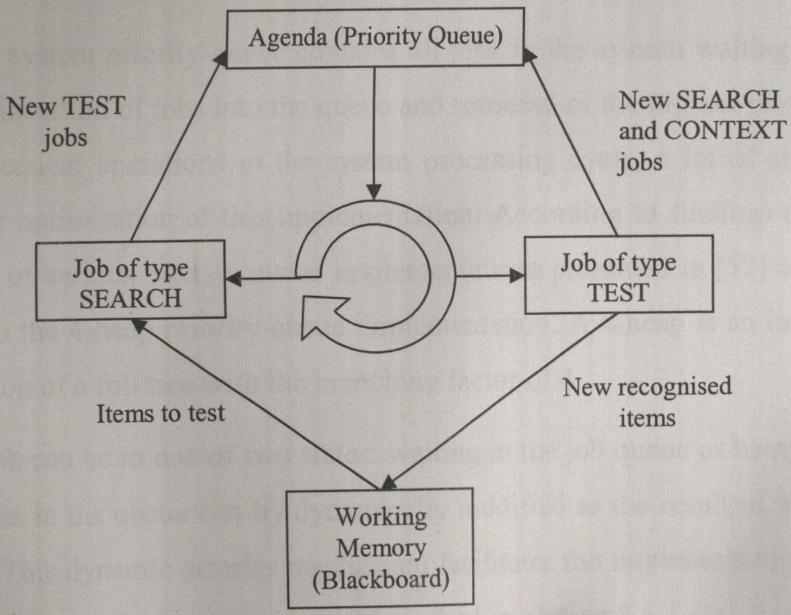


Figure 8 Recognition Algorithm

The entire recognition process is split into elementary jobs. These jobs correspond to Knowledge Source Activating Records in blackboard systems (see section 3.1 for details). There exist a few different types of jobs. The main job types are 'TEST' and 'SEARCH'. All jobs are stored in the system's central priority queue. The processing loop extracts the job with the highest priority from the queue and evaluates it. As the result of this evaluation new jobs can be added to the queue or newly recognised items can be added to the system working memory. Unlike in other blackboard systems, our working memory is not split into 'presentation levels'. Rather, each node in the Recognition Knowledge Base has a list of recognised items associated with it.

'SEARCH'-type jobs search the working memory for candidate sets of items, which can constitute a whole, and insert 'TEST'-type jobs into the queue for each such set. 'TEST'-type jobs test constraints associated with a given node in the RKB and if those constraints are satisfied by the set of items, they create a new item and insert a 'SEARCH'-type job for this item into the queue.

8.2 Elementary Jobs and Priority Queue

A job in our system is a (relatively) simple indivisible operation, which is scheduled through the system priority queue. Depending on its type, an individual job can check 'Whole' node constraints, add a new item to the working memory, or search working memory for candidate item sets, etc.

The system priority queue contains all jobs in the system waiting for evaluation. Since insertion of jobs into the queue and retrieval of the highest priority job are the most frequent operations in the system processing cycle, a lot of attention was given to the optimisation of this implementation. According to findings of comparative studies of various priority queue implementations presented in [52] and [57], we have chosen the 4-heap priority queue implementation. A 4-heap is an implicit array representation of a full tree with the branching factor of 4.

A job can be in one of two states: waiting in the job queue or being evaluated. Job priorities in the queue can by dynamically modified as the result of evaluation of other jobs. This dynamic priority mechanism facilitates the implementation of the dynamic inhibition/excitation propagation described in section 8.5.1.

8.3 Bottom-up Processing

The purpose of the bottom-up processing part of the algorithm is to construct new items from their parts according to system knowledge stored in the Recognition Knowledge Base. This is achieved in two steps. In the first stage, for each newly created item a job of type 'SEARCH' tries to find candidate items, which potentially can be used to build a new item (if they satisfy spatial constraints associated with that item type). To perform the actual constraint checking, new jobs of type 'TEST' are inserted into the job queue. When such a 'TEST' job becomes the highest priority jobs in the queue, it is extracted from the queue by the processing loop and evaluated. If all constraints are satisfied a new item is created and added to working memory. Simultaneously a new job of type 'SEARCH' for this item is inserted into the queue.

8.3.1 Working Memory and Agenda Initialisation

When the system starts recognition it finds all foreground pixels in the input image. For each foreground pixel an item corresponding to the 'Start' node (section 7.2.1) is created and a 'SEARCH'-type job is inserted into the job queue. When this process of the initial 'seeding' is finished the system starts its main processing cycle.

Alternatively, the system working memory can be initialised with nodes (and their corresponding 'SEARCH' jobs) which represent elementary (1 pixel long) edges with 4 possible orientations. In this case the first recognition step, from pixels to elementary edges, is performed implicitly, during the system initialisation phase.

8.3.2 Main processing cycle

The main processing cycle consists of the following steps:

1. retrieve the highest priority job from the job queue;
2. evaluate the job;
3. check termination condition(s);
4. if termination condition(s) evaluates to false go to step 1;
5. scan working memory for 'result' items;
6. report results of recognition.

8.3.3 'SEARCH' Jobs

A job of type 'SEARCH' is associated with exactly one item in the working memory. Each 'Whole' node in the RKB contains a list of references to the 'Whole' nodes of which this node can be a part and a list of references to possible siblings, i.e. nodes, which can be parts of the same 'Whole' node. Node references in those lists are sorted according to the priorities of corresponding links. A 'SEARCH' job traverses those lists and examines working memory items connected to RKB nodes trying to find a set of sibling items, which can potentially constitute the target 'Whole' node. For each such set the 'SEARCH' job inserts a 'TEST' job into the system job queue.

In order to decide if a particular sibling item (or set of items) deserves a test 'SEARCH' jobs use dynamically created filters. A filter is a set of range constraints on attributes of a sibling item. Those ranges are computed by the reverse traversal of the target 'Whole' node's constraints with substituting attribute values of the item associated with the 'SEARCH' job. The system effectively resolves equations for attribute values of a sibling node using an opportunistic strategy: if it cannot resolve the equation, it assumes the maximum attribute range calculated so far. Using these range filters a 'SEARCH' job rejects candidate siblings with attribute values outside of the ranges in the filter. This allows significant reducing of the number of generated 'TEST' jobs.

8.3.4 'TEST' Jobs

The purpose of 'TEST' jobs is to test candidate sets of parts and, in the case they satisfy constraints of a particular 'Whole' node, to create a new instance of this node, which is then stored as an item in the system working memory.

This pretty straightforward scenario is complicated by the presence of the 'abstraction-of' relation between nodes. In order to avoid redundant computations the recognition algorithm makes extensive use of this relation. The rationale behind this use is quite simple: a set of parts can constitute an instance of a specialisation node only if they constitute an instance of its abstraction(s). On the other hand, once the set of constraints of an abstract node is satisfied it is not necessary and moreover, it is undesirable, to recalculate it for all specialisation nodes.

Therefore, the constraint testing procedure starts testing with the most abstract node in a given lattice of specialisations (section 7.7). If the test fails (i.e. returns the confidence value below the rejection threshold, see next section for details of the confidence level computation), it just returns. Otherwise, it recursively calls itself for all specialisation nodes of the current node. If the specialisation node test succeeds, the procedure continues with the lattice downward recursion, until it encounters the most specific node, i.e. a node without specialisation. If the constraint test succeeds for that node, an item of corresponding type is added to the working memory of the system and the procedure returns the 'success' return code. Otherwise, it returns the 'failure' return code. An instance for a node that has specification nodes is created only if all invocations of the constraint testing procedure for those nodes return 'failure'. This algorithm facilitates minimal computations of constraint expressions and creation of only most specific valid instances from a given set of parts. Special care is taken in the case of a node with multiple abstractions. An instance for such a node can be created only if all constraints of all abstraction nodes are satisfied.

8.3.5 Working Memory Items

Working memory items store information about instances of 'Whole' nodes completely or partially recognised by the system in the process of recognition. Each item contains the following information:

- a reference to its corresponding 'Whole' node;
- a list of instances of parts;
- a list of references to items of which this item is a part;
- the confidence value for the item;
- the dynamic processing priority of the item;
- the minimal bounding rectangle;
- the beginning and the end points for contour segments
- a list of jobs associated with the item;
- an array of attribute values computed for the item.

We use the R-Tree structure ([36]) to index items stored in the working memory. This allows significant acceleration of item retrieval.

8.3.6 Recognition as Evidence Propagation

Every pattern recognition system has to deal with uncertainty in the decision process. There exist two main sources of this uncertainty, namely distortions of the input data and the fact that a certain pattern can be recognised as an instance of more than one class (with different degrees of confidence). Our recognition system addresses the problem of uncertainty using the evidence propagation paradigm and fuzzy constraints.

Each recognised item has a confidence level associated with it. From the point of view of probability theory this confidence level describes the probability that the item is actually an instance of its corresponding class (that is, *a posteriori* probability). The confidence level depends on the confidence levels of parts, on the degree of the fuzzy constraint satisfaction, and on the missing parts. For a whole consisting of *n* parts its confidence is calculated by the formula:

$$\texttt{Confidence} = \prod \texttt{Constraint}_i \cdot 1/n \cdot (\sum \texttt{Confidence}_j)$$

$\texttt{Constraint}_i$ here is the degree of satisfaction of the *i*-th fuzzy constraint; $\texttt{Confidence}_j$ is the confidence level of *j*-th part, which is assumed to be 0 if the part is missing.

The degree of satisfaction of a fuzzy constraint is computed by a fuzzy predicate or a fuzzy predicate function (section 7.5) using the formula:

$$\texttt{Constraint}_i = e^{(-\texttt{factor} \cdot \Delta^2)}$$

where 'factor' is a constant determining the 'steepness' of the fuzzy class membership function and 'Δ' is the distance between the region described by the corresponding binary predicate and actual value of the expression. For instance, in case of the '<' (less than) predicate this distance is equal to 0, if the first operand is less than the second, and the difference between the first and the second operands otherwise.

In order to avoid computation of items with low degree of confidence ('noise' items) the recognition algorithm uses the global rejection threshold. If the confidence value of an item is less than this threshold then this item cannot be used as a part in the construction of new items. If it has some parts missing, it is stored in the working memory and waits for those parts to become available.

8.3.7 Symmetry and Part Permutations

Some 'Whole' nodes can contain parts of the same type. For instance, the structural definition of the square in section 2.2 has four equivalent parts – straight line segments. For such nodes the recognition algorithm checks all suitable permutations of the set of potential parts. The best permutation is then the one which maximises the confidence value discussed in the previous section. This permutation is used to create the 'whole' item. Special care has to be taken when specialisation nodes are tested, as a permutation of parts, which is suitable for an abstract node, may turn out non-suitable for its specialisation.

A permutation of parts is considered a suitable one when each part item in this permutation has the type node specified at the corresponding part slot in the 'Whole' node or its specialisation.

8.3.8 Context-sensitive Processing

As we stated in section 6.2.2, context sensitivity is one of the most important desirable qualities of a recognition algorithm. In our system we provide explicit means to specify context-sensitive processing. This is achieved by the coherent use of the abstract-specific relationship between nodes in the knowledge base and the top-down processing.

A constraint associated with a 'Whole' node, which is a specialisation of another 'Whole' node, can refer to attributes of a parent node of the more abstract of the two nodes. When the system encounters this situation, it doesn't try to test the constraint (it cannot do it at this stage as the parent node doesn't exist yet), but rather posts a request for that parent node. Once the parent node instance, which is the parent of the corresponding instance of the abstract node, is created the system starts top-down processing by inserting a job of the type 'CONTEXT' to the system job queue. This job performs the same processing as a 'TEST' job for the more specific of two nodes mentioned above, except it uses the reference to the parent instance while calculating the value of the constraint referring to the parent. Therefore, this mechanism allows using context dependency in the knowledge base specification. We shall see an application of this mechanism in section 10.1.2, where certain inherent letter-specification ambiguities are resolved through references to attributes of the word comprising the letter.

Comparing this mechanism to the notion of context dependency in the theory of formal languages we notice that it can be described by a rule of the form

$$\{A, B, \ldots\} <- \{C, B, \ldots\}$$

where A and C are non-terminal symbols, B and optional other symbols are terminals or non-terminals, '{}' around rule parts designate lack of linear precedence, A is a specialisation of C in our terminology, and, finally, symbols B and the rest provide the context in the standard terminology of formal languages. Thus, the context sensitivity mechanism we introduced here is a special case of the generic context sensitivity notion but unlike the latter it facilitates computationally efficient processing.

8.4 Computational Tractability Problems

The bottom-up part of the recognition algorithm described so far performs exhaustive search. Although this straightforward approach is guaranteed to find all instances existing in the image, it is computationally very expensive in all but the most trivial cases. Even the process of extraction of straight-line segments of edges (section 9.3) can become quite a lengthy procedure as it would generate (in terms of syntactic recognition methods) all possible parsing trees, the number of which grows exponentially with the length of the segment being parsed. This fact is an unpleasant consequence of the use of powerful recursive definitions. Besides, as most recognition problems are severely under-constrained, the algorithm would generate an exponentially growing set of conflicting hypotheses without any means to make a choice between them.

Therefore, it is necessary to keep a certain balance between the ability of the algorithm to evaluate simultaneously multiple hypotheses and the computational cost of the evaluation of too many hypotheses. In our system this is achieved in two ways: first, we use domain-independent heuristics to prune the search tree; second, we introduced a top-down process, which controls the order of evaluation of conflicting hypotheses.

The overall bottom-up information processing in our system can be viewed as a kind of parsing of a 2-dimensional pictorial language. Drawing further the analogy between our recognition algorithm and bottom-up parsers used to parse natural and computer languages we can state that our algorithm employs only the immediate dominance relation and not the linear precedence relation. The processing time of the standard bottom-up chart parser in this situation grows exponentially with the size of the input, as at each scanning step it doesn't have a way to choose the next input symbol and has to consider all of them. Much of the effort in the design and the implementation of our system is devoted to devising heuristics which can solve this problem.

8.5 Domain-independent Heuristics

The domain-independent heuristics implemented in our system try to reduce the amount of computation using only generic relationships between the recognition knowledge base nodes, i.e. the part/whole, abstract/specific relationships.

8.5.1 Inhibition and Excitation: Search Tree Pruning

In order to describe the modification of the basic bottom-up algorithm that optimises the order of computation in the system we will use neural networks terminology, specifically, the concepts 'inhibition' and 'excitation'. This use is inspired mostly by the analogy between the Interactive Activation Model ([60]), the Letter Spirit project ([61]), and our system.

The following operations on the job queue and knowledge base nodes can modify the order of computation in our system:

- increase individual job priority (excitation);
- decrease individual job priority (relative inhibition);
- discard an individual job (absolute inhibition);
- increase priority of all jobs for a particular instance (instance excitation);
- decrease priority of all jobs for a particular instance (relative instance inhibition);
- discard all jobs for a particular instance (absolute instance inhibition);
- discard all jobs corresponding to a particular node (absolute node inhibition);

- allow processing jobs trying to instantiate a particular node (node excitation).

Apparently, only absolute inhibition operations reduce the number of jobs processed by the system. Other operations only modify the order of evaluation. This reordering accelerates finding the first suitable solution; besides, the reordering facilitates some extra absolute inhibition operations, thus implicitly reducing the amount of computation. In the next sections we shall see domain-independent heuristics, which perform excitation and inhibition operations.

There exist two main ways of reducing amount of computation in the system: first, to try to prevent the system from performing redundant computations and, second, to guide the system towards the most probable solution. The absolute inhibition operations fulfil the former function, whereas relative inhibition and excitation fulfil the latter.

8.5.2 Recursive Nodes Greedy Parsing

As was mentioned above, the main problem tends to be related to recursive nodes, as they cause exponential growth of (mostly redundant) computations. In order to solve the problem, we adopted a greedy parsing approach for these nodes. This approach is enforced by two restrictions of the basic algorithm:

- each instance can have only one parent instance corresponding to a recursive node;
- a parent of the same type as the current instance (therefore, this is an instance of a recursive node) inhibits all other parents.

This is a very powerful heuristic, which, on one hand, solves the problem of unaffordable computation cost for recursive definitions; on the other hand, although disallowing some valid parses, this heuristic produces sufficiently rich recognition output for the recognition process to come to successful completion. As a matter of fact, as we shall see in Chapter 10, this heuristic alone allows the system to achieve good performance on a moderate-sized recognition knowledge base.

8.5.3 Mutual Inhibition of Conflicting Hypotheses

Similar to other blackboard and neural-like systems, conflicting hypotheses in our system perform mutual inhibition. This is achieved by decreasing job priorities for all the jobs associated with a given working memory item, once a successful instantiation of a parent of this item happens in a 'TEST' job.

8.5.4 Excitation: Target Order Choice

One factor that determines the order in which the algorithm tries to instantiate parents for the given instance of a part node was mentioned in section 7.3: this is the static priority of the link connecting the nodes. The system can also modify these priorities dynamically, using top-down excitation. Once a parent node is partially instantiated, it increases priorities of all jobs that can potentially create a missing part instance for the parent instance. This process is localised, i.e. only jobs for those nodes that are situated in proximity of the (partially instantiated) parent node. Besides, it is used only for those part nodes that don't have clear target order preference specified by static link priorities.

8.5.5 Excitation: Delayed Specification

A big well-designed recognition knowledge base consists of a large number of nodes with abstract/specific relationships. Not all 'specific' nodes are met in the input with the same frequency. Some of them are necessary to make correct recognition decisions in only some specific contexts. Therefore, the system adopts a 'lazy' top-down evaluation strategy for this type of nodes. It means that the nodes of this type are initially inhibited and the system doesn't try to instantiate them in the basic bottom-up part of the algorithm. Rather, if the situation arises, when a certain 'specific' parent node requires one of its parts to be of the type of the inhibited node and the actual instance corresponds to an abstraction of the inhibited node, the system insert a top-down 'TEST' job for this node, which effectively means local excitation of the node. The implementation of this procedure is similar to context-sensitive processing described in section 8.3.8.

Another situation, where an initially inhibited node can be 'unlocked' arises when the system tries to instantiate a node with number of parts more then 2. If all but one part are instantiated (or, alternatively, when the confidence associated with this partial instantiation reaches a certain threshold), the system performs the same top-down excitation procedure.

8.6 Recognition Process Termination

The choice of termination criteria in blackboard systems is not a trivial problem. As discussed in [14], due to the under-constrained nature of most tasks it is not realistic to expect the system to try all possible routes leading to a solution and then terminate. The problem was first encountered during the implementation of the Hearsay-II system and since then no general solution has been found.

There are four termination criteria for the recognition of an individual input image in our system. When the system job queue (agenda) becomes empty the system stops and reports results of recognition. This criterion is the simplest and most obvious one. It is used also by other blackboard architecture systems where the process of recognition is expected to go through all search paths in reasonable time, e.g. in ADIK ([78]). Unfortunately, in handwriting recognition as well as in real world scene analysis the number of hypothesis is so big that this criterion has to be supplemented by additional ones.

The system can stop processing when the number of jobs evaluated or the time limit for recognition is exceeded. The time limit and total number of jobs are two global parameters, which can be configured in the system prior to the start of the recognition process.

The most sophisticated of all termination criteria is the degree of compression. The threshold value for the degree of compression is another globally configurable parameter. The value for the current degree of compression is computed as:

$$\texttt{Compression = Nresults * ResPixels / TotPixels}^{2}$$

Here `Nresults` is the number of ‘result’ items, `ResPixels` – number of pixels in them, `TotPixels` – total number of foreground pixels in the image.

When the system terminates the recognition process it examines its working memory trying to find items corresponding to the nodes of the recognition knowledge base marked as 'results'. These items (except the ones that have 'result' type parents) together constitute the result of the recognition process.

8.7 Comparison to Similar Architectures

Our knowledge representation is similar to that of ERNEST (section 3.2) or other semantic network systems. The main difference is that the RKB is a static structure, which is not modified during the process of recognition. Unlike in ERNEST, in our system concept nodes and instance items are strictly separated and reside in the recognition knowledge base and working memory respectively. This separation also corresponds to the separation between declarative knowledge and working memory in production systems ([3], see also section 3.3), or the separation between T-Box and A-Box in the KL-ONE terminology ([10]).

The recognition algorithm used in our system belongs to the class of blackboard algorithms. Alternatively, the architecture of the system can be described as the production system architecture (see section 3.1 for a discussion of differences between the two).

The evidence propagation mechanism used in our system to treat inherent uncertainties of recognition is similar to those used in parsing of context-free stochastic grammars ([30]) and Bayesian inference methods in computer vision ([6]).

A very similar approach to recognition as parsing of fuzzy shape grammars was presented by Parizeau et al. in [75], [76], [77]. As it is the case with most handwriting recognition systems, their system is created specially to perform online handwriting recognition and cannot be used for anything else. Incidentally, the syntax of the allograph model description language used by Parizeau et al. is very similar to ours.

One of the features of our system distinguishing it from the other systems is the extensive use of recursive definitions in the knowledge base and corresponding mechanisms in the recognition algorithms, such as inhibition of certain processing paths leading to exponential growth of computations. Recursive definitions are usually forbidden in recognition systems and only in ADIK (section 3.4) they are given some peripheral attention.

Another important feature of our system is the constraint computation algorithm for a lattice of specialisation nodes, which avoids multiple re-calculation of the same expression. Although similar knowledge representation schemes can be found in other recognition systems, such as ERNEST (section 3.2) or ADIK (section 3.4), none of them try to optimise computations of structural constraints. It is necessary to point out, that the very definition of the abstraction relation in our systems differs from corresponding definitions in ERNEST and ADIK. In ERNEST, a node is a specialisation of another node if it imposes tighter restrictions on the attribute ranges, i.e. it is completely attribute-based. In ADIK, a specialisation node can be obtained from another node by adding a part to the latter (e.g. the resistor and variable resistor example, [78, p.109]). In our system a specialisation node can constrain attribute ranges as well as part types of its abstract counterpart.

While designing the system we tried to combine attractive features of the systems reviewed in Chapter 3: clear presentation and maintainability of knowledge in semantic networks and simultaneous processing of multiple concurrent hypotheses in blackboard and production systems.

Chapter 9
Low-level Processing: from Pixels to Contours

Now that we have described the knowledge representation and the recognition algorithm, we shall turn our attention towards the actual job of the low-level pattern recognition. In this chapter we describe how edge pixels are detected in a binary image, how they are combined into straight edge segments, and how the latter combined into contour descriptions.

A major obstacle for the reliable recognition is noise present in all real-life images. A discussion of various sources of noise and various ways it can distort an image is concluded with a description of how the system counteracts those distortions.

The application of the 'good continuation' principle in section 9.6 allows connecting occluded edge segments and finding line intersections. Finally, we shall discuss how other Gestalt grouping principles are implemented in the system and how they affect recognition.

9.1 Recognition Process as Simultaneous Segmentation, Primitive Extraction, and Model Matching

As we pointed out in section 6.2, the main feature of our recognition framework is the absence of separate functional modules. This feature contrasts our system with all the 'standard' systems described in Chapter 4. All the operations performed by those systems in sequential modules, such as pre-processing, segmentation, feature extraction, and classification, are performed in our framework implicitly, as the higher-level result of the evaluation of elementary jobs.

Segmentation of the input is produced as an implicit result of constructing wholes from parts. Each item in the working memory of the system corresponds to a region in the input image. On the other hand, each item is created as an instance of a node in the RKB. Thus, an item assigns a label of its corresponding node to the region of the image. The same process can be considered as a feature extraction process or a model matching process – depending on the level of objects in the part/whole hierarchy.

Therefore, although our recognition system contains just one computational process, which is indivisible into separate modules, it effectively computes the same information as pattern recognition systems with traditional modular architecture, which we considered in Chapter 2 and Chapter 4.

9.2 Regions and Contours

All shape description methods can be divided into two big groups: region-based methods and contour-based methods. The former use the shape internal points, the latter – only the shape boundary points. Recent studies [92] demonstrated, that notwithstanding the common belief, that human object recognition is mainly based on the information contained in edges, contour information alone is not sufficient and has to be combined with other sources of information, such as regions, surfaces, etc.

Nevertheless, in order to simplify the implementation of the system we chose a purely contour-based representation. In the future, especially for grey-level images, this representation should be extended to include region information. This can be done by defining a region as an object consisting of a contour and a blob, where the contour is adjacent to the blob.

The following geometrical and topological properties of contours and contour segments are used in the recognition process:

- connectedness, closedness;
- convexity/concavity;
- relative and absolute orientation;
- relative qualitative distance and size;
- relative and absolute qualitative curvature.

9.3 Edges from Pixels

In order to determine which pixels belong to edges, the system uses the set of templates shown in Figure 9, with black colour corresponding to foreground pixels and white – to foreground. Template matching detects edge pixels and categorise them according to the direction of the edge they belong to. The same pixel matching more than one template can belong to different edges.

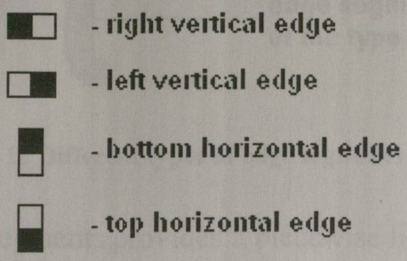


Figure 9 Edge pixel templates

Edge pixels detected by the template matching process are combined into straight edge segments. The following fragment of the knowledge base describes the process for right-hand side vertical edges:

```
RVE1 = RVE + RVE
       WHERE @0.end = @1.beg

RVE = RVE1 | RV
```

The next step in the edge construction is based on the observation of the edge shape representation in binary images.

Figure 10 below shows an enlarged image of the hand-written letter 'M'. It is easy to notice that the edge of the letter is made of step-wise edge segments. The figure also demonstrates 3 different types of edge segments – 'step', 'bump', and 'hollow'. The system recognises these three types of segments and uses them in order to compute approximate edge characteristics, such as direction, curvature, etc. Edge segments of type 'bump' and 'hollow' are considered by the system as strictly vertical or horizontal, whereas segments of type 'step' are assumed to approximate inclined lines. The direction of a 'step' edge segment is assumed to coincide with the direction of its diagonal.

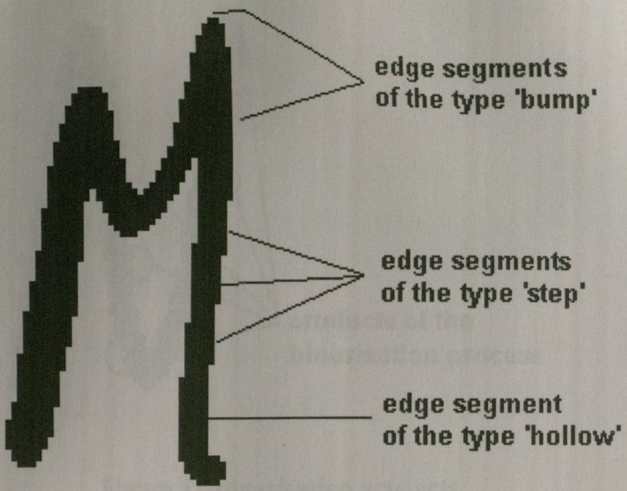


Figure 10 Different types of edge segments

Thus, a combination of edge segments provides a piecewise linear approximation of a (curved) edge. The result of edge segment fitting into an image of the uppercase letter 'N' is shown in Figure 11.

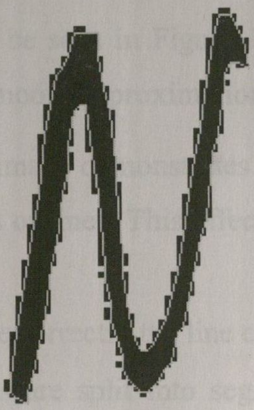


Figure 11 The result of edge segments fitting

9.4 Noise

The image of the uppercase letter 'M' in Figure 10 is an 'ideal' image because all artefacts of the binarisation process were removed from that image by hand to make the structure of edges more easily recognisable.

The image of the uppercase letter 'N' containing typical binarisation noise is shown in Figure 12.

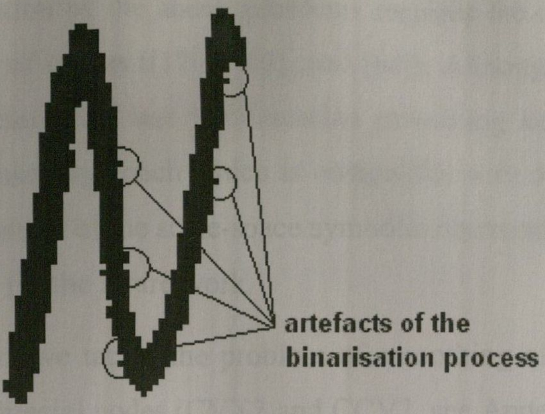


Figure 12 Binarisation artefacts

As Figure 12 demonstrates, the most common manifestation of noise in binary images is an edge segment of the type ‘bump’ or ‘hollow’ with the length of 1 pixel. Longer segments of those types can arise due to noise, but can be proper edge segments, so they cannot be simply ignored. It turns out that the simple straight edge segment fitting process described in the previous section automatically performs smoothing of this type of noise. The evidence can be seen in Figure 11, where connected straight edge segments produce sufficiently smooth approximation of the letter’s contour.

On the other hand, the same image demonstrates that binarisation noise leads to small fluctuations in the directions of lines. This effect has a few undesirable consequences:

- it is impossible to calculate correctly the line curvature;
- concave or convex curves are split into segments interlaced with small segments of opposite concavity;
- smooth curved or straight lines are split into small segments represented by arcs or corners.

All these problems arise as a result of binary noise displacing ends of straight edge segments and therefore changing their directions. It is much more difficult to find a solution for these problems, as the interpretation of changes in the contour direction is very context-dependent. Depending on where in the contour it appears and on its relative size, an element, consisting of two connected non-parallel straight segments, can be interpreted as a corner, as a part of a curved line, or as a deviation from the actual contour. It is also clear that the interpretation is highly scale-dependent.

A proper solution of the above problems requires the introduction of scale-space representations of curves ([120], [63], and [64]). Although most approaches to the scale-space representation use the Gaussian smoothing kernel, [93] presents a completely symbol-based approach which is compatible with our recognition framework. The implementation of the scale-space symbolic representation of curves is one the topmost priorities for the future work.

At the moment we tackle the problem of smoothing of noise-disturbed contours by introducing special nodes (CVX2 and CCV2, see Appendix B) in the knowledge base. These nodes define a combination of two convex (respectively, two concave) curves with a small concavity (resp. convexity) between them. This mechanism facilitates correct recognition of convex and concave curves in the presence of moderate noise.

Unfortunately, we didn't manage to find a similar ad hoc solution for constant curvature lines and we don't use those lines in letter definitions (section 10.1.2). It turned out though, that the system could properly recognise handwriting even when constant curvature lines are not used.

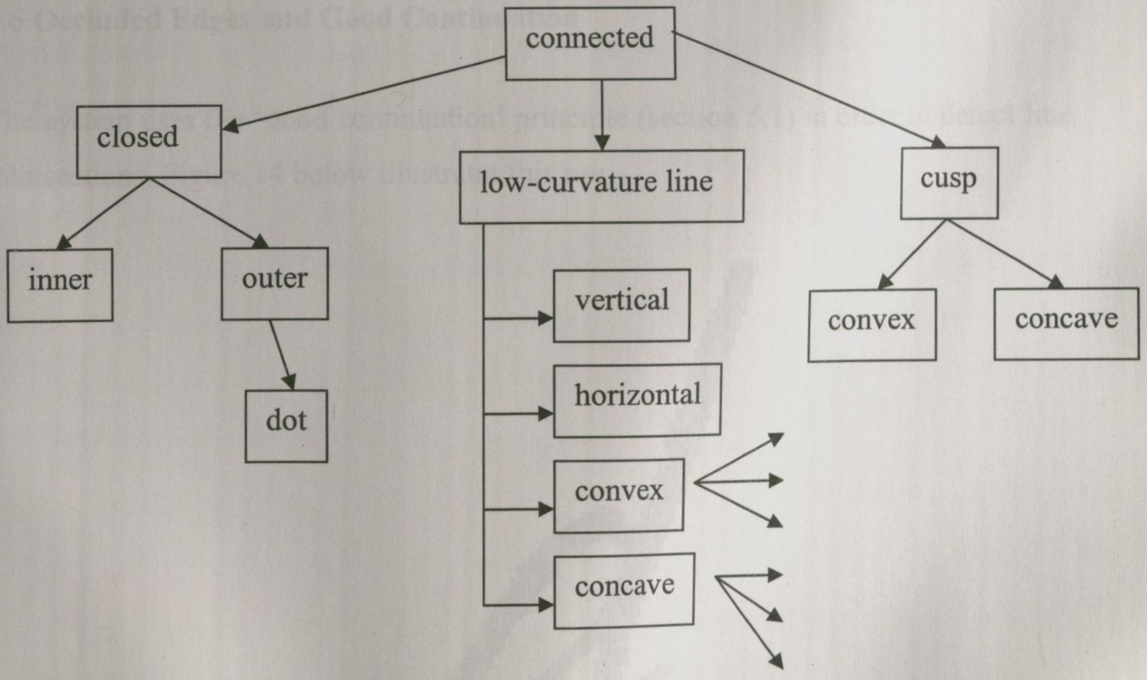


Figure 13 Hierarchy of connected lines

9.5 Curves, Contour Smoothing and Multiple Scale Representations

The low-level description of edges in the RKB contains a few levels of abstractness. At the highest level is the description of edges based on the property of connectedness. Any two edge segments at this level can be combined into a new edge if the end-point of one of them coincides with the start-point of the other. This definition ignores obviously any issues related to edge directions and allows the system to detect closed contours of figures.

More specific definitions take into account such properties of edges as the mutual and absolute orientation of edge segments, convexity or concavity of their combination, approximated local curvature and so on. Based on those properties a simple connected edge can be represented as a set of partitions into instances of more specific edge concepts. This representation allows detection of any properties of edges, from most abstract topological, such as closedness, to most specific metric, such as exact length and curvature. For instance, in order to be able to describe qualitatively the shape of curved edges without sharp corners, the system splits edges into low-curvature and high-curvature parts (cusps).

9.6 Occluded Edges and Good Continuation

The system uses the ‘good continuation’ principle (section 5.1) in order to detect line intersections. Figure 14 below illustrates this use.

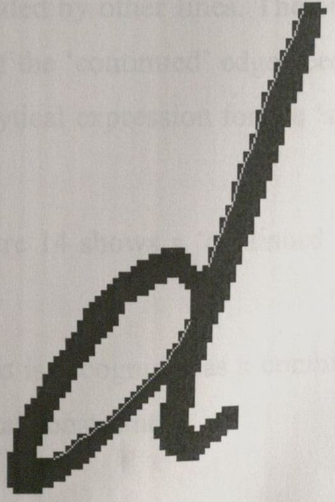


Figure 14 Good continuation principle and line intersections

The below fragment of the recognition knowledge base contains the definitions of nodes responsible for the implementation of the ‘good continuation’ principle[9]:

```
Corner1 = LCL += ConcaveCusp

Corner2 = ConcaveCusp += LCL

Cont_candidate_1 = LCL + LCL
 WHERE  #corner1:0
 SPECIALISATION OF LCL

Cont_candidate_2 = LCL + LCL
 WHERE  #corner2:0
 SPECIALISATION OF LCL

Continued = Cont_candidate_1 +~ Cont_candidate_2
 WHERE  @0.endAngle -^ @1.begAngle in {PI/6}
 AND          dist(@0.end,@1.beg)*2 < max(@0.len,@1.len) ~ 1
 AND          angle(@0.end,@1.beg) -^ @0.endAngle in {$maxAngle}
```

The above fragment defines 5 nodes of the type ‘Whole’. The first two nodes contain definitions of ‘concave corners’ – abrupt changes in the direction of low curvature lines (LCL). The order of concatenation of the cusp and the line in those definitions is important; hence we need two of them.

The next two nodes define candidate LCLs. This is an example of the context-dependent category definition (section 7.8). The meaning of the ‘WHERE’ clause in those definitions is that a candidate LCL must have a parent of the type ‘Corner1’ or ‘Corner2’ respectively. This contextual condition allows filtering out the majority of (approximately) collinear pairs of low curvature lines and focus only on those which can really be parts of an occluded edge. Finally, the last node defines the category of ‘continued’ edges, i.e. edges occluded by other lines. The ‘+~’ sign in the definition informs the system that the parts of the ‘continued’ edge need not to be adjacent. The ‘WHERE’ clause contains an analytical expression for the ‘smoothness’ condition of the ‘good continuation’ principle.

The thin white line in Figure 14 shows a ‘continued’ edge discovered by the system in the image.

An intersection of two lines is recognised as a combination of four occluded edges (with corresponding positional constraints).

[9] Note that the ‘good continuation’ principle is implemented as a part of the recognition knowledge base, not a hard-coded feature of the recognition algorithm.

9.7 Gestalt Grouping Principles

In the previous section we already saw how the use of the 'good continuation' principle facilitates detection of edge occlusion and line intersections. In this section we shall discuss the way other grouping principles (see section 5.1) influence the low-level image processing, as well as the implementation of the principles in our system.

The most important principle in this context is proximity. It is used to combine disconnected letters into words, to recognise letters containing disconnected parts (e.g. lowercase 'i'). Finally, it is implemented as a built-in job priority criterion: the closer are the parts of an object to each other the higher is the priority of their corresponding 'TEST' job (see section 8.3.4). Size similarity underlies the scale space based representations discussed in section 9.5. Orientation similarity is used when letters are combined into words (section 10.1.2).

Some criteria, such as colour similarity and the common fate principle are not applicable in the domain of handwriting recognition.

The most important difference between traditional visual pattern recognition systems and the low-level processing algorithm described in this chapter is that the latter doesn't make final segmentation decisions, but rather submits a set of possible edge interpretations to higher levels of the image processing. This set is also very different from elementary edge segment lists used in some systems as it contains a wide spectrum of edge interpretations – from most abstract such as closed contours of figures, to most specific, such as a straight-line edge segment. Moreover, the generation of some segmentations with low probability can be delayed by the system until contextual top-down influences will not 'unblock' them. This mechanism allows recognition of rare 'exotic' situations without adding overhead to the system.

This feature of our recognition system doesn't require any special interfaces between low-level and high-level modules, as those modules just don't exist. The segmentation reiteration process employed by some advanced recognition systems in order to improve system performance is also not required. All this is achieved by the standard recognition algorithm described in Chapter 8 as a simple consequence of its properties.

Chapter 10
Cursive Handwriting Recognition

In order to prove viability of our recognition system we applied it in a notoriously difficult domain – the recognition of unconstrained cursive handwriting. In this chapter we present the test system we developed for this purpose and describe an extension of the recognition knowledge base allowing the recognition of letter parts, letters, and words. We conclude the chapter with a presentation of some examples of correctly and incorrectly recognised input and discuss some methods to fix recognition errors.

10.1 Cursive Handwriting Recognition Framework

In order to prove practical usefulness of the general ideas introduced in previous chapters, we created a cursive handwriting recognition system. The program is written in C++ using Microsoft® Foundation Classes and runs under the Windows® XP operating system. The main window of the program with the input bitmap window, the output result window and controls is presented in the Figure 15.

This program was developed only for the demonstration of the viability of our approach. We made a number of simplifying assumptions about the nature of input images and didn't fulfil any comparative studies of the system performance, as at the current stage of the program development those studies would be premature. In the future, when the system's noise tolerance is improved, we plan to compare our system to a number of existing handwriting recognition systems, such as [21], [94], and [35].

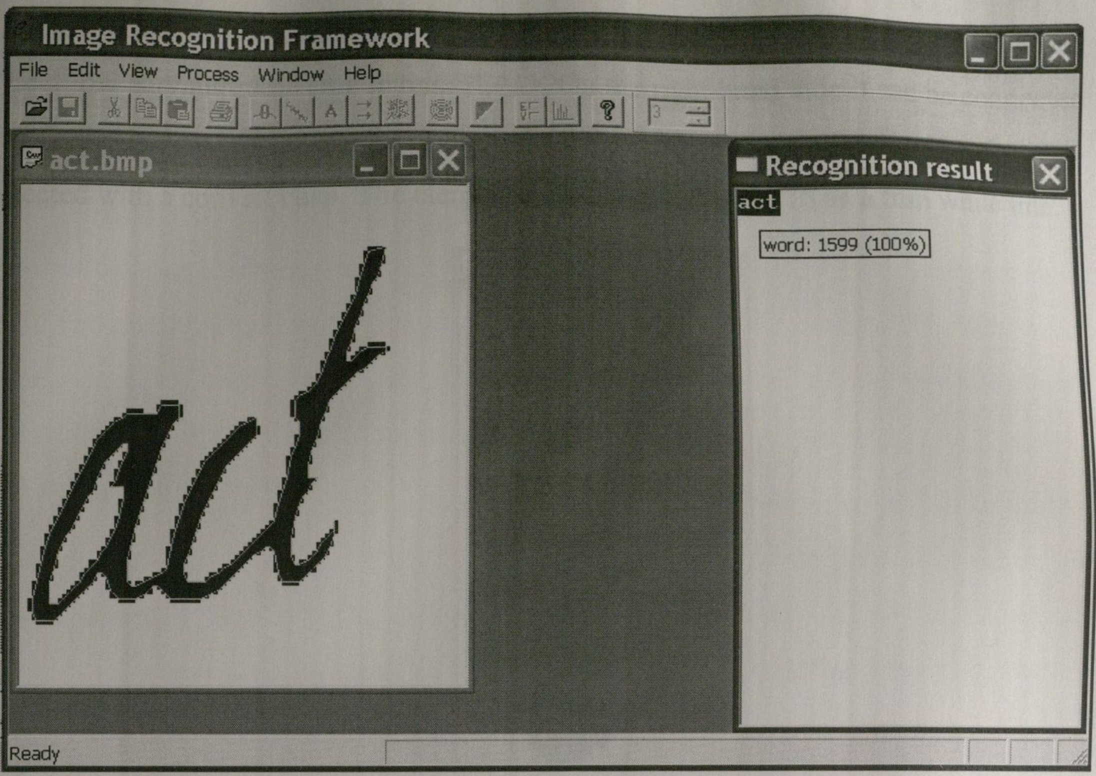


Image Recognition Framework
File Edit View Process Window Help
act.bmp
Recognition result
Ready

act

word: 1599 (100%)

Figure 15 Cursive Handwriting Recognition Framework

10.1.1 From Lines to Letter Elements

Contemporary handwriting is based on the use of pens, pencils, etc. The most distinctive feature of these instruments is that they produce (relatively) thin lines, which constitute characters and words. One can imagine writing systems based on a different means, e.g. a writing system where meaning is encoded by changes of colour. If we had such a system we would have to augment our low-level recognition knowledge base with corresponding knowledge. Fortunately, it is not the case, and the low-level elements present in our recognition knowledge base – curved edges, cusps, line intersections, etc. - suffice for further construction of letter elements from them.

Appendix B contains knowledge base descriptions of letter elements and letters. Here we shall discuss some typical examples.

Letter elements can be divided into a few basic groups. The first group consists of the elements, which are defined by their absolute orientation. An example of such an element is *rtail_left*, which denotes a convex curve at the bottom of the letters '*i*', '*l*', etc. This element is used in many letter definitions, specifically for those letters, which have down-to-up right-hand side connections. The element is defined by the ranges for absolute angles of its beginning and its end. This element can be connected on the left-hand side with a concave cusp (the letters '*a*','*d*','*u*') or with a vertical low curvature line (the letters '*i*', '*l*'). On the right-hand side it can be connected with the *bottom-left* connection point of the next letter in the word, or can be connected with a convex cusp. The element is depicted in Figure 16 as a thin white line.

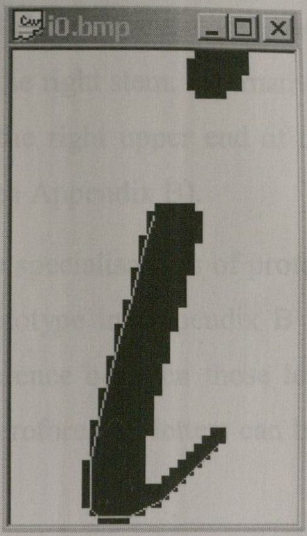


Figure 16 The *rtail_left* letter element

Another example of an element with absolute orientation is a vertical poll (see *vpoll_1* and *vpoll_2* in Appendix B)

The second category of the letter elements comprises elements with intersections, such as a loop, a horizontal bar intersection in the letter '*t*', etc.

Finally, the third group consists of closed contour elements such as dots and holes.

10.1.2 Letters and Words

Letters are constructed from letter parts described in the previous section. Since handwriting in all European languages uses horizontal lines written from left to right, we introduced common elements in all handwritten letters, namely connections of letters. Letters in words are connected from left to right and those connections are approximately horizontal. Therefore, we introduced four optional connection points for definitions of connected letters – top-left, bottom-left, top-right and bottom-right. Any pair of connection points (left or right) can be absent if the letter is not connected at the corresponding side.

Other common letter attributes include the baseline position and direction, x-height, and slant. Not all letter definitions can provide this information, e.g. in the definition of the letter ‘l’ we don’t know the x-height attribute.

Most letters are defined by a few prototypes differing in some topological properties. For instance, the letter ‘*a*’ can have a closed contour inside when the upper left curved line is connected with the right stem. Alternatively, it can be written as the ‘cl’ connected combination with the right upper end of the ‘c’ character positioned sufficiently close to the ‘l’ stem (see Appendix B).

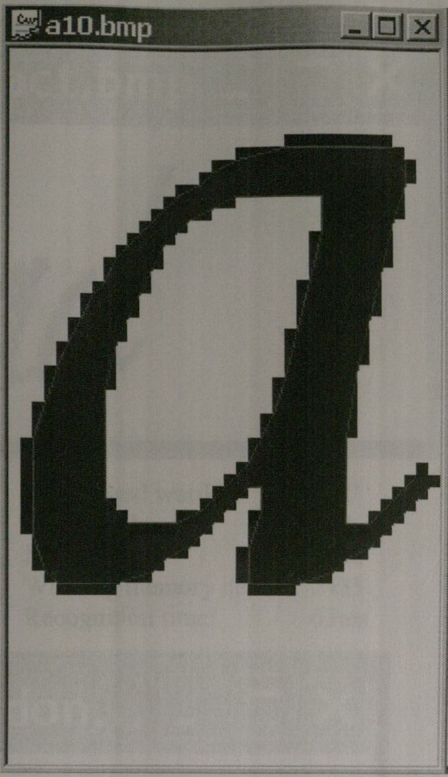
Some letters are defined as specialisations of prototypes common for a set of letters. An example of such a prototype in Appendix B is the Lad node describing both letters ‘*a*’ and ‘*d*’. The difference between those letters is only in the relative sizes of the stem and the loop. Therefore, the letters can be defined as specialisations of the same prototype.

Some specialisations of this type require contextual information. The letters ‘l’ and ‘i’ (without the dot) can be distinguished only by their size relative to the x-height parameter of the word they are imbedded into.

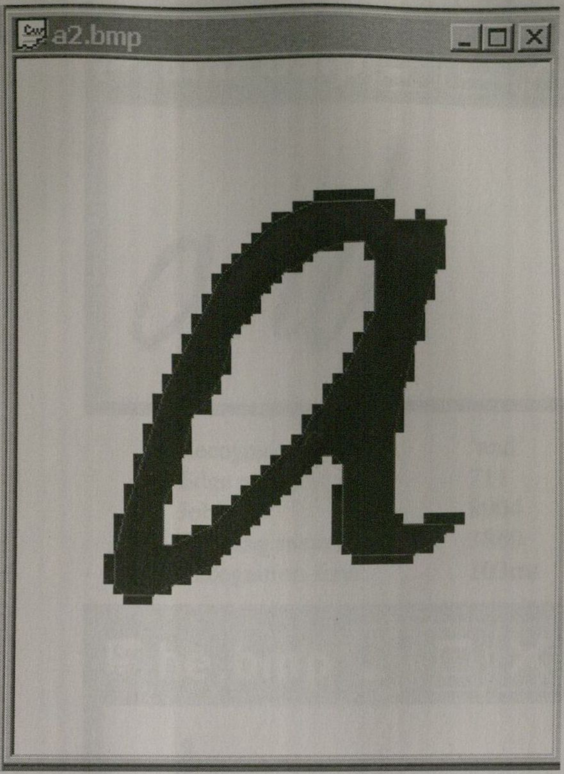
Letters are combined into words based on their connectedness or proximity. Words themselves are not considered as results of the recognition process; rather, they serve as context objects for the ambiguity resolution and filtering out inconsistent in size, direction, and position letter instances. Word attributes include the averaged baseline position and direction, the averaged x-height and slant.

10.2 Experimental Results

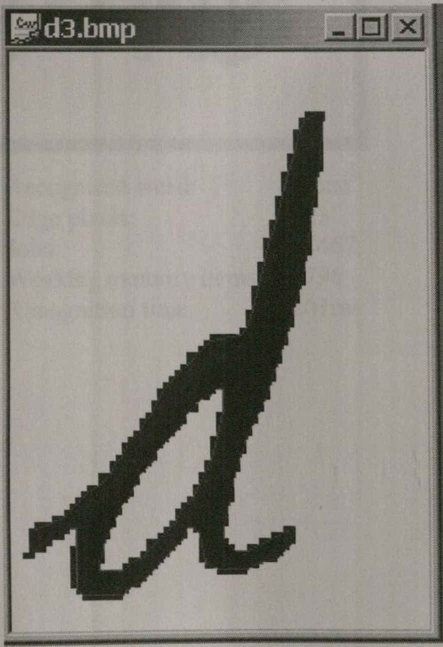
In order to test our system we used samples of handwriting gathered by A. Senior for his Ph.D. thesis [94] and then made publicly available. In this section we shall present images of some successfully recognised letters and words along with some performance statistics data. All measurements were performed on a PC compatible desktop computer with the 1.9 GHz Intel IV processor and 768 MB of RAM.



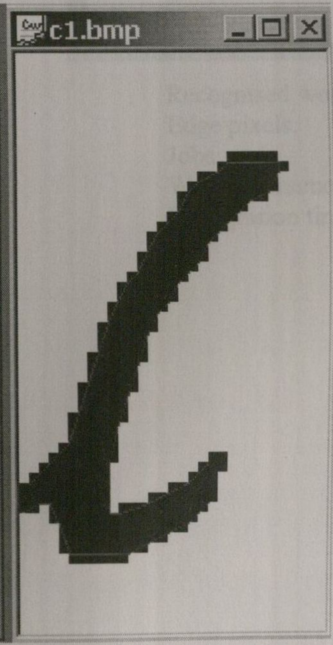
Recognised letter:	‘a’
Edge pixels:	197
Jobs:	1947
Working memory items:	849
Recognition time:	17ms



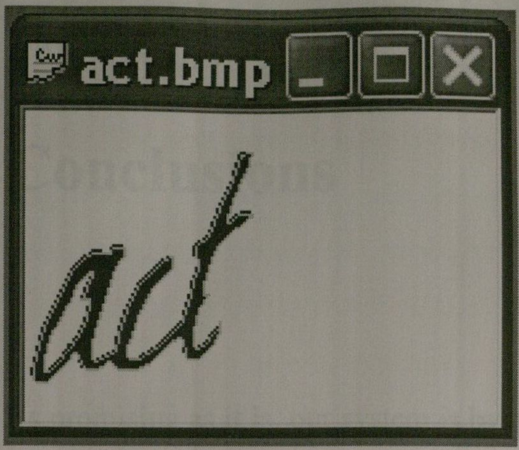
Recognised letter:	‘a’
Edge pixels:	201
Jobs:	1891
Working memory items:	838
Recognition time:	15ms



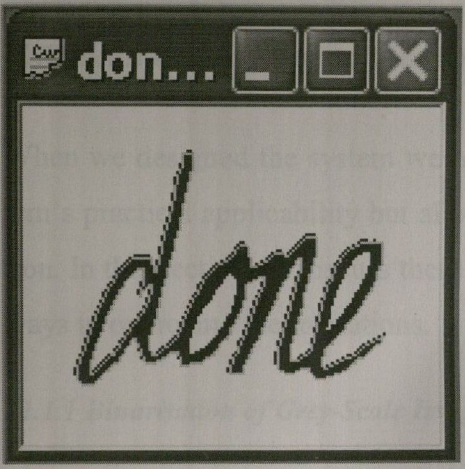
Recognised object:	‘d’
Edge pixels:	298
Jobs:	3265
Working memory items:	1337
Recognition time:	28ms



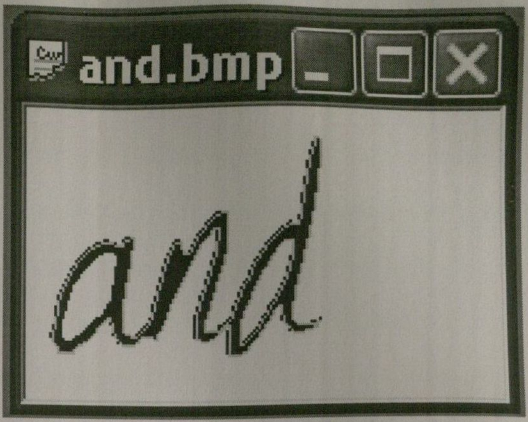
Recognised object:	‘c’
Edge pixels:	114
Jobs:	1416
Working memory items:	551
Recognition time:	9ms



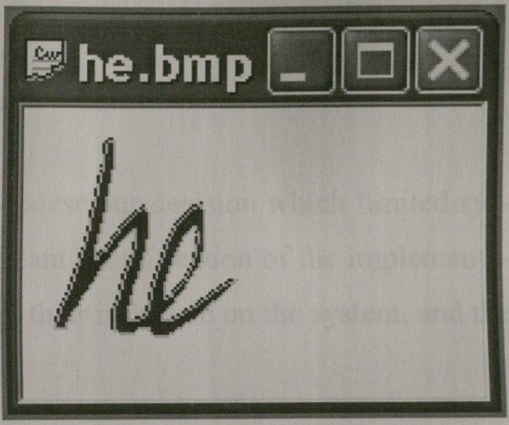
Recognised word:	'act'
Edge pixels:	537
Jobs:	6356
Working memory items:	2485
Recognition time:	63ms



Recognised word:	'done'
Edge pixels:	923
Jobs:	13467
Working memory items:	5796
Recognition time:	201ms



Recognised word:	'and'
Edge pixels:	711
Jobs:	8904
Working memory items:	3860
Recognition time:	103ms



Recognised word:	'he'
Edge pixels:	488
Jobs:	6667
Working memory items:	2908
Recognition time:	65ms

Chapter 11

Conclusions

As promising as it is, our system is by no means perfect and complete. In this chapter we examine the current limitations of the system and potential ways to overcome those limitations. Then we discuss future directions for the system development. Finally, we summarise the thesis.

11.1 Limitations of Current Implementation

When we designed the system we made a few conscious decision which limited system's practical applicability but allowed significant simplification of the implementation. In this section we discuss these limitations, their influence on the system, and the ways to overcome the limitations.

11.1.1 Binarisation of Grey-Scale Images

Current implementation of our recognition system processes only binary images. This is a significant technical simplification. We also implemented a number of algorithms converting grey scale images into binary images. Some of those algorithms are quite sophisticated and produce much better results than the primitive global threshold method (see [105]). Still, even the best binarisation algorithms produce extra noise and lead to loss of information in the image.

As argued in [114], binarised images don't reflect the structure of the initial document, because it can have coloured or textured background (e.g. magazines, business forms, checks), or various colours of ink can be used. Besides, the imaging process can introduce miscellaneous noise, for instance non-uniform illumination, non-uniform paper reflection, convolution distortion because of the point spread function of the scanner, etc. All this leads to the input image containing a wide range of grey scale pixels. Binarisation of such an input image leads to significant loss of information, e.g. touching or broken characters.

The best solution of this problem would be not to improve binarisation methods, but to work directly with grey-scale images. It will require significant changes and extensions to template nodes (section 7.2.2) in the recognition knowledge base as well as some other low-level processing related nodes. The advantage of this change is that will allow drastic expansion of the applicability area of the recognition systems. We plan to include this amendment in the future versions of the system.

11.1.2 Restricted Set of Symbols

Currently the recognition knowledge base contains descriptions of only 26 lower-case Latin letters. Even for those letters not all their variations are implemented. A practically usable system must obviously have descriptions for many more characters: upper-case letters, digits, punctuation marks, etc. There is nothing magical in adding this information to the recognition knowledge base, rather - significant amount of tedious work.

11.1.3 Generality vs. Performance

In order to preserve theoretical 'purity' of the implementation all the recognition processes in our system are expressed in the recognition knowledge base. It doesn't have to be so. There exist many fast and reliable methods for edge extraction ([13]), curve and straight line detection ([88], [118]), which work faster than the general algorithm of our system. These methods can be integrated into the system to increase its speed.

Expressions for the constraint and attribute calculation (section 7.5) are interpreted during the recognition process. This is necessary to make experimentation with the system easier (when an expression is changed, the program is not recompiled, rather the recognition knowledge base is re-loaded). In a commercial system it is possible to compile those expression into machine codes to make their evaluation and overall processing significantly faster.

11.2 Future Directions

In this section we focus on the possible directions for future research, which can significantly improve the system and bring it even closer to the human level of recognition performance.

11.2.1 Learning

One of the main aspirations for the whole project, as discussed in section 6.4, was to create a system capable of learning from experience. This issue has not been addressed directly in the thesis because current implementation of the system cannot learn. We plan to augment the system with learning capabilities in future versions.

This problem is not an easy one to solve. The learning process that we foresee for our system should not be restricted to a (relatively) simple adjustment of numerical parameters in order to improve system speed. Nor is it a kind of grammatical inference process extending the knowledge base of the system. Rather, the learning process should combine features of the two and be able to infer the following information from training data:

- new nodes of type 'Whole' with constraint expressions and expressions to calculate attributes of the node;
- new 'Class' nodes;
- priority values for the links of new nodes;
- steepness values for fuzzy class membership functions;
- intensional 'abstraction-of' relations between nodes.

Creation of an algorithm capable of inferring all this information is a very challenging problem. Some attempts in this direction are reported in [17] and [7].

11.2.2 Neuronal Implementation

Contemporary approach to the modelling of human perceptual and cognitive activities calls for the abstraction of the algorithm implementation, as it is stated, for instance, in [58] and [5]. However, recent developments in cognitive science suggest that the way the human mind is embedded into the human brain do matter for the functioning of the mind (see [82], [50]). Therefore, we believe that for our recognition approach to become a real model of human visual perception it has to be implemented on the neural network basis. After all, the one thing that we know for sure about the human brain is that it consists of a network of interconnected neurons.

We already noted that our algorithm bears certain similarity with the neuronal-like model of word recognition introduced by McClelland and Rumelhart in [60] (see also [19], [20], and [21] for a recent adaptation of this model for cursive script recognition). This similarity suggests that there should exist a mapping of our algorithm to the neuronal architecture. The main problem of this mapping is an explicit representation of the 'part/whole' relation in the neural network. Despite many efforts, a satisfactory solution of this problem has not been found yet.

11.3 Summary

In the thesis we presented a pattern recognition system designed according to fundamental principles of human visual perception and its application in the cursive handwriting recognition domain.

The system is based on the blackboard architecture and contains a knowledge base storing information about patterns being recognised in the structured manner. The knowledge base is represented by a graph with nodes corresponding to partial graphical concepts, their aggregates, and their abstractions (intensional and extensional). The knowledge base also contains links describing relationships between nodes (such as 'part-of', 'subclass-of', and 'abstraction-of'), spatial constraints for parts constituting a node, and descriptions of node attributes.

We provided a detailed description of the recognition algorithm. The algorithm consists of elementary jobs, which are scheduled by the system job priority queue. These jobs are used to find candidate sets of items, test their compatibility, and create new items. We also described the system job scheduling strategy, which consists of two virtual processes – the bottom-up process and the top-down process.

Then we presented the application of the system to the low-level visual processing, such as noise reduction, creating edges from pixels and lines from edges. Then we described the specific implementation of the system for recognition of cursive handwriting and experimental results obtained with this implementation.

Finally, we discussed the limitations of the current system implementation and potential directions for future research.

Bibliography

1. Abuhaiba, I.S.I., Ahmed, P.: A Fuzzy Graph Theoretic Approach to Recognize the Totally Unconstrained Handwritten Numerals. Pattern Recognition, vol. 26, no. 9, pp. 1335-1350, (1993)
2. Al-Badr, B., Haralick, R.M.: Segmentation-Free Word Recognition with Application to Arabic. ICDAR'95: Third International Conference on Document Analysis and Recognition, Montreal, Canada, August 14-16 (1995)
3. Anderson, J.R.: Rules of the Mind. Lawrence Erlbaum Associates, Hillsdale, NJ (1993)
4. Biederman, I.: Recognition-by-Components: A Theory of Human Image Understanding. Psychological Review, Vol. 94, No. 2, pp. 115-147 (1987)
5. Binford, T.O.: Visual perception by computer. Paper presented at the IEEE System Science and Cybernetics Conference, Miami, FL, December, (1971)
6. Binford, T.O., Levitt, T.S., Mann, W.B.: Bayesian Inference in Model-Based Machine Vision. In: Kanal, L.N., Levitt, T.S., Lemmer, J.F.(eds.): Uncertainty in AI 3. North Holland, New York, (1989)
7. Bischof, W. F., Caelli, T.: Visual Learning of Patterns and Objects. IEEE Transactions on Systems, Man and Cybernetics, 27, 907-917, (1997)
8. Boff, K.R., Kaufman, R., Thomas, J.P. (eds.): Handbook of Perception and Human Performance, Vol. II: Cognitive Processes and Performance. John Wiley and Sons, New York Chichester Brisbane Toronto Singapore (1986)
9. Bozinovic, R.M., Srihari, S. N.: Off-Line Cursive Script Word Recognition, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 11, no. 1, pp. 68-83, 1989.
10. Brachman, R.J., Schmolze, J.G.: An Overview of the KL-ONE Knowledge Representation System. Cognitive Science, Vol. 9, pp. 171-216, (1985)
11. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth International Group, Belmont, CA, (1984)
12. Bülthoff, H. H., Edelman, S. Y., Tarr, M. J.: How are three-dimensional objects represented in the brain? Cerebral Cortex, Vol. 5, No. 3, pp. 247-260, (1995)
13. Canny, J.: A Computational Approach to Edge Detection, IEEE Transactions on PAMI, Vol. 8, No. 6, pp. 679-698, (1986)
14. Carver, N., Lesser, V.: The Evolution of Blackboard Control Architectures. Expert Systems with Applications, Special Issue on The Blackboard Paradigm and Its Applications, Vol. 7, No. 1, pp. 1-30, (1994)
15. Casey, R., Lecolinet, E.: A Survey of Methods and Strategies in Character Segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), Vol. 18, No. 7, pages 690-706, July 1996.

16. Chan, K., Yeung, D.: Recognizing On-line Handwritten Alphanumeric Characters through Flexible Structural Matching. Pattern Recognition, Vol. 32, pp. 1099-1114 (1999)

17. Cho, K., Dunn, S. M.: Learning Shape Classes. IEEE Transactions on PAMI, Vol. 16, No. 9, pp. 882-888, (1994)

18. Corkill, D.: Blackboard Systems, AI Expert, 6(9):40-47, September, 1991.

19. Côté, M.: Utilisation d'un modèle d'accès lexical et de concepts perceptifs pour la reconnaissance d'images de mots cursifs. Thèse de l'Ecole Nationale Supérieure des Télécommunications (ENST), June, (1997)

20. Côté, M., Lecolinet, E., Cheriet, M., Suen, C.Y.: Using Reading Models for Cursive Script Recognition. In: Simner, M.L., Leedham, C.G., Thomassen, A.J.W.M. (eds.): Handwriting and Drawing Research: Basic and Applied Issues, pp. 299-313, IOS Press, Amsterdam, (1996)

21. Côté, M., Lecolinet, E., Cheriet, M., Suen, C.Y.: Automatic reading of cursive scripts using a reading model and perceptual concepts. The PERPECTO system. International Journal of Document Analysis and Recognition, Vol. 1, No. 1, pp. 3-17, (1998)

22. Dill, M., Wolf, R., Heisenberg, M.: Visual Pattern recognition in *Drosophila* involves retinotopic matching. Science, vol. 365, pp. 751-753, (1993)

23. Dzuba, G., Filatov, A., Gershuny, D., Kil, I., Nikitin, V.: Check amount recognition based on the cross validation of courtesy and legal amount fields. Int. Journal of Pattern Recognition and Artificial Intelligence, Vol. 11, No. 4, pp. 639-655, (1997)

24. Dzuba, G., Filatov, A., Volgunin, A.: Handwritten ZIP Code Recognition, ICDAR'97, Proceedings of 4th International Conference on Document Analysis and Recognition, Ulm, Germany, pp.766-770, (1997)

25. Edelman, S., Flash, T., Ullman, S.: Reading cursive handwriting by alignment of letter prototypes. International Journal of Computer Vision, Vol. 5, No. 3, pp. 303-331, (1990)

26. Engelmore, R.S., Morgan, A. (eds.): Blackboard Systems, Addison-Wesley, (1988)

27. Feder, J.: Plex languages. Information Sciences, Vol. 3, pp. 225-241, (1971)

28. Filatov, A., Gitis, A., Kil, I.: Graph-based Handwritten Digit String Recognition, ICDAR'95, Proceedings of 3rd International Conference on Document Analysis and Recognition, Montreal, Canada, August 14-16, pp.845-848, (1995)

29. Flasinski, M.: Mathematical Linguistics Models for Computer Vision, Machine GRAPHICS & VISION, vol. 5, ½,1996, pp.87-97.

30. Fu, K.S.: Syntactic Pattern Recognition and Applications, Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

31. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co., New York, (1979)

32. Gibson, E.J., Levin,H.: The psychology of reading. MIT Press, Cambridge, Massachusetts, (1975)

33. Gleitman, L.R., Rosin, P.: The structure and acquisition of reading I: Orthographies and the structure of language. In: Reber, A.S., Scarborough, D.L. (eds.): Toward a psychology of reading. Erlbaum, Hillsdale, NJ, (1977)

34. Gregg, J.R., Zoubek, C.E., Condon, G.: Gregg Shorthand Dictionary (abridged). Gregg/Community College Div., (1995)

35. Guillevic, D.: Unconstrained Handwriting Recognition Applied to the Processing of Bank Cheques, Doctoral thesis, Computer Science Department, Concordia University, Montreal, September 1995.

36. Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching. In: Yormark, B., (ed.): SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, pp. 47-57, (1984)

37. Haralick, R., Kanungo, T.: Model-based Character Recognition, DARPA Workshop on Document Understanding, pp.1-5, Palo Alto, USA, May 6-8, 1992.

38. Haralick, R.M., Shapiro, L.G.: Computer and Robot Vision. Vol. 1, Addison Wesley (1992)

39. Harmon, L.D.: Scanning the Issue. Proceedings of the IEEE, v.60, No. 10, p.1117 (1972)

40. Haykin, S.: Neural Networks. A Comprehensive Foundation. Macmillan College Publishing Inc., New York, (1994)

41. Hoffman, D. D., Richards, W. A.: Parts of recognition. Special Issue: Visual cognition. Congnition, Vol. 18, No. 1-3, pp. 65-96, (1984)

42. Hofstadter, D., McGraw, G.: Letter Spirit: An Emergent Model of the Perception and Creation of Alphabetic Style, Indiana University, CRCC Technical Report 68, 1993.

43. Hofstadter, D.: Fluid Concepts and Creative Analogies. Allen Lane, The Penguin Group, London New York Ringwood Toronto (1997)

44. Hummel, J., Biederman, I.: Dynamic binding in a neural network for shape recognition. Psychological Review, Vol. 99, pp. 480-517, (1992)

45. Jacobs, A.M., Grainger, J.: Models of visual word recognition: sampling the state of the art. Journal of Experimental Psychology: Human Perception and Performance, Vol. 20, No.6, pp.1311-1334, (1994)

46. Johnson. T.R.: Control in Act-R and Soar. In: Shafto, M., Langley, P.(eds.): Proceedings of the Nineteenth Conference of the Cognitive Science Society, pp. 343-348 (1997)

47. Jolicoeur, P., Gluck, M. A., Kosslyn, S. M.: Pictures and names: Making the connection. Cognitive Psychology, Vol. 16, No. 2, pp. 243-275, (1984)

48. Kellman, P. K., Spelke, E. S.: Perception of partly occluded objects in infancy. Cognitive Psychology, Vol. 15, No. 4, pp. 483-524, (1983)

49. Kummert, F., Niemann, H., Prechtel, R., Sagerer, G.: Control and explanation in a signal understanding environment. Signal Processing, Vol. 32, pp.111-145, (1993)

50. Lakoff, G., Johnson, M.: Philosophy in the Flesh. Basic Books, New York, (1999)

51. Laird, J., Newell, A., Rosenbloom, P.S.: SOAR: An architecture for general intelligence. Artificial Intelligence, Vol. 33, pp. 1-64 (1987)

52. LaMarca, A., Ladner, R.: The Influence of Caches on the Performance of Heaps. Technical Report UW-CSE-96-02-03. Department of Computer Science, University of Washington, Seattle, (1996)

53. Lecolinet, E., Baret, O.: Cursive Word Recognition: Methods and Strategies, in Fundamentals in Handwriting Recognition, S. Impedovo Ed., pages 235-263, NATO ASI Series F: Computer and Systems Sciences, Vol. 124, Springer Verlag, 1994.

54. Lim, G.: Visual Object Shape Recognition Using Hierarchical Syntax Extraction. Ph.D. Thesis, The University of Western Australia, Nedlands, Australia (1997)

55. Lu, Y., Shridhar, M.: Character Segmentation in Handwritten Words - an Overview, Pattern Recognition, vol. 29, no. 1, pp. 77-96, 1996.

56. Luschei, E. C.: The Logical Systems of Lesniewski. North Holland Publishing Co., Amsterdam, London, (1962)

57. Marín, M.: An Empirical Comparison of Priority Queue Algorithms. Technical report PRG-TR-10-97, Oxford University, (1997)

58. Marr, D.: Vision. Freeman, San Francisco, (1982)

59. Marr, D., Nishimara, H.K.: Representation and recognition of the spatial organization of three dimensional structure. Proceedings of the Royal Society of London B, Vol. 200, pp. 269-294, (1978)

60. McClelland, J.L., Rumelhart, D.E.: An interactive activation model of context effects in letter perception. Psychological Review, Vol. 88, pp. 375-407, (1981)

61. McGraw, G.: Emergent High-Level Perception of Letters Using Fluid Concepts, Ph.D. Thesis, Indiana University, 1995.

62. Messmer, B., Bunke, H.: A decision tree approach to graph and subgraph isomorphism detection, Pattern Recognition, Vol. 32, No. 12, pp. 1979 – 1998, (1999)

63. Mokhtarian, F., Mackworth, A.K.: Scale-based description and recognition of planar curves and two-dimensional shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 8, pp. 34-43, (1986)

64. Mokhtarian, F., Mackworth, A.K.: A theory of multiscale, curvature-based shape representation for planar curves. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 14, pp. 789-805, (1992)

65. Mori, S., Suen, C.Y., Yamamoto, K.: Historical Review of OCR Research and Development, Proceeding of the IEEE, vol. 80, no. 7, pp. 1029-1058, 1992.

66. Moses, Y., Adini, Y., Ullman, S.: Face recognition: the problem of compensating for illumination chages. Proceedings of the European Conference on Computer Vision, pp. 286-296, (1994)

67. Newell, A.: Unified Theories of Cognition. Harvard University Press, Cambridge, MA, (1990)

68. Niemann, H., Brünig, H., Salzbrunn, R., Schröder, S.: Interpretation of industrial scenes by semantic networks. In Proc. IAPR Int. Workshop on Machine Vision Applications, pages 39-42, Tokyo, 1990

69. Niemann, H., Sagerer, G., Schröder, S., Kummert, F.: ERNEST: A semantic network system for pattern analysis. IEEE Trans. Pattern Analysis and Machine Intelligence, 9:883-905, 1990

70. Nii, H. P.: Blackboard Systems: The blackboard model of problem solving and the evolution of blackboard architectures. AI Magazine, Vol. 7, No. 2, pp. 38-53 (1986)

71. Palmer, S.E.: Hierarchical Structure in Perceptual Representation, Cognitive Psychology, vol.9, pp. 441-474, (1977)

72. Palmer, S.E.: Structural Aspects of Visual Similarity, Memory & Cognition, vol. 6, no. 2, pp. 91-97, (1978)

73. Palmer, S.E.: Vision Science. Photons to Phenomenology. Bradford Book. The MIT Press, Cambridge London (1999)

74. Palmer, S. E., Rock, I.: On the nature and order of organizational processing: A reply to Peterson. Psychonomic Bulletin & Review, Vol. 1, pp. 515-519, (1994)

75. Parizeau, M.: Reconnaissance d'écriture cursive par grammaires floues avec attributs: étape vers la conception d'un bloc-notes électronique. Ph.D. Thesis, Ecole Polytechnique de Montréal, (1992)

76. Parizeau, M., Plamondon R.: A Fuzzy Syntactic Approach to Allograph Modelling for Cursive Script Recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 17, No. 7, pp. 702-712, (1995)

77. Parizeau, M., Plamondon, R., Lorette, G.: Fuzzy-Shape Grammar for Cursive Script Recognition, in Advances in Structural and Syntactic Pattern Recognition, H. Bunke (Ed.), World Scientific Publishing, Singapore, New Jersey, London, Hong Kong, pp. 320-332, (1993)

78. Pasternak, B.: Adaptierbares Kernsystem zur Interpretation von Zeichnungen Motivation - Entwurf – Realisierung. Dissertation zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.), Fachbereich Informatik der Universität Hamburg, (1996)

79. Pavlidis, T.: Structural Pattern Recognition. Springer Series in Electrophysics I. Springer Verlag, New York, (1977)

80. Pearce, A. R., Caelli, T., and Bischof, W. B.: Learning Relational Structures: Applications in Computer Vision. Applied Intelligence: The International Journal of Artificial Intelligence. Vol.4, pp. 257-268 (1994)

81. Pfleger, K., Hayes-Roth, B.: An Introduction to Blackboard-Style Systems Organization. Technical Report KSL-98-03, Knowledge Systems Laboratory, Stanford University, (1998)

82. Putnam, H.: Representation and Reality. The MIT Press, Cambridge, MA, (1989)

83. Quinlan, J.R.: Induction of decision tress. Machine Learning, Vol.1, No.1, pp. 81-106, (1986)

84. Quint, F.: MOSES: A Structural Approach to Aerial Image Understanding. In: Gruen, A., Baltsavias, E., Henricsson, O.: Extraction of Man-Made Objects from Aerial and Space Images, II. Birkhauser Verlag, Meeting held in Ascona, Switzerland, May 5-9, pp. 323-332, (1997)

85. Rosch, E.: Natural categories. Cognitive Psychology, Vol. 4, No. 3, pp. 328-350, (1973)

86. Rosch, E.: On the internal structure of perceptual and semantic categories. In Moore, T. E. (ed.): Cognitive development and the acquisition of language. New York, Academic Press, (1973)

87. Rosch, E.: Cognitive reference point. Cognitive Psychology, Vol. 7, No. 4, pp. 532-547, (1975)

88. Rosin, P. L., West, G. A.: Segmentation of Edges into Lines and Arcs. Image and Vision Computing, Vol. 7, No. 2, pp. 109-114, (1989)

89. Rocha, J., Pavlidis, T.: A Shape Analysis Model with Applications to a Character Recognition System, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 16, No. 4, pp. 393-404, (1994)

90. Rocha, J., Pavlidis, T.: Character Recognition Without Segmentation, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 17, no. 9, pp. 903-909, 1995

91. Salzbrunn, R., Niemann, H., Harbeck, M., Winzen, A.: Object recognition by a robust matching technique. In H. Bunke, editor, Advances in Structural and Syntactic Pattern Recognition, volume 5 of Series in Machine Perception and Artificial Intelligence, pages 481-495. World Scientific, 1992.

92. Sanoki, T., Bowyer K.W., Heath, M.D., Sarkar, S.: Are edges sufficient for object recognition. Journal of Experimental Psychology: Human Perception and Performance, Vol. 24, No. 1, pp. 1-10, (1998)

93. Saund, E.: Symbolic Construction of a 2-D Scale Space Image. IEEE Trans. On Pattern Analysis and Machine Intelligence, Vol. 12, No. 8, pp. 817-830, (1990)

94. Senior, A.: Off-Line Handwriting Recognition Using Recurrent Neural Networks, Ph.D. Thesis, University of Cambridge (1994)

95. Shapiro, L., Haralick, R.: Structural Descriptions and Inexact Matching, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-3, no. 5, pp. 504-519, 1981.

96. Shaw, A.C.: A formal picture description scheme as a basis for picture processing systems. Information Control, Vol. 14, pp. 9-52, (1969)

97. Simon, H.: The Sciences of the Artificial. MIT Press, Cambridge, Massachusetts, (1981)

98. Simon, J.: Off-Line Cursive Word Recognition, Proceedings of the IEEE, vol. 80, no. 7, pp. 1150-1161, (1992)

99. Spelke, E. S.:Principles of object perception. Cognitive Science, Vol. 14, No. 1, pp. 29-56, (1990)

100. Steinherz, T., Rivlin, E., Intrator, N.: Offline cursive script word recognition - a survey. International Journal of Document Analysis and Recognition, Vol. 2, No. 2/3, pp. 90-110, (1999)

101. Stilla, U., Michaelsen, E., Lütjen, K.: Structural 3D-Analysis of Aerial Images with a Blackboard-based Production System. In: Gruen, A., Kuebler, O. (eds.): Automatic Extraction of Man-Made Objects from Aerial and Space Images, Ascona Workshop (1995)

102. Stilla, U., Jurkiewicz, K.: Structural 3D-Analysis of Urban Scenes from Aerial Images. In: Kraus, K., Waldhäusel, P.(eds.): International Archives of Photogrammetry and Remote Sensing, Vol. 31, Part B3, p. 832-838 (1996)

103. Tarr, M.J., Bülthoff, H.H.: Is human object recognition better described by geon-structural-descriptions or by multiple-views? Journal of Experimental Psychology: Human Perception and Performance, Vol. 21, No. 6, pp. 1494-1505, (1995)

104. Tombre, K.: Structural and Syntactic Methods in Line Drawing Analysis: To which Extent do they Work? In: Advances in Structural and Syntactical Pattern Recognition (Proceedings of SSPR'96, Leipzig, Germany). Lecture Notes in Computer Science, Vol. 1121, Springer-Verlag, Berlin Heidelberg New York (1996) 310-321

105. Trier, Ø.D., Jain, A.K.: Goal-Directed Evaluation of Binarization Methods. IEEE Transactions on PAMI, Vol.17, No. 12, pp. 1191-1201, (1995)

106. Trier, Ø.D., Jain, A.K., Taxt, T.: Feature Extraction Methods for Character Recognition – a Survey, Pattern Recognition,Vol.29, No. 4, pp. 641-662, (1996)

107. Tversky, A.: Features of similarity, Psychological Review, vol. 84, pp. 327-352, 1977.

108. Tversky, A., Hemenway, K.: Objects, parts, and categories. Journal of Experimental Psychology: General, Vol. 113, pp. 169-193, (1984)

109. Uhr, L.: Pattern recognition. Wiley & Sons, New York, (1966)

110. Uhr, L.: Pattern Recognition, Learning and Thought: Computer-Programmed Models of Higher Mental Processes. Prentice Hall, Englewood Cliffs, NJ, 1973

111. Ullman, S.: High-level Vision. The MIT Press, Cambridge London (1996)

112. Vaughan, W., Jr., Greene, S.L.: Pigeon visual memory capacity. Journal of Experimental Psychology: Animal Behavior Processes, vol. 10, pp. 256-271, (1984)

113. Venguerov, M., Cunningham, P.: Generalised Syntactic Pattern Recognition as a Unifying Approach in Image Analysis. In: Amin, A., Dori, D., Pudil, P., Freeman, H., (eds.): Advances in Pattern Recognition. Lecture Notes in Computer Science, Vol. 1451. Springer-Verlag, Berlin Heidelberg New York (1998) 913–920

114. Wang, L., Pavlidis, T.: Direct Gray-Scale Extraction of Features for Character Recognition. IEEE Transactions on PAMI, Vol. 15, No. 10, pp. 1053-1067, (1993)

115. Watanabe, S.: Knowing and Guessing – A Quantitative Study of Inference and Information, John Wiley and Sons, Inc., (1969)

116. Webb, A.: Statistical Pattern Recognition. Arnold, London Sydney Auckland (1999)

117. Weisstein, N., Harris, C. S.: Visual detection of line segments: an object-superiority effect. Science, Vol. 186, No. 4165, pp. 752-755, (1974)

118. West, G.A., Rosin P.L.: Techniques for Segmenting Image Curves into Meaningful Descriptions. Pattern Recognition, Vol. 24, No. 7, pp. 643-652 (1991)

119. Winston, P.H.: Artificial Intelligence, Third Edition. Addison-Wesley, Reading, Menlo Park, New York, Wokingham, Amsterdam, Boon, Sydney. (1993)

120. Witkin, A.P.: Scale-space filtering. In Proceedings of the 8th International Joint Conference on Artificial Intelligence, pp. 1019-1022, (1983)

121. Wittenburg, K., Weitzman, L.: Relational Grammars: Theory and Practice in a Visual Language Interface for Process Modeling. In K. Marriott and B. Meyer (eds.), Visual Language Theory, Springer-Verlag, pp. 193-217, (1998)

122. Wittenburg, K.: Predictive Parsing for Unordered Relational Languages. In: Bunt. H., Tomita, M. (eds.): Recent Advances in Parsing Technologies, Kluwer, pp. 385-407, (1996)

123. Wittenburg, K.: Earley-style Parsing for Relational Languages. In Proceedings of the IEEE Workshop on Visual Languages, University of Washington, Seattle, pp. 192-199, (1992)

Appendix A

Recognition Knowledge Base Description Language

Terminals

The set of terminals includes literals (strings of characters in apostrophes), identifiers (strings of letters and digits starting with a letter), numbers (strings of digits with a potential '.' separator), and the new line character (NL).

Grammar

```
RKB              :=    Node NL NL RKB
                       | Expr NL RKB

Expr             :=    ExprName '=' ExprBody

ExprName         :=    '$' Identifier

Node             :=    ClassNode
                       | WholeNode
                       | TemplateNode

ClassNode        :=    NodeName '=' SubClasses NL Attributes
                       | NodeName '=' SubClasses

SubClasses       :=    NodeNamePrty
                       | NodeNamePrty '|' SubClasses

NodeNamePrty     :=    NodeName '(' Priority ')'
                       | NodeName

NodeName         :=    Identifier

Priority         :=    Number

Attributes       :=    'ATTRIBUTES' AttrDefs

AttrDefs         :=    AttrDef
                       | AttrDef NL AttrDefs

AttrDef          :=    AttrName '=' ExprBody
                       | 'FLAGS' '=' FlagList
```

```
                        | ‘LABEL’ ‘=’ Character
                        | ‘COLOUR’ ‘=’ Number‘,’Number‘,’Number

AttrName          :=    AttributeName
                        | ‘*’ AttributeName

AttributeName     :=    Identifier

WholeNode         :=    NodeName ‘=’ Parts NL CondSpecAttrs
                        | NodeName ‘=’ Parts


Parts             :=    NodeRef
                        | NodeRef ‘+’ Parts
                        | NodeRef ‘+=’ Parts
                        | NodeRef ‘+~’ Parts
                        | NodeRef ‘+&’ Parts

CondSpecAttrs     :=    Conditions NL Specs NL Attributes
                        | Conditions NL Attributes
                        | Specs NL Attributes
                        | Attributes

Conditions        :=    ‘WHERE’ CondExpr NL CondExprs
                        | ‘WHERE’ CondExpr

CondExprs         :=    ‘AND’ CondExpr NL CondExprs
                        | ‘AND’ CondExpr

Specs             :=    ‘SPECIALISATION OF’ SpecNodeList

SpecNodeList      :=    NodeName ‘,’ SpecNodeList
                        | NodeName

TemplateNode      :=    NodeName ‘=’ Template NL Attributes
                        | NodeName ‘=’ Template

Template          :=    ‘TEMPLATE’ Rect ‘:’ PixelList ‘:’ Ends

Rect              :=    Number ‘,’ Number ‘,’ Number ‘,’ Number

PixelList         :=    PixelValue
                        PixelValue ‘,’ PixelList

PixelValue        :=    Number
                        | ‘X’

Ends              :=    Corner ‘,’ Corner

Corner            :=    ‘RT’
                        | ‘RB’
                        | ‘LT’
                        | ‘LB’

ExprBody          :=    RelationalExpr
                        | RangeExpr
```

```
RelationalExpr     :=    FuzzyRelExpr
                         | BinaryRelExpr
FuzzyRelExpr       :=    BinaryRelExpr ‘~’ Number

BinaryRelExpr      :=    RangeExpr ‘in’ RangeExpr
                         | RangeExpr ‘=’ RangeExpr
                         | RangeExpr ‘<>’ RangeExpr
                         | RangeExpr ‘<’ RangeExpr
                         | RangeExpr ‘>’ RangeExpr
                         | RangeExpr ‘<=’ RangeExpr
                         | RangeExpr ‘>=’ RangeExpr
                         | RangeExpr ‘-=’ RangeExpr
                         | RangeExpr ‘<->’ RangeExpr
                         | RangeExpr ‘<<’ RangeExpr
                         | RangeExpr ‘>>’ RangeExpr

RangeExpr          :=    RangeExpr ‘|’ AdditiveExpr
                         | RangeExpr ‘&’ AdditiveExpr
                         | AdditiveExpr

AdditiveExpr       :=    AdditiveExpr ‘+’ MulExpr
                         | AdditiveExpr ‘-’ MulExpr
                         | AdditiveExpr ‘+^’ MulExpr
                         | AdditiveExpr ‘-^’ MulExpr
                         | MulExpr

MulExpr            :=    MulExpr ‘*’ UnaryExpr
                         | MulExpr ‘/’ UnaryExpr
                         | MulExpr ‘%’ UnaryExpr
                         | NegExpr

UnaryExpr          :=    ‘-’ FuncRangeExpr
                         | ‘!’ FuncRangeExpr
                         | FuncRangeExpr

FuncRangeExpr      :=    BuiltinFuncName ‘(’ ArgList ‘)’
                         | ‘[’ RangeExpr ‘,’ RangeExpr ‘]’
                         | PrimaryExpr

ArgList            :=    RangeExpr ‘,’ ArgList
                         | RangeExpr

PrimaryExpr        :=    Constant
                         | AttributeRef
                         | PartRef
                         | ‘$’ ExprName
                         | ‘(’ RangeExpr ‘)’

Constant           :=    Number
                         | ‘PI’

AttributRef        :=    PartRef ‘.’ AttributeName
                         | ContextAttrRef

ContextAttrRef     :=    ‘#’ NodeName ‘.’ AttributeName
```

```
PartRef          :=   '@@'
                      | '@' PartNumber

AttributeName    :=   Identifier

BuiltinFuncName  :=   'abs'
                      | 'avg'
                      | 'dist'
                      | 'connected'
                      | 'angle'
                      | 'min'
                      | 'max'
                      | 'delta'
                      | 'length'
                      | 'inside'
                      | 'close'
                      | 'above'
                      | 'left'
                      | 'sin'
                      | 'cos'
                      | 'tan'
                      | 'leftmost'
                      | 'rightmost'
                      | 'topmost'
                      | 'lowest'
                      | 'x'
                      | 'y'
                      | 'point'
```

Appendix B

Fragments of Recognition Knowledge Base for Handwriting Recognition

Straight line segments:

```
RV = TEMPLATE 0,0,2,1:1,0:rt,rb
       ATTRIBUTES flags = STRAIGHT

RVE1 = RVE(1.1) + RVE(1.1)
       ATTRIBUTES flags = STRAIGHT

RVE = RVE1 | RV
       ATTRIBUTES flags = STRAIGHT
              begAngle              = PI/2
              endAngle              = PI/2


LV = TEMPLATE -1,0,1,1:0,1:lb,lt
       ATTRIBUTES flags = STRAIGHT

LVE1 = LVE(1.1) + LVE(1.1)
       ATTRIBUTES flags = STRAIGHT

LVE = LVE1 | LV
       ATTRIBUTES flags = STRAIGHT
              begAngle              = -PI/2
              endAngle              = -PI/2

TH = TEMPLATE 0,0,1,2:1,0:lt,rt
       ATTRIBUTES flags = STRAIGHT

THE1 = THE(1.1) + THE(1.1)
       ATTRIBUTES flags = STRAIGHT

THE = THE1 | TH
       ATTRIBUTES flags = STRAIGHT
              begAngle              = PI
              endAngle              = PI

BH = TEMPLATE 0,-1,1,1:0,1:rb,lb
       ATTRIBUTES flags = STRAIGHT

BHE1 = BHE(1.1) + BHE(1.1)
       ATTRIBUTES flags = STRAIGHT

BHE = BHE1 | BH
       ATTRIBUTES flags = STRAIGHT
              begAngle              = 0.
              endAngle              = 0.
```

```
RT = THE(0.9) + RVE(0.9)
       WHERE  @0.w+@1.h>=@0.w*@1.h
       ATTRIBUTES flags = STRAIGHT


RB = RVE(0.9) + BHE(0.9)
       WHERE  @0.h+@1.w>=@0.h*@1.w
       ATTRIBUTES flags = STRAIGHT


LT = LVE(0.9) + THE(0.9)
       WHERE  @0.h+@1.w>=@0.h*@1.w
       ATTRIBUTES flags = STRAIGHT


LB = BHE(0.9) + LVE(0.9)
       WHERE  @0.w+@1.h>=@0.w*@1.h
       ATTRIBUTES flags = STRAIGHT


RR = RR1 | RR2
       ATTRIBUTES flags = STRAIGHT
              begAngle              = PI/2
              endAngle              = PI/2

RR1 = RT(1.6) + BH(0.8)
       ATTRIBUTES flags = STRAIGHT

RR2 = TH(0.8) + RB(1.6)
       ATTRIBUTES flags = STRAIGHT


LL = LL1 | LL2
       ATTRIBUTES flags = STRAIGHT

LL1 = BH(0.8) + LT(1.3)
       ATTRIBUTES flags = STRAIGHT

LL2 = LB(1.6) + TH(0.8)
       ATTRIBUTES flags = STRAIGHT


TT = TT1 | TT2
       ATTRIBUTES flags = STRAIGHT

TT1 = LV(0.8) + RT(1.6)
       ATTRIBUTES flags = STRAIGHT

TT2 = LT(1.6) + RV(0.8)
       ATTRIBUTES flags = STRAIGHT


BB = BB1 | BB2
       ATTRIBUTES flags = STRAIGHT

BB1 = RB(1.6) + LV(0.8)
       ATTRIBUTES flags = STRAIGHT

BB2 = RV(0.8) + LB(1.6)
       ATTRIBUTES flags = STRAIGHT

SL = SLS1(1.1) + SLS1(1.1)
       WHERE @0.endAngle = @1.begAngle
       ATTRIBUTES   begAngle      = @0.begAngle
                    endAngle      = @1.endAngle
```

```
SLS1 = RVE(0.3)|LVE(0.3)|THE(0.3)|BHE(0.3)|RT|RB|LT|LB|SL|RR|LL|TT|BB
       ATTRIBUTES flags = STRAIGHT
              begAngle                = angle(@@.beg,@@.end)
              endAngle                = @@.begAngle
              dphi                    = 0
              len                     = dist(@@.beg,@@.end)

SLS = SLS1(0.3)
```

Topological definitions (connected and closed):

```
$dphi = (@0.endAngle -^ @1.begAngle) + @0.dphi + @1.dphi

Conn0 = SLS(0.8) | Connected
       ATTRIBUTES dphi                = $dphi
              begAngle                = @0.begAngle
              endAngle                = @1.endAngle

Connected = Conn0 + Conn0
       ATTRIBUTES flags = TRACE
              len          = @0.len+@1.len
              begSeg       = @0.begSeg : @0.len
              endSeg       = @1.endSeg : @1.len
              prevBegAngle = @0.prevBegAngle : @1.begAngle
              prevEndAngle = @1.prevEndAngle : @0.endAngle

CntrX = Conn0(0.9) + Conn0(0.9)
       WHERE        @0.beg=@1.end
       AND          $dphi > 0
       SPECIALISATION OF Connected
       COMPLEMENTARY OF CntrO
       ATTRIBUTES flags = REGION, TRACE label = *
              len    = @0.len+@1.len

Dot = Conn0(0.9) + Conn0(0.9)
       WHERE  @@.len < 30
       AND          @@.w = @@.h ~ 1
       SPECIALISATION OF CntrX
       ATTRIBUTES flags = REGION, RESULT
       label = .
       colour = 200,255,144

CntrO = Conn0(0.9) + Conn0(0.9)
       WHERE        @0.beg=@1.end
       AND          $dphi < 0
       AND          @@.len > 8
       SPECIALISATION OF Connected
       ATTRIBUTES flags = REGION, TRACE
              len    = @0.len+@1.len
```

Curve primitives and Cusps:

```
$maxAngle = 0.32176        //atan(1/1) - atan(1/2) + eps

XCusp0 = SLS(0.8) | XCusp_quasi(1.1)

XCusp_quasi = XCusp0 + XCusp0
       WHERE  @0.endAngle -^ @1.begAngle >= $maxAngle
       SPECIALISATION OF Connected
       ATTRIBUTES flags = TRACE
```

```
XCusp = XCusp0 + XCusp0
       WHERE  $dphi > PI/4
       SPECIALISATION OF XCusp_quasi
       ATTRIBUTES flags = TRACE

OCusp0 = SLS(0.8) | OCusp_quasi(1.1)

OCusp_quasi = OCusp0 + OCusp0
       WHERE  @0.endAngle -^ @1.begAngle <= -$maxAngle
       SPECIALISATION OF Connected

OCusp = OCusp0 + OCusp0
       WHERE  $dphi < -PI/4
       SPECIALISATION OF OCusp_quasi
       ATTRIBUTES flags = TRACE


$sigma = 100.


LCL = SLS | LCL1

LCL1 = LCL(0.8) + LCL(0.8)
       WHERE  @0.endAngle -^ @1.begAngle in {$maxAngle}
       AND    $dphi/@@.len < 0.008
       SPECIALISATION OF Connected

CVX = SLS | CVX0

CVX0 = CVX1 | CVX2

CVX1 = CVX + CVX
       WHERE  @0.endAngle -^ @1.begAngle >= 0
       SPECIALISATION OF Connected
       ATTRIBUTES flags = TRACE

CVX2 = CVX0(0.5) + CVX0(0.5)
       WHERE  @0.endAngle -^ @1.begAngle < 0
       AND    @0.prevEndAngle -^ @1.prevBegAngle >= 0
       AND    @0.endSeg * @1.begSeg * sin(@1.begAngle -^ @0.endAngle) < 10
       SPECIALISATION OF Connected
       ATTRIBUTES flags = TRACE

CCV = SLS | CCV0

CCV0 = CCV1 | CCV2

CCV1 = CCV + CCV
       WHERE  @0.endAngle -^ @1.begAngle < 0
       SPECIALISATION OF Connected
       ATTRIBUTES flags = TRACE

CCV2 = CCV0(0.5) + CCV0(0.5)
       WHERE  @0.endAngle -^ @1.begAngle > 0
       AND    @0.prevEndAngle -^ @1.prevBegAngle < 0
       AND    @0.endSeg * @1.begSeg * sin(@0.endAngle -^ @1.begAngle) < 20
       SPECIALISATION OF Connected
       ATTRIBUTES flags = TRACE
```

Letter elements:

```
vpoll_up = VSTR1(0.1) += XCusp(0.1) += VSTR1(0.1)
       WHERE  @0.endAngle in [-3*PI/4,-PI/4]
       AND           @2.begAngle in [PI/4,3*PI/4]
```

```
        ATTRIBUTES top = point(avg(x(@0.end),x(@2.beg)),@1.t)
                begAngle                = @0.begAngle
                endAngle                = @2.endAngle
                dphi                    = $dphi
                len                     = @0.len+@1.len+@2.len

vpoll_down = VSTR1(0.1) += XCusp(0.1) += VSTR1(0.1)
        WHERE @0.endAngle in [PI/4,3*PI/4]
        AND         @2.begAngle in [-3*PI/4,-PI/4]
        ATTRIBUTES bottom = point(avg(x(@0.end),x(@2.beg)),@1.b)
                begAngle                = @0.begAngle
                endAngle                = @2.endAngle
                dphi                    = $dphi
                len                     = @0.len+@1.len+@2.len

vpoll_1 = vpoll_down += vpoll_up
        WHERE !connected(@0,@1,@0.beg,@1.end)
        ATTRIBUTES    top = @1.top
                bottom = @0.bottom

vpoll_2 = vpoll_up += vpoll_down
        ATTRIBUTES    top = @0.top
                bottom = @1.bottom

vpoll = vpoll_up += vpoll_down
        WHERE connected(@0,@1,@0.beg,@1.end)
        SPECIALISATION OF vpoll_2
        ATTRIBUTES    flags = REGION


//----------------------------------------------------

c = c_left += XCusp += c_right
        WHERE connected(@0,@1,@0.tr:@0.end,@1.beg)
        ATTRIBUTES flags = REGION, TRACE
                tl      = @0.tl
                bl      = @0.bl
                tr      = @2.end
                br      = @0.br:@0.beg

// ---------------------------------------------------

$epsilon = 0.0001

c_left = c_left_start | c_left_conn

c_left_0 = SLS(0.8) | c_left_(1.1)

c_left_ = c_left_0 + c_left_0
        WHERE @1.endAngle in [PI/2+$epsilon,-PI/3]
        AND         @0.begAngle in [PI/2-$epsilon,-PI/3]
        SPECIALISATION OF CVX0
        ATTRIBUTES top_right =
rightmost(@0.top_right:@0.beg,@0.end,@1.top_right:@1.beg,@1.end)
        flags = TRACE

c_left_top0 = c_left_top(1.2) | c_left_0

c_left_top = c_left_top0 + c_left_top0
        WHERE @1.endAngle in [PI/2+$epsilon,-3*PI/4]
        AND   @0.begAngle in [-3*PI/4,-PI/3]
        AND   above(@1.end,@0.beg)
        SPECIALISATION OF c_left_
        ATTRIBUTES top_right =
rightmost(@0.top_right:@0.beg,@0.end,@1.top_right:@1.beg,@1.end)
        flags = TRACE

c_left_top_conn = CCV0 += c_left_top
```

```
       ATTRIBUTES   top_right = @1.top_right
              cusp = @1.beg
              cuspAngle = @1.begAngle
       flags = TRACE

c_left_start = rtail_left(0.8) += c_left_top(0.8)
       WHERE        $dphi >= 3*PI/4
       AND          above(@1.end,@0.beg) ~ 1
       ATTRIBUTES top_right = @1.top_right

c_left_conn = c_left_top_conn +~ i_left
       WHERE  above(@0.cusp,@1.cusp) ~ 1
       AND    close(@0,@1,@0.cusp,@1.cusp) ~ 5
       ATTRIBUTES   top_right = @0.top_right
              tl     =      @0.beg
              bl     =      @1.end
              tr     =      @0.end
              br     =      @1.beg
       flags = REGION

//-----------------------------------------------------------

c_right_0 = SLS(0.8) | c_right_(1.1)

c_right_ = c_right_0 + c_right_0
       WHERE  @1.endAngle in [-PI/3,-2*PI/3]
       AND    @0.begAngle in [-PI/3,-2*PI/3]
       SPECIALISATION OF CCV0

c_right0 = c_right(1.2) | c_right_0

c_right = c_right0 + c_right0
       WHERE  @1.endAngle in [3*PI/4,-2*PI/3]
       AND         @0.begAngle in [-PI/3,PI/4]
       AND          $dphi <= -PI
       AND          above(@0.beg,@1.end) ~ 1
       SPECIALISATION OF c_right_

//---------------------------------------------------------

_c_right_0 = SLS(0.8) | _c_right_(1.1)

_c_right_ = _c_right_0 + _c_right_0
       WHERE  @1.endAngle in [-PI/3,-2*PI/3]
       AND    @0.begAngle in [-PI/3,-2*PI/3]
       SPECIALISATION OF CVX0

_c_right0 = _c_right(1.2) | _c_right_0

_c_right = _c_right0 + _c_right0
       WHERE  @1.endAngle in [-PI/3,PI/4]
       AND    @0.begAngle in [3*PI/4,-2*PI/3]
       AND    $dphi >= PI
       AND    above(@0.beg,@1.end) ~ 1
       SPECIALISATION OF _c_right_

//---------------------------------------------------------

_c_left_0 = SLS(0.8) | _c_left_(1.1)

_c_left_ = _c_left_0 + _c_left_0
       WHERE  @1.endAngle in [3*PI/4,PI/4]
       AND    @0.begAngle in [3*PI/4,PI/4]
       SPECIALISATION OF CCV0

_c_left0 = _c_left(1.2) | _c_left_0

_c_left = _c_left0 + _c_left0
```

```
      WHERE  @1.endAngle in [-PI/4,PI/4]
      AND    @0.begAngle in [3*PI/4,-3*PI/4]
      AND    $dphi <= -PI
      AND    above(@1.end,@0.beg)
      SPECIALISATION OF _c_left_

//--------------------------------------------------------

i = i_start

i_start = rtail_left += vpoll_up += rtail_right
      WHERE  dist(@0.bottom,@2.bottom) < 20
      ATTRIBUTES bottom = @0.bottom
             top = @1.top

i_right = rtail_right | i_right1 | i_right2 | i_right3

i_right1 = XCusp += rtail_right
      ATTRIBUTES bottom = @1.bottom

i_right2 = OCusp += XCusp += rtail_right
      ATTRIBUTES bottom = @2.bottom

i_right3 = OCusp += vpoll_up += rtail_right
      ATTRIBUTES bottom = @2.bottom


rtail_right_0 = SLS(0.8) | rtail_right_(1.1)

rtail_right_ = rtail_right_0 + rtail_right_0
      WHERE  @1.endAngle in [PI/3,-2*PI/3]
      AND    @0.begAngle in [PI/3,-2*PI/3]
      SPECIALISATION OF CCV0
      ATTRIBUTES bottom =
lowest(@0.bottom:@0.beg,@0.end,@1.bottom:@1.beg,@1.end)

rtail_right0 = rtail_right(1.2) | rtail_right_0

rtail_right = rtail_right0 + rtail_right0
      WHERE  @1.endAngle in [PI/2,-2*PI/3]
      AND    @0.begAngle in [PI/3,2*PI/3]
      AND    $dphi < 0
      SPECIALISATION OF rtail_right_
      ATTRIBUTES bottom =
lowest(@0.bottom:@0.beg,@0.end,@1.bottom:@1.beg,@1.end)

i_left = rtail_left += OCusp
      ATTRIBUTES bottom =
lowest(@0.bottom:@0.beg,@0.end,@1.bottom:@1.beg,@1.end)
             cusp         =      @0.end
             cuspAngle    =      @0.endAngle

rtail_left_0 = SLS(0.8) | rtail_left_(1.1)

rtail_left_ = rtail_left_0 + rtail_left_0
      WHERE  @1.endAngle in [-2*PI/3,PI/3]
      AND    @0.begAngle in [-2*PI/3,PI/3]
      SPECIALISATION OF CVX0
      ATTRIBUTES bottom =
lowest(@0.bottom:@0.beg,@0.end,@1.bottom:@1.beg,@1.end)

rtail_left0 = rtail_left(1.2) | rtail_left_0

rtail_left = rtail_left0 + rtail_left0
      WHERE  @1.endAngle in [-2*PI/3,-PI/6]
      AND    @0.begAngle in [-PI/2,PI/3]
      AND    $dphi > 0
      AND    left(@1.end,@0.beg)
```

```
       SPECIALISATION OF rtail_left_
       ATTRIBUTES bottom =
lowest(@0.bottom:@0.beg,@0.end,@1.bottom:@1.beg,@1.end)

//----------------------------------------------------------

ldesc = ldesc_tail

ldesc_tail = ldesc_right += XCusp += ldesc_left
       ATTRIBUTES bottom = @0.bottom


ldesc_left_0 = SLS(0.8) | ldesc_left_(1.1)

ldesc_left_ = ldesc_left_0 + ldesc_left_0
       WHERE  @1.endAngle in [PI/3,PI/6]
       AND    @0.begAngle in [PI/3,PI/6]
       SPECIALISATION OF CCV0
       ATTRIBUTES bottom =
lowest(@0.bottom:@0.beg,@0.end,@1.bottom:@1.beg,@1.end)

ldesc_left0 = ldesc_left(1.2) | ldesc_left_0

ldesc_left = ldesc_left0 + ldesc_left0
       WHERE  @1.endAngle in [-2*PI/3,-PI/3]
       AND    @0.begAngle in [PI/6,-2*PI/2]
       AND    $dphi < -PI/3
       AND    above(@1.end,@0.beg)
       AND    left(@0.beg,@1.end)
       SPECIALISATION OF ldesc_left_
       ATTRIBUTES bottom =
lowest(@0.bottom:@0.beg,@0.end,@1.bottom:@1.beg,@1.end)

ldesc_right_0 = SLS(0.8) | ldesc_right_(1.1)

ldesc_right_ = ldesc_right_0 + ldesc_right_0
       WHERE  @1.endAngle in [-5*PI/6,2*PI/3]
       AND    @0.begAngle in [-5*PI/6,2*PI/3]
       SPECIALISATION OF CVX0     ATTRIBUTES bottom =
lowest(@0.bottom:@0.beg,@0.end,@1.bottom:@1.beg,@1.end)

ldesc_right0 = ldesc_right(1.2) | ldesc_right_0

ldesc_right = ldesc_right0 + ldesc_right0
       WHERE  @1.endAngle in [-5*PI/6,PI/3]
       AND    @0.begAngle in [PI/3,2*PI/3]
       AND    $dphi >= PI/3
       AND    above(@0.beg,@1.end)
       AND    left(@1.end,@0.beg)
       SPECIALISATION OF ldesc_right_
       ATTRIBUTES bottom =
lowest(@0.bottom:@0.beg,@0.end,@1.bottom:@1.beg,@1.end)

//----------------------------------------------------------

nm_top_0 = SLS(0.8) | nm_top_(1.1)

nm_top_ = nm_top_0 + nm_top_0
       WHERE  @1.endAngle in [PI/3,-PI/2]
       AND           @0.begAngle in [PI/3,-PI/2]
       SPECIALISATION OF CVX0
       ATTRIBUTES top = topmost(@0.top:@0.beg,@0.end,@1.top:@1.beg,@1.end)

nm_top0 = nm_top(1.2) | nm_top_0

nm_top = nm_top0 + nm_top0
       WHERE  @1.endAngle in [PI/3,2*PI/3]
       AND    @0.begAngle in [-PI,-PI/2]
       AND   $dphi >= PI/2
       AND   left(@0.beg,@1.end)
       AND   above(@0.beg,@1.end)
       SPECIALISATION OF nm_top_
       ATTRIBUTES top = topmost(@0.top:@0.beg,@0.end,@1.top:@1.beg,@1.end)

nm_bottom_0 = SLS(0.8) | nm_bottom_(1.1)

nm_bottom_ = nm_bottom_0 + nm_bottom_0
       WHERE  @1.endAngle in [0,-PI/3]
       AND           @0.begAngle in [0,-PI/3]
       SPECIALISATION OF CCV0
       ATTRIBUTES top = topmost(@0.top:@0.beg,@0.end,@1.top:@1.beg,@1.end)

nm_bottom0 = nm_bottom(1.2) | nm_bottom_0

nm_bottom = nm_bottom0 + nm_bottom0
       WHERE  @1.endAngle in [0,PI/2]
       AND    @0.begAngle in [-2*PI/3,-PI/3]
       AND    $dphi <= -PI/2
       AND    left(@1.end,@0.beg)
       AND    above(@1.end,@0.beg)
       SPECIALISATION OF nm_bottom_
       ATTRIBUTES top = topmost(@0.top:@0.beg,@0.end,@1.top:@1.beg,@1.end)

//----------------------------------------------------------

ni_top = nm_top += i_right
       WHERE  above(@0.beg,@1.end)
       ATTRIBUTES top = @0.top
             bottom = @1.bottom

ni_bottom = i_left += nm_bottom
       WHERE above(@1.end,@0.beg)
       ATTRIBUTES top = @1.top
             bottom = @0.bottom

n_beg = vpoll_1 //| n_conn
```

Letters:

```
letter = L_a | L_b | L_c | L_d | L_e | L_f | L_g | L_h | L_i | L_j | L_k |
L_L_ | L_m | L_n | L_o | L_p | L_q | L_r | L_s | L_t | L_u | L_v | L_w | L_x
| L_y | L_z
       ATTRIBUTES   colour = 255,0,0
             baselineAngle = 0
             x_height             = @@.h * cos(baselineAngle)
             slant                = 0

//----------------------------------------------------------

Lad1 = i_left += c_left += i_right +& CntrO
       WHERE  connected(@0,@1,@0.end,@1.br:@1.beg)
       AND    connected(@1,@2,@1.tr:@1.end,@2.beg)
       AND    inside(@3,@0,@1,@2)
       ATTRIBUTES   flags = REGION, TRACE
             *tr = @2.end
             *tl = @1.tl
             *bl = @1.bl
             *br = @0.beg

Lad2 = c += i_conn
       WHERE  connected(@0,@1,@0.tr:@0.end,@1.tl)
       AND    connected(@0,@1,@0.br:@0.beg,@1.bl)
       AND    close(@0,@1,@0.top_right)          // -> not 'u'!
       ATTRIBUTES    flags = REGION, TRACE
              *tr = @1.tr
              *tl = @0.tl
              *bl = @0.bl
              *br = @1.br

//---------------------------------------------------

L_a = La1 | La2
       ATTRIBUTES    label = a

La1 = i_left += c_left += i_right +& CntrO
       WHERE  @1.h >= @2.h ~ $sigma
       SPECIALISATION OF Lad1
       ATTRIBUTES    flags = REGION, TRACE, RESULT


La2 = c += i_conn
       WHERE  @0.h >= @1.h ~ $sigma
       SPECIALISATION OF Lad2
       ATTRIBUTES    flags = REGION, TRACE, RESULT


//---------------------------------------------------


L_d = Ld1 | Ld2
       ATTRIBUTES    label = d


Ld1 = i_left += c_left += i_right +& CntrO
       WHERE  @1.h*1.5 < @2.h ~ $sigma
       SPECIALISATION OF Lad1
       ATTRIBUTES    flags = REGION, TRACE, RESULT

Ld2 = c += i_conn
       WHERE  @0.h*1.5 < @1.h ~ $sigma
       SPECIALISATION OF Lad2
       ATTRIBUTES    flags = REGION, TRACE, RESULT
```

Words:

```
word = letter | connected_word
       ATTRIBUTES tl                 = @0.tl
              bl                     = @0.bl
              tr                     = @1.tr
              br                     = @1.br
              baseline               = @0.baseline
              baselineAngle =
avg(@0.baselineAngle,@1.baselineAngle,angle(@1.baseline,@0.baseline))
              x_height               = avg(@0.x_height,@1.x_height)
              slant                  = avg(@0.slant,@1.slant)

connected_word = word += word
       WHERE  connected(@0,@1,@0.tr:@0.end,@1.tl)
       AND    connected(@0,@1,@0.br:@0.beg,@1.bl)
       AND    @0.baselineAngle = @1.baselineAngle ~ $sigma
       AND    angle(@1.baseline,@0.baseline) = @0.baselineAngle ~ $sigma
       AND    @0.x_height = @1.x_height ~ 1
       ATTRIBUTES    flags = REGION, TRACE
```